Progress in Artificial Intelligence (2021) 10:83–97
https://doi.org/10.1007/s13748-020-00225-z

REGULAR PAPER

A synchronous deep reinforcement learning model for automated multi-stock trading

Rasha AbdelKawy[1] · Walid M. Abdelmoez[2] · Amin Shoukry[3,4]

Received: 28 April 2020 / Accepted: 16 November 2020 / Published online: 5 January 2021

Abstract
Automated trading is one of the research areas that has benefited from the recent success of deep reinforcement learning (DRL) in solving complex decision-making problems. Despite the large number of researches done, casting the stock trading problem in a DRL framework still remains an open research area due to many reasons, including dynamic extraction of financial data features instead of handcrafted features, applying a scalable DRL technique that can benefit from the huge historical trading data available within a reasonable time frame and adopting an efficient trading strategy. In this paper, a novel multi-stock trading model is presented, based on free-model synchronous multi-agent deep reinforcement learning, which is able to interact with the trading market and to capture the financial market dynamics. The model can be executed on a standard personal computer with multiple core CPU or a GPU in a convenient time frame. The superiority of the proposed model is verified on datasets of different characteristics from the American stock market with huge historical trading data.

Keywords Automated trading · Deep reinforcement learning · Synchronous parallel multiple agents · Deep belief network (DBN) · Long short-term memory (LSTM)

1 Introduction

The worldwide financial crisis of 2011 prompted the usage of automated technologies to help investors take the right investment decisions in the stock market. Automated or algorithmic trading is the expression used when using a computer program to execute and monitor trades orders. Traditional algorithmic trading models were built based on mathematical and financial theories that rely on human experts to understand and apply.

Time series analysis was used in [1] to investigate trading behavior and market organization, while stock price predictions were investigated in [2]. Next, machine learning techniques were used combined with human expert knowledge (supervised algorithms) to build more complex models to support traders' decision-making processes [3]. For example, support vector machine (SVM) technique was used in [4] to classify the stock market assets, and the genetic algorithm (GA) has been applied to select the best trading assets.

Deep learning (DL) techniques have been excessively used in financial stock trading market. In [5] the Chinese stock market stability was studied, and the stock price prediction was modeled, through an ensemble of multilayer feed-forward networks, while in [6] the study predicted both the stock price movement and its interval of growth (or decline) rate within the predefined prediction duration. In [7] an analysis of using DL techniques in the stock market prediction was provided, mentioning both the advantages and drawbacks of each technique. The recent work in [8] predicts the future price of the stock indices using the financial data of individual companies to train the neural networks. Recent

✉ Rasha AbdelKawy
Rasha.Shokry@student.aast.edu

Walid M. Abdelmoez
walid.abdelmoez@aast.edu

Amin Shoukry
amin.shoukry@ejust.edu.eg

1 Computer Science Department, College of Computing and Information Technology, Arab Academy for Science Technology and Maritime transport, Alexandria, Egypt

2 Software Engineering Department, College of Computing and Information Technology, Arab Academy for Science Technology and Maritime Transport, Alexandria, Egypt

3 Computer Science and Engineering Department, Egypt-Japan University of Science and Technology, Alexandria 21934, Egypt

4 Computer and Systems Engineering Department, Faculty of Engineering, Alexandria University, Alexandria 21544, Egypt

progress on using DL techniques in stock market prediction is presented by Jiang in [10] where more than 100 articles, published during the last three years, that cover data collection, processing analysis, model building, and model evaluation, are cited. Murat et al. [11] provide a state-of-the-art survey for the DL models developed for financial applications and identify possible future applications for these models.

Despite the success of DL in financial stock market prediction (e.g., [7,8,10,11]), its role in stock trading has been limited to decision-making support, as the techniques used are only able to forecast the market volatility, predict the prices movement, and predict the next trend, without offering a direct trading action such as buy/hold/sell of a certain amount of trading stocks' shares. To use DL in stock trading, e.g., [9] , the model needs both a DL module that predicts the price of the stock, and a trading module (extra logic layer) that takes the trading action (buy/hold/ sell) of specified amounts of shares. Moreover, the efficiency of the DL trading model relies on both the DL module and the trading module since the prediction may be good, but the trading decision logic is inaccurate due to the high volatility of the market. Also, the risk factor and the transaction cost are difficult to handle in the DL trading model.

On the other hand reinforcement learning rose as a distinguished technique that enables an agent to learn the right behavior directly from interaction with the environment through trial and error without the need of any domain or expert knowledge. The unprecedented success of the deep reinforcement learning technique (DRL) in playing games [12,15,29,30], e.g., the success of Silver et al. in 2017 [12], to beat the world champion in the Chinese game "Go" motivated other researchers to apply this technique in other domains, especially in dynamic decision making for stock trading.

In this paper, a DRL model is presented for multiple stocks trading. It uses two different types of deep neural networks: the deep belief network (DBN) and the long short-term memory (LSTM) network. Both types of networks proved to be excellent in its domain of specialization. The DBN is a feature extraction network that discovers the complex regularities in the data while being able to reduce its dimensionality, while the LSTM network is suitable for making predictions on long time series data. The proposed model includes several agents, which learn according to a synchronized reinforcement technique. The end-to-end learning of the DBN and LSTM DRL models proved to be efficient in optimizing the trading decisions of the users. Our contributions in this research can be summarized in the following points:

- Using the deep belief network (DBN) to extract the financial data regularities and reducing its dimensionality. To the best of our knowledge, this is the first time to use DBN in a DRL model.
- Extending the synchronous multi-agent deep reinforcement algorithms described in [30] and utilizing LSTM networks as the actor-critic controller functions for both policy-based and value-based algorithms.
- Designing flexible action-domain and the associated reward calculations to fit the nature of different (policy-based/ value-based) reinforcement learning algorithms.
- Conducting intensive experiments on using different reinforcement learning techniques with different settings in order to get insight into the best model to use.

The remaining of this paper is organized as follows: Sect. 2 gives a technical background about the deep learning networks and the reinforcement learning techniques in addition to an overview of using the deep reinforcement learning techniques in the stock trading domain. Section 3 presents the proposed deep reinforcement learning model architecture. Section 4 covers the experiments performed with the proposed model and its performance evaluation. Finally, Sect. 5 summarizes the conclusions and possible future work.

2 Background

2.1 Deep learning techniques

Artificial neural networks (NNs) are connectionist systems consisting of a large number of neurons, usually organized in layers, as well as connections between them. They aim at mimicking the human brain structure and problem-solving capabilities and proved to be efficient in solving regression, classification and prediction tasks. For decades, these problems have been solved using feed-forward NNs consisting of one hidden layer (shallow NNs), usually trained with supervised gradient-descent learning algorithms. Later, due to the success in overcoming the difficulties (e.g., vanishing gradients) associated with training networks with many layers and the increase in the computational power (availability of GPUs), deep learning emerged and became the dominant learning paradigm to train deep NNs having millions of parameters to solve problems in many areas including computer vision, robotics, healthcare and automation [16].

2.1.1 Restricted Boltzmann machine

The restricted Boltzmann machine (RBM) is a shallow, two-layer NN that learns to approximate/reconstruct the input data in an unsupervised way. Its topology corresponds to a symmetrical bipartite graph with a set/layer of visible nodes that receives the data, while the other set/layer encodes the features extracted from the visible layer with the constraint that nodes in the same layer are conditionally independent. RBMs constitute the building blocks of deep belief networks.

2.1.2 Deep belief networks

A deep belief network (DBN) is a special class of deep NN, It is a probabilistic generative model composed of simple stacked auto-encoders or RBMs. The hidden layers in a DBN has the following joint distribution:

$$P(x, h_1, h_2, \ldots, h_l) = P(x|h_1)P(h_1|h_2)\ldots P(h_{l-1}|h_l) \quad (1)$$

where x is the input to the DBN and h_i is the hidden layer at level i. Based on the above equation, Hinton et al. [17] introduced a fast and efficient unsupervised learning algorithm that trains each layer in a DBN separately.

Layer-wise greedy unsupervised training Bengio et al. [18] presented an effective greedy layer-by-layer training technique for a DBN (composed of stacked RBMs) that can be summarized as follows:

- At each layer i the visible layer v_i receives the input vector and maps it to the hidden layer (h_i):

$$h_i(v_i) = \sigma(W_i v_i + b_i) \quad (2)$$

  where W_i is the weight of the hidden layer, and b_i is the bias vector of the hidden layer.
- The next layer reconstructs the input vector by mapping the hidden vector data $h_i(v_i)$ to the next visible layer

$$v_{i+1} = \sigma(W_j h_i(v_i) + c_j) \quad (3)$$

  where the sigmoid function σ is defined as

$$\sigma(t) = \frac{1}{1 + e^{-t}} \quad (4)$$

- The optimization function is minimizing the error between the input vector to the ith layer $v_{(i)}$ and its reconstruction $v_{(i+1)}$, formulated as:

$$Argmin_{Wi,Wj,bi,cj}[J] = Argmin_{Wi,Wj,bi,cj}\left[\frac{1}{2}\sum_{k=1}^{n}\|v_i - v_{i+1}\|^2\right] \quad (5)$$

  where J is one-half squared of the error cost function, and n is the size of the training dataset. The greedy unsupervised training technique applies the minimization function to each layer separately, starting from the first layer to learn its parameters $(w_1, w_{j_1}, b_1, c_{j_1})$ using the training dataset. Next, it freezes the learned parameters of the first layer and uses its output data (v_{i+1}) as the training dataset for the second layer and proceeds recursively for as many layers as there exist in the DBN.

Supervised fine-tuning training As the weights of each RBM are learned independently of the next layer weights, the DBN needs a supervised training phase in which the back-propagation technique is applied to fine-tune all the DBN layers using the unsupervised trained weights as initial weights.

2.1.3 Long short-term memory (LSTM) networks

Unlike the other NNs, which handle the input data as independent points, the recurrent neural network (RNN) was designed to deal with the dependencies in sequential input data. By adding its hidden state to the input data to generate the output, as in Fig. 1, the output state becomes dependent on both the current and previous inputs. Figure 1 shows the unfolded architecture of a RNN, where x and o are the input data and output data, U, W and V are the weights associated with the input, output and the hidden state, respectively.

The LSTM network is a special kind of RNN, which has the capability to remember long-term temporal dependencies in the input sequence while avoiding exploding and vanishing gradient problems encountered in the training of normal RNNs. An LSTM layer consists of concurrent units, where each unit is composed of four parts, a cell c, which plays the role of memory of input values over time intervals, an input gate i, an output gate o, and a forget gate f, which regulate the flow of information into and out of the cell.

The LSTM architecture shown in Fig. 2 is inspired by [13], and the following equations describe how the LSTM units' gates are updated:

1. Forget gate's activation vector equation:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \quad (6)$$

2. Input/update gate's activation vector equation:

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \quad (7)$$

3. Output gate's activation vector equation:

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \quad (8)$$

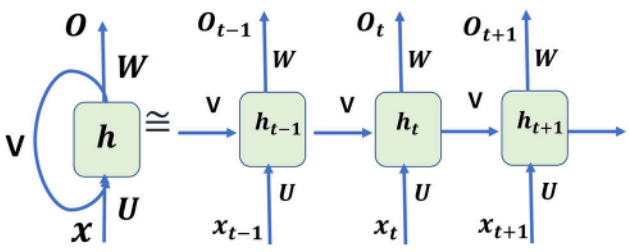
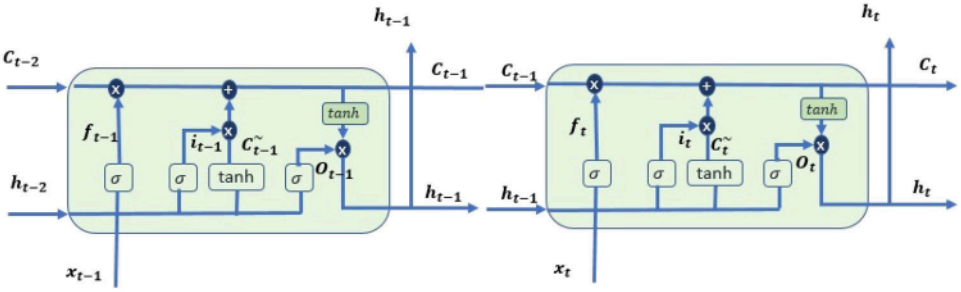


Fig. 1 Unfolded recurrent neural network

Fig. 2 LSTM units architecture



4. Cell input activation vector equation:

$$c_t^{\sim} = \sigma_g(W_c x_t + U_c h_{t-1} + b_c) \tag{9}$$

5. Cell state vector equation:

$$c_t = f_t \otimes c_{t-1} + i_t \otimes c_t^{\sim} \tag{10}$$

6. Hidden state vector which is the value of memory cell and an output vector of the LSTM unit:

$$h_t = o_t \otimes \sigma_h(c_t) \tag{11}$$

where

- x_t is the input vector to the LSTM at time t.
- (W_f, U_f),(W_i, U_i), (W_o,U_o),(W_c,U_c) are the weight matrices for the forget gate, input gate, output gate, and cell, respectively.
- b_f, b_i, b_o, b_c are bias vectors for the forget gate, input gate, output gate and cell, respectively.
- σ_g is the gate activation function (sigmoid function).
- $\otimes$ refers to element-wise product.

2.2 Reinforcement learning techniques

Standard reinforcement learning problem is modeled as a Markov decision process (MDP), with state space S, action space A, and reward domain $S \times A \rightarrow R$, where an agent interacts with an environment ϵ over a finite set of time intervals in order to maximize its reward. At each time step t the agent receives a state $s_t \in S$ from the environment ϵ and selects an action $a_t \in A$ that achieves the maximum expected reward R_t. In return, the agent receives the next State s_{t+1} and a scalar reward r_t. The maximum expected reward R_t is the accumulated reward received from time step t to the end of the process and calculated by:

$$R_t = \sum_{k=1}^{\infty} \gamma^k r_{t+k} \tag{12}$$

where $\gamma \in (0, 1]$ is a discount factor that represents the priority of long-term rewards compared to short-term rewards. γ is set closer to 1, if the agent cares about long-term rewards, while it is set closer to zero, if the agent cares more about immediate rewards. The process continues until the agent reaches a terminal state, or the time intervals ends, after which the whole process is restarted. The free-model reinforcement learning algorithms can be categorized in two types: value-based and policy-based algorithms.

2.2.1 Value-based algorithms

In value-based algorithms, also called critic-based reinforcement learning algorithms, an agent calculates a value function, namely action value $Q(s_t, a)$, for the current state and greedily chooses the next action that results in the maximum expected return. The simplest and most popular algorithm in this category is the Q-learning algorithm [14] which is based on the iterative Bellman equation:

$$Q^*(s_t, a_t) = E_{s_t, a_t \sim \epsilon}[r + \gamma \max_{a_{t+1}}(Q^*(s_{t+1}, a_{t+1})|s_t, a_t)] \tag{13}$$

action-value function $Q^*(s_{t+1}, a_{t+1})$ is known for all possible next states s_{t+1} and all possible actions a_{t+1}, then the optimal action value for current state s_t is the action a_t that gives the maximum accumulative reward. With the adoption of NN, with weight vector parameter (θ), as a nonlinear function used to estimate the action-value function, the deep Q-learning (DQN) technique achieved a great success in playing video games [15,29]. The deep Q-network (DQN) is trained by minimizing a sequence of loss functions $L_t(\theta_t)$ that changes at each iteration t:

$$L_t(\theta_t) = E_{s_t, a_t \sim \epsilon}[(y_t - Q(s_t, a_t; \theta_t))^2] \tag{14}$$

where y_t is the target for iteration t, calculated as:

$$y_t = E_{s_t, a_t \sim \epsilon}[(r + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})|s_t, a_t] \tag{15}$$

where s_{t+1} is the state encountered after state s_t.

Performing the gradient procedure using the differentiating equation of the loss function with respect to the neural network weights gives as follows:

$$\begin{aligned} \nabla_{\theta_i} L_t(\theta_t) = & E_{s_t, a_t, s_{t+1}, a_{t+1} \sim \epsilon}[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_{t-1}) \\ & - Q(s_t, a_t; \theta_t) \nabla_{\theta_t} Q(s_t, a_t; \theta_t)] \end{aligned} \tag{16}$$

The DQN algorithm uses experience replay to overcome the nonstationarity of the learning process. Also, it performs network updates by accumulating gradients over several time steps (similar to mini-batches) to reduce the correlation between these updates. Several variants of the DQN have been presented [19–21] to make it more scalable and efficient to the extent of surpassing human level in playing Atari 2600 benchmark. This necessitated that the output action space be discrete and of low dimensionality in order to reduce the computation complexity.

2.2.2 Policy-based algorithms

In policy-based algorithms, also called actor-based reinforcement learning algorithms, the agent searches for an optimal policy that maximizes the expected return. A policy $\pi : S \rightarrow \rho(A)$ maps states S to a probability distribution over actions $\rho(A)$. The return from a state, namely state function $V^{\pi}(s)$, is the expected discounted rewards from state s after following policy π, calculated as:

$$V^{\pi}(S) = E(R_t|s_t = s) \tag{17}$$

The agent adopts a policy π and continually adjusts it to be equal to or better than any other policy. The policy gradient technique is applied on both stochastic and deterministic policies, with the addition of an entropy regularization term to ensure that the policy keeps exploring:

$$\pi^{\sim}(s_t) = \pi(s_t|\omega_t^{\pi}) + N \tag{18}$$

The exploration policy $\pi^{\sim}$ is constructed by adding noise sample from a noise process N to the actor policy π. In DRL, a NN is used to approximate the policy parameters π_{ω}. Based on the policy gradient theorem [22], this can be done by continually adjusting the policy parameters in the direction of performance gradient. In DRL policy-based algorithms, the action value $Q^{\pi}(s_t, a_t)$ is the expected discounted return after taking an action a_t following policy π in a state s_t, and calculated as:

$$\begin{aligned} Q^{\pi}(s_t, a_t) =& E_{s_t, a_t \sim \pi}[r(s_t + a_t) \\ &+ E_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]] \end{aligned} \tag{19}$$

Then, policy gradient equation is calculated as:

$$\nabla_{\omega} V^{\pi}(s) = E_{s_t \sim \pi}[\nabla_{\omega} \pi \nabla_a (Q^{\pi}(s, a))|_{a=\pi(s)}] \tag{20}$$

The above equation relies on two components($\nabla_a (Q^{\pi}(s, a))$ and $\nabla_{\omega} \pi$) which usually require an actor-critic architecture. *Actor-critic architecture* The actor-critic architecture consists of two separate components: an actor and a critic. The actor updates the policy parameters π_{ω} in the direction suggested by the critic, while the critic calculates a value function $Q^{\pi_{\omega}}(s, a)$ to guide the actor through the policy-learning process. Therefore, the actor depends on the function value generated by the critic to update the policy parameters in the right direction, and the critic takes the feedback from the effect of the actor decision on the environment to calculate a new value function. A lot of policy gradient actor-critic deep RL algorithms were proposed in recent years, for examples: DDPG [23], TRPO [24], PPO [25], and ACER [26].

2.2.3 Asynchronous Advantage Actor-Critic (A3C):

A3C [30] is a parallel policy gradient DRL algorithm where the critic learns the value function, while multiple actor agent/learners, running in parallel, explore different parts of the environment. Each actor learner can run different exploration policies in different threads. It shares its learned policy parameters with a central critic which in turn updates the actors' policy parameters with the central policy parameters. The k-step learning algorithm is used where the updates of the actors' policy parameters are accumulated for k-steps and used to update the critic parameters in a single gradient step.

2.2.3.1 Synchronous Advantage Actor-Critic (A2C):

A2C is a synchronized version of the A3C algorithm [30],"A2C" with the first "A" ("asynchronous") removed, where multiple agents are trained in a synchronized way. At each iteration, a coordinator in A2C waits until all agents update the global policy parameters. This ensures that at the next iteration, all agents work on exploring the same exploration policy while possibly exploring different parts of the environment. This modification allowed the A2C algorithm to converge faster than the A3C algorithm, while improving its performance. Both A3C and A2C algorithms offer practical benefits since training of the agents can be done in parallel and efficiently on a multi-core CPU or a GPU.

2.3 Deep reinforcement learning techniques in automatic stock trading: background

The remarkable work of Moody et al. [31], which combined reinforcement learning with recurrent NNs, namely recurrent reinforcement learning (RRL), and proposed the risk-aversion measurement, namely sharp ratio (SR) as the performance measurement, has been considered as a pioneer research in automatic trading. Deng et al. [32] were the first to use a convolutional NN (CNN) to extract features from the input financial information before applying a modified RRL technique. Almahdi et al. [33] presented a deep recurrent reinforcement learning (DRRL) method for portfolio management using a statistically coherent downside

risk-adjusted performance objective (Sortino Ratios), which is calculated using profits and volatilities. Jiang et al. [34] presented a DRL model that deals with multiple assets trading. Their model used three different kinds of NNs (CNN, LSTM, and basic RRN), each having an equal vote in predicting the trading decision. Liang et al. in [40] investigated deep deterministic policy gradient (DDPG) [23], proximal policy optimization (PPO) [25] and policy gradient (PG) [22] algorithms for portfolio management. Their experiments included different settings, such as different feature combinations, objective functions and learning rates. The "DDPG" technique was evaluated in [41] for automatic multiple assets' trading and construction of a risk-aversion trading portfolio. However, [42] proposed a DRL model containing two deep learning layers: The first layer is a group of LSTM units, one per each trading asset, to extract the historical price information for that asset. The second layer, namely CAAN, used an attention mechanism to model the inter-relationships among trading stocks. In [43], the A3C RL algorithm was combined with LSTM deep neural networks using the value-based Q-networks (DQN) and "stacked denoising auto-encoders" (SDAEs) noise generation technique to increase the policy exploration and, consequently, the robustness of the obtained solution. The output of that model gives the trading position for only one asset at a time. In [44] researchers developed a DRL framework, namely "Deep-Breath," that consists of a restricted stacked auto-encoder and a convolutional neural network (CNN) in order to continuously reallocate funds in portfolio management, while [45] proposed an action-specialized ensemble expert trading system using the discrete action space DRL technique. For more information about DRL usage in financial market the reader can refer to [46,47].

3 Proposed deep reinforcement learning model

The architecture of the proposed DRL model is shown in Fig. 3. It is specialized in multi-stocks trading. Per each trading day, the model is automatically fed with the data received from the financial market, depending on these data the system generates a trading decision (buy/hold/sell) for the given multi-stocks. Then, the reward of the trading decision is calculated and the back-propagation learning technique is applied to update the weights of the model's NNs.

The model consists of two components:

1. A DBN that extracts discriminant features of reduced dimensionality from the high-dimensional raw financial data,
2. A multi-agent DRL module that is fed with the DBN features to generate the trading decisions.

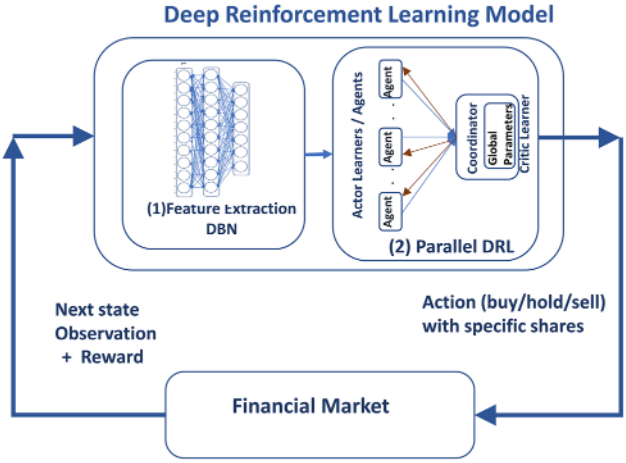


Fig. 3 Proposed deep reinforcement learning model architecture

The offline training phase of the model is based on the following assumptions:

- The trading decision is executed immediately with no change in the stocks' prices,
- The system's trading decisions do not affect the stocks' prices in the financial market. This assumption is fulfilled by using a small amount of trading money.

3.1 Raw financial data

The following are the types of financial data used:

1. *OHLCV data* That includes the opening price, highest price, lowest price, closing price, and the trading volume of the stock.
2. *Technical indicators* Which correspond to equations applied on the OHLCV data for a stock to obtain indications about its future position in the market, e.g., stock price trend/oscillation/volatility/moving average. The open-source technical analysis software library "TA_LIB" [1], provides more than 200 stock market technical indicators classified into seven different categories. It has been used in our model to calculate the technical indicators chosen from the library.
3. *Financial market indices* Which correspond to the average of the OHLCV data for a group of stocks, that is used to get an indication about the whole market direction, e.g., the "Dow Jones Industrial Average" which contains 30 industrial stocks in the USA market.

The above three types of financial data are fed to the system on a daily basis.

[1] https://goldenjumper.wordpress.com/tag/ta-lib/.

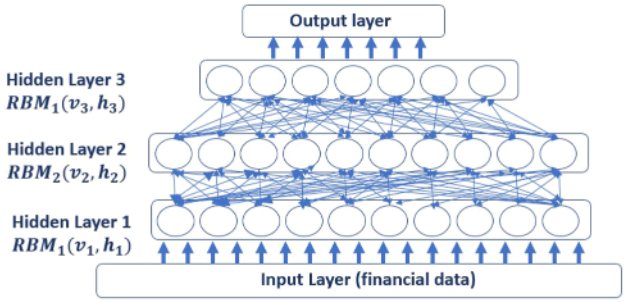


Fig. 4 DBN with three hidden RBM layers and an output layer

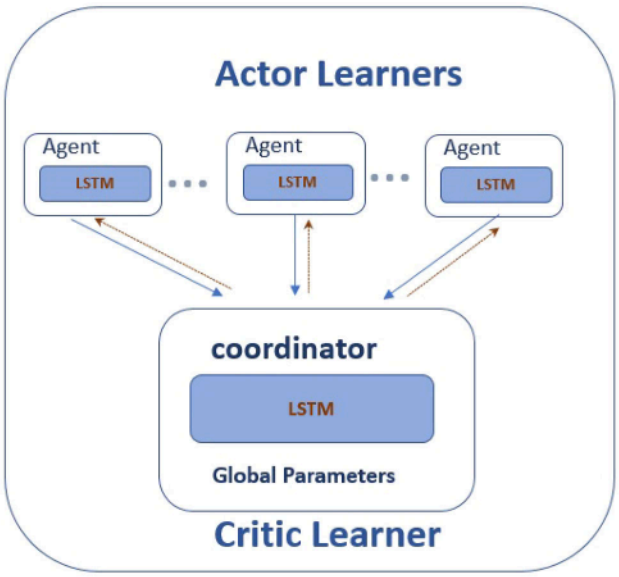


Fig. 5 DRL module architecture with learning actors and critic

3.2 The DBN network

This part of the model is responsible of extracting highly uncorrelated features from the financial data market. The DBN network shown in Fig. 4 has 125 inputs that go through three RBM hidden layers, with (100, 80, 60) neurons, respectively, and an output layer with 40 neurons.

3.3 The deep reinforcement learning module

The second part, in the model, is the DRL module, illustrated in Fig. 5. It is based on the synchronous parallel DRL algorithm proposed in [30]. As explained in Sect. 2.2, the synchronous parallel DRL technique relies on a set of actors' learners (agents) in addition to a central critic learner. The critic learner updates its global parameters according to the information received from each actor's parameters and provides all actors with the final version of the global parameters. The LSTM network type is chosen to be the deep neural network controller for both the learning actors and critic. The LSTM networks have the capability to learn long-term dependencies in the sequential financial data, memorize the agent's decisions, and prevent unwanted oscillation in trading by avoiding unnecessary transaction cost (TC) due to frequent buying/selling decisions. In our research, we implemented the synchronized parallel DRL with two underlying different reinforcement learning techniques.

- First is the policy-based technique, called synchronous advantage actor-critic (A2C) [30]. Each actor controller function is a three-layer network, and the first layer is a feed-forward layer consisting of 40 neurons that receives its input from the DBN network. The second layer consists of 128 LSTM units, as described in 2.1, while the output layer has five neurons each generating a continuous value in [-1..1]. The critic controller function is also a three-layer 5-128-1 feed-forward network that receives its five inputs from the actor network, its hidden layer has 128 LSTM units and its output has one neuron to calculate the value function.
- Second is the value-based technique, called multi-step Q-learning algorithm [30]. Each actor controller function is a three-layer network, similar to the policy-based actor controller but with an output layer consisting of 21 neurons. Each neuron represents one value from $\{-1, -0.9, -0.8, \ldots, -0.1, 0, 0.1, \ldots, 1\}$ according to their order in the layer, respectively. A softmax activation function is used with the output layer to ensure that one value is generated at a time. The critic controller is similar to the policy-based technique critic controller but with the first layer consisting of 21 neurons.

3.4 The action-domain Design

The action domain varies according to the RL technique used. Value-based technique generates discrete actions, while policy-based technique generates continuous actions, as explained below:

3.4.1 Value-based RL (DQN) technique:

The value-based model trades one stock at a time and the output is a discrete value belonging to the set $\{-1, -0.9, -0.8, \ldots, -0.1, 0, 0.1, \ldots, 1\}$ of 21 possible actions. A negative, positive or zero value corresponds to a selling, buying or no action, respectively. For example, the action "+0.9" means buying with 90% of the available cash, while action "−0.5" indicates selling 50% of the available stock. In case the process is inapplicable, e.g., the action is buying, but there is no available cash, or the action is selling, while no asset exists, then no operation is executed by the agent. Note that due to the curse of dimensionality, the system trades only one stock at a time, because if m different actions can be taken per stock, then for n stocks, the cardinality of the action domain

becomes n^m. If the number of actions (m) is decreased to increase the number of traded stocks, the accuracy is affected, e.g., decreasing the actions available per stock to three values {-1,0,1}, and trading five stocks generates $5^3 (= 125)$ discrete actions values that will negatively affect the profit of the proposed design and increase its time complexity.

3.4.2 Policy-based RL (DDPG) technique

The policy-based model generates continuous actions each lying in the interval $[-1..1]$, where the agent can buy/sell any portion of an asset using the same rules of the discrete action process, e.g., action "+0.145" means buying the desired stock with 14.5% of the available cash, and action "-0.2" means selling 20% of the available asset. Since the model trade multi-stocks, the system output is a vector in $[-1..1]^n$, where n denotes the number of assets in the trading process. So, each value in the output vector represents the required action for the corresponding stock. The selling processes are executed first to get more cash for trading, followed by the buying processes. In case the cash required for the buying processes is less than 100% of the available cash, the buying actions are executed in order. Otherwise, the buying amounts are first normalized and then executed. Finally, in case the available cash is not enough the buying process is not performed. For example, given the action vector $[0, 0.9, -0.2, -0.4, 0.6]$, the agent will sell 20% and 40% of the stocks in the third and fourth dimensions, respectively, to augment the cash available for the buying processes. Since the cash required for buying represents 150% of the available cash, the buying amounts are normalized to 60% and 40%). Accordingly, the agent will buy the second and fifth stocks with 60% and 40% of the available cash, respectively. In case the available cash to perform the buying process is not enough to complete the buying process, it is not executed. A transaction cost is considered when the system measures the applicability of its action. It is calculated as 0.5% of the number of traded shares with a maximum of 1% of the trading value.

3.5 Reward calculation

Assuming n, $S_t \in R^n$ *and* $P_t \in R^n$, Avc_t denote the number of trading stocks, the amount of stocks possessed by the system, the stock-closing price and , available cash at time t, respectively, it follows that the total portfolio value V_t is equal to:

$$V_t = Avc_t + S_t \times P_t \tag{21}$$

The profit and loss PL_t at time t is calculated as the difference between the portfolio values at times t and t-1:

$$PL_t = V_t - V_{t-1} \tag{22}$$

The sharp ratio (SR_t) at time t is calculated as the average return earned above the risk-free return (FR_t), for the same time interval, per the volatility of the earned return (the volatility is represented by the Average Deviation of the profit and loss during the time interval δ). The risk-free return is measured by the return of the American treasury bills FR_t:

$$SR_t = \frac{\sum_{t\in 1}^{T}(PL_t - FR_t)}{\delta PL_t)} \tag{23}$$

4 Experiments

4.1 Datasets used

The American stock market is one of the oldest financial markets, and it is matured and regulated which makes it a good baseline financial market for our experiments. In order to examine our model with different kinds of stocks in various industrial areas, two different datasets have been used. Each dataset is composed of five stocks chosen randomly from the oldest industrial American companies. Tables 1 and 2 contain the names of the stocks used in the first and second datasets, respectively. Table 3 contains the names of the largest five American-stock-market's indices that are used in our experiments. Both datasets and financial market indices are available online at "Yahoo Finance." The datasets for the past forty years from January 1980 to January 2020 have been downloaded.

Table 1 First trading stocks dataset

ID	Mnemonic	Name
1	JNJ	Johnson & Johnson
2	WMT	Walmart
3	MCD	McDonald's Corporation
4	F	Ford Motor Company
5	MRK	Merck & Co., Inc.

Table 2 Second trading stocks dataset

ID	Mnemonic	Name
1	MMM	3M Company
2	TM	Toyota Motor Corporation
3	IBM	International Business Machines Corporation
4	UTX	United Technologies Corporation
5	PG	The Procter & Gamble Company

Table 3 Description of the financial market indices

ID	Mnemonic	Name	Definition
1	^GSPC	S&P 500	Measures the stock performance of 500 large companies listed on stock exchanges in the USA
2	^DJI	Dow Jones Industrial Average	Measures the stock performance of 30 large companies listed on stock exchanges in the USA
3	^IXIC	NASDAQ Composite	Third most followed index in US stock market, heavily weighted toward information technology companies
4	^NYAC	NYSE Composite	Covers all common stocks listed on the New York Stock Exchange. It includes corporations in each of the ten industries listed in the Industry Classification Benchmark
5	^XAX	NYSE AMEX Composite	Covers a portion of securities traded on the NYSE Amex exchange.

Table 4 Hyper-parameters adopted in the implemented RL algorithm

Hyper-parameter	Value	Definition
nsteps	32	Number of steps of the vectorized environment per update
Total_timesteps	80M	total number of timesteps to train on
nactors	5	Number of actors learners
TC	5%	Rate of commission fee applied to each transaction
lr	7e–4	Learning rate for RMSProp
epsilon	1e–5	Stabilizes square root computation in denominator of RMSProp update
alpha	0.99	RMSProp decay parameter
gamma	0.99	Reward discounting parameter

4.2 Model setting

The DBN is composed of an input layer that receives 125 input features, three hidden layers with 100 , 80 and 60 neurons, respectively, and an output layer with 40 neurons. Each action-learner and critic-learner LSTM neural network controller consists of 40 inputs (received from the DBN), a hidden layer with 128 cells and an output layer with the number of action values based on the running model (one neuron for single stock or five neurons for five stocks trading). Table 4 lists hyper-parameters adopted in the implemented RL algorithm.

The adopted software framework is OpenAI [27]. It has the advantages of being open source besides containing a lot of reinforcement learning techniques including the synchronous parallel DRL algorithm. The implementation given in [28] has been modified to implement the following models:

1. Synchronous parallel DRL model with underlying DDPG technique (A2C) using LSTM controller function, fed by a DBN network, namely DBN-LSTM-DDPG-M model, to trade multi-stocks. The action domain in this model is a vector in $[-1..1]^n$.
2. Synchronous parallel DRL model with underlying DDPG technique (A2C) using LSTM controller function fed by a DBN network, namely DBN-LSTM-DDPG-S model, to trade one stock at a time. The action domain in this model is a vector in $[-1..1]$. The profit/loss of this model is calculated as the average profit/loss resulting from running this technique on the five stocks separately.
3. Synchronous parallel DRL model with underlying deep Q-learning technique (n-step Q-learning) using LSTM controller function fed by a DBN network, namely DBN-LSTM-DQN model, to trade one stock. The profit of this model is the average profit of trading five stocks separately.
4. Synchronous parallel DRL model with underlying DDPG technique (A2C) using feed-forward (FF) controller function with one hidden layer with 128 neurons, fed directly with the environment features, namely DDPG-FF-M model.
5. The classic Buy-and-Hold strategy, as a baseline trading strategy, where a stock is bought at the beginning of a test period and holds until the end of this period and then sold with the profit calculated based on the stock price at the beginning and the end of this period.

The training is performed in two stages: The first stage uses offline training where thirty-year data (from 01/01/1980 to 31/12/2009), resulting in 7824 samples, are used to train the proposed models. The testing (with training) is performed on the next ten years from 01/10/2010 to 01/10/2020. The

test period for an agent is one year, which is used also as a training period (250 samples) to allow the model to track the market. In order to test different financial market conditions, the average of the ten test periods from (01/10/2010 to 01/10/2020) is reported. In each test period year, the capital is set at the start of the year with no stocks in possession. The system is allowed to trade with the available capital money in addition to the money gained during the test period. In the case of the "Buy-and-Hold" experiment using the Q-learning technique, the trading money is divided equally between the stocks, and the profit is calculated as the sum of the profits over all the assets in the experiment.

4.3 Performance evaluation metrics

The performance metrics used to evaluate the proposed model are:

1. Accumulated Wealth Rate (AWR): it is defined as the sum of all profits and losses occurring during the test trading period T divided by the available cash at the start of the trading $Avc_{t=0}$:

$$\mathrm{AWR}_T = \frac{\sum_{t\in 1}^{T} PL_t}{Avc_{t=0}} \tag{24}$$

2. Average Profit Return (APR): it is defined as the average of the annual return rate, calculated as the average of the accumulated wealth rate gained during each year in the test period

$$\mathrm{APR}_T = \frac{\sum_{t\in 1}^{T}(AWR_t)}{T} \tag{25}$$

3. Average Sharp Ratio (ASR) : it is defined as the average of the annual sharp ratio during the test trading period T. The sharp ratio (SR) metric is defined in equation 23:

$$\mathrm{ASR}_T = \frac{\sum_{t\in 1}^{T}(SR_t)}{T} \tag{26}$$

4. Average Maximum Draw Down (AMDD): the Maximum Draw Down (MDD) is defined as the maximum loss from a peak to a trough in the value of the trading portfolio during time interval T, while AMDD is the average of the annual MDD during the test trading period

$$\begin{aligned} \mathrm{MDD}_t &= \frac{(\max v_t - \min v_t)_{t\in T}}{v_T} \\ \mathrm{AMDD}_T &= \frac{\sum_{t\in 1}^{T}(MDD_t)}{T} \end{aligned} \tag{27}$$

5. Average Calmar Ratio (ACR): the Calmar Ratio (CR) is defined as the average profit return during some interval divided by the MDD during this interval, while ACR is the average of the annual Calmar Ratio (CR) during the test trading period T

$$\begin{aligned} CR_t &= \frac{APR_t}{MDD_t} \\ ACR_T &= \frac{\sum_{t\in 1}^{T}(CR_t)}{T} \end{aligned} \tag{28}$$

6. Annualized Terms (ARR: Annualized Return Rate and ANSR: ANnualized Sharp Ratio): the annualized terms are defined as a time-weighted basis terms, an investor received over a given period (where period is greater than one year). To measure these terms the dataset1 and dataset2 are gathered in one dataset and the ten test period are gathered in one test period from 2010 to 2020 and the Accumulated Wealth Rate (AWR_T) defined in Eq. 24 and Sharp Ratio (SR_T) defined in equation 23 are calculated for the ten-year test period.

$$\begin{aligned} \mathrm{ARR}_T &= (1 + AWR_T)^{(1/T)} - 1. \\ \mathrm{ANSR}_t &= \frac{\sum_{t\in 1}^{T}(PL_t - FR_t)}{\delta(PL_T)} \\ \text{where} T &= 10 \end{aligned} \tag{29}$$

The first and second performance metrics are important for speculating investors who look for higher returns from the investments, whatever the risk they are exposed to during trading. The higher the values of these metrics (the profit gained) the better for the investors. The other metrics are used by risk-averse or risk-adjusted investors that adjust their investments according to the risk they are exposed to. The higher the value of the sharp ratio (SR) the better for the investors, while the lower the values of the MDD and CR metrics, the better for the investors.

4.3.1 Models comparison

The performance of our proposed models has been compared against other known benchmark models, presented in [37], and listed below. The open-source project code [35,36] was used and modified for this purpose.

1. Buy-and-Hold benchmark (B&H): buying stocks with equal capital and hold them until the end of the investment period and then sell them.
2. Best Stock benchmark (BS): put all the capital investment fund on buying the stock with the best expected performance, hold the stock until the end of the investment period and then sell it.
3. Constant Rebalanced Portfolios (CRP): construct the portfolio with the best expected stocks shares and then

reconstruct the stocks shares (Rebalance the portfolio) periodically to reach the best accumulative wealth until the end of the investment period. In our experiment, the portfolio is rebalanced at the end of each year.

Moreover, the open-source project code [38] was used and modified to models presented in [34,39]. The three different species of ensemble of identical independent evaluators (IIEs) framework presented in [34] are tested in this experiment: a convolutional neural network (CNN), a basic recurrent neural network (RNN), and a long short-term memory (LSTM), with the integrated convolutional neural network (ICNN) proposed in [39].

4.3.2 Time complexity

In order to estimate the time complexity introduced by combining DBN and LSTM NNs with the reinforcement learning technique, the time needed to train the model using thirty years of historical data, is compared with the time needed when the DBN is removed and the LSTM is replaced with a controller represented with a single hidden layer (with 128 neurons) feed-forward NN that is fed directly with the raw data.

4.3.3 Robustness check

As the proposed model "multi-stocks policy-based (DBN-LSTM-DDPG-M) model" is a continuous-action DRL model, its accuracy will be affected by increasing the number of traded stocks. Its financial profit is expected to increase as the chance of using the available cash in a suitable stock trading transaction increases. To measure the effect of increasing the traded stocks, the model has been run on different number of datasets with sizes 3, 5, 7, 10, 15 and 20 stocks. Table 5 contains the names of the stocks used in this experiment chosen from datasets 1 and 2.

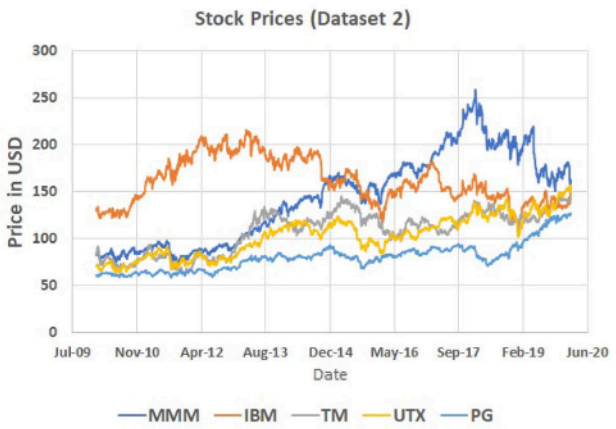
Table 5 Third trading stocks dataset

ID	Mnemonic	Name
1	AAPL	Apple Inc
2	AAL	American Airlines Group Inc
3	MSFT	Microsoft Corporation
4	JPM	JPMorgan Chase & Co
5	AMZN	Amazon.com INC
6	CSCO	Cisco Systems Inc
7	AXP	American Express Co
8	UNH	UnitedHealth Group Inc
9	CAT	Caterpillar Inc
10	PFE	Pfizer Inc

4.4 Results

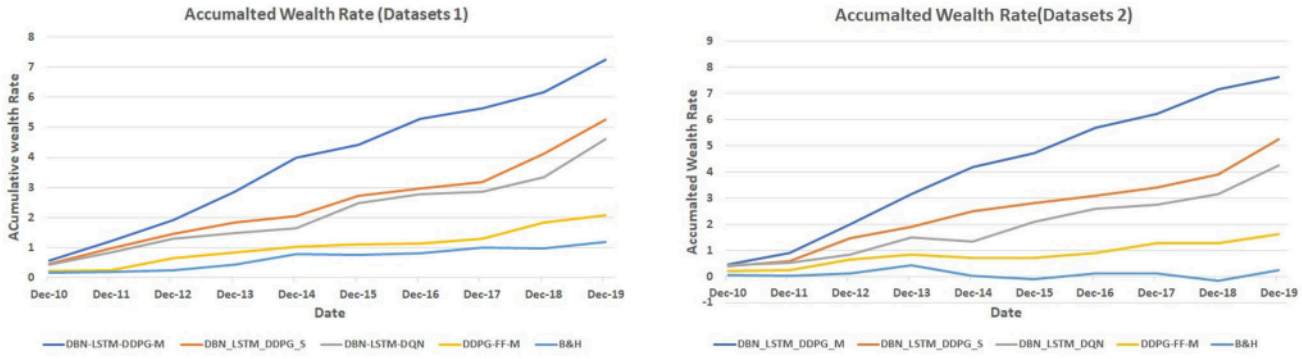
Figure 6 shows the stock prices of both datasets during the testing period, while Fig. 7 shows the accumulated wealth rate gained by different models on each dataset. Tables 6 and 7 illustrate the results for other performance metrics, each value in these tables is calculated as the average result (from 2010 to 2020) of the corresponding performance metric for each model. In general, the performance of the proposed model is much better than the other models. The proposed model achieves the highest accumulated wealth return with lower risk since the average sharp ratio is the highest ratio. Another indication for the lower risk are the values of the Maximum Draw Down (MDD), and Calmar Ratio (CR) which represent the lowest values among the other techniques. The reported results represent the best results found after trying different

(a) Stock Prices for dataset 1



(b) Stock Prices for dataset 2

Fig. 6 The stock prices for datasets during the testing period



(a) Accumulated wealth rate for dataset 1

(b) Accumulated wealth rate for dataset 2

Fig. 7 Accumulated wealth rate for different models during the test period

Table 6 Performance evaluation (dataset 1)

ID	Model	APR	ASR	AMDD	ACR
1	Multi-stocks policy-based (DBN-LSTM-DDPG-M) model	0.73	0.969	0.217	3.38
2	Single-stock policy-based (DBN-LSTM-DDPG-S) model	0.52	0.699	0.264	1.98
3	Single-stock value-based (DBN-LSTM-DQN) model	0.49	0.564	0.389	1.26
4	Multi-stocks policy-based (DDPG-FF-M) model	0.21	0.215	0.426	0.49
5	Buy-and-Hold model	0.12	0.194	0.551	0.22

Table 7 Performance evaluation (dataset 2)

ID	Model	APR	ASR	AMDD	ACR
1	Multi-stocks policy-based (DBN-LSTM-DDPG-M) model	0.76	0.821	0.375	2.03
2	Single-stock policy-based (DBN-LSTM-DDPG-S) model	0.57	0.648	0.484	1.19
3	Single-stock value-based (DBN-LSTM-DQN) model	0.43	0.532	0.543	0.79
4	Multi-stocks policy-based (DDPG-FF-M) model	0.16	0.325	0.622	0.24
5	Buy-and-Hold model	0.02	0.213	0.715	0.03

Table 8 Annualized terms performance evaluation for different models

ID	Model	ARR (%)	ANSR
1	DBN-LSTM-DDPG-M	23.7	1.96
2	DBN-LSTM-DDPG-S	20.1	1.14
3	DBN-LSTM-DQNl	18.4	0.098
4	DDPG-FF-M	11.9	0.08
5	EIIE-CNN	23.1	1.92
6	EIIE-RNN	21.7	1.69
7	EIIE-LSTM	19.6	1.11
8	ICNN	9.67	0.69
9	CRP	7.23	0.45
10	BS	8.2	0.63
11	B&H	6.98	0.25

sets of parameters for each technique. These results indicate that our trading model benefited from the feature extraction and reduction as well as time series predictions of the DBN and LSTMs, respectively.

Also, the following remarks can be concluded from the results:

1. The proposed "DBN" network and "LSTM" networks have improved the performance for both policy-based and value-based reinforcement learning techniques.
2. The policy-based RL with multi-stock trading model gives better performance metrics than both policy-based and value-based RL with single-stock trading model. This is attributed to the freedom given to the system to allocate the available money to any candidate stock. On the other hand, in case of one-stock trading, a lot of cash can be available, but not used, due to the lack of a good trading

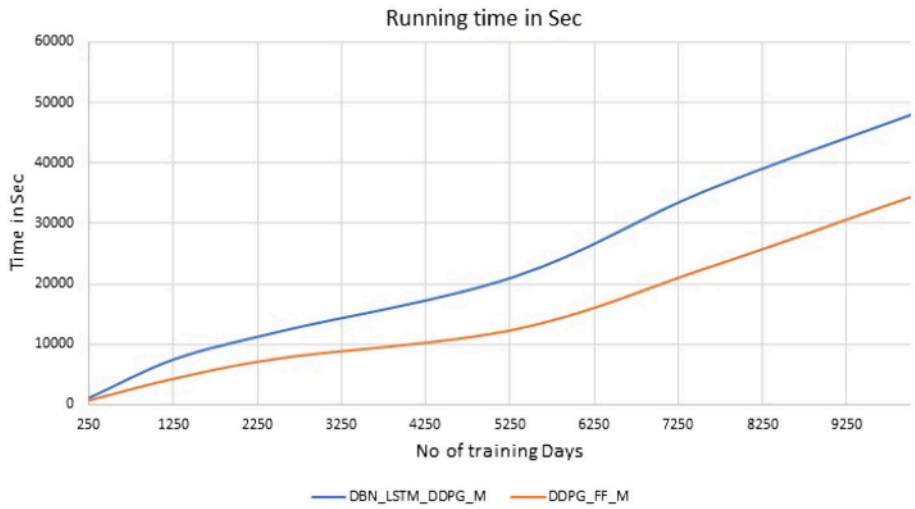


Fig. 8 The running time measured in seconds

Table 9 Robustness check performance evaluation

ID	Dataset size	APR	ASR	AMDD	ACR
1	3	0.681	0.628	0.321	2.28
2	5	0.755	0.895	0.296	2.79
3	7	0.754	0.828	0.288	2.82
4	10	0.712	0.725	0.314	2.18
5	20	0.512	0.525	0.528	0.89

opportunity to invest in (shown clearly in dataset 1, where the Ford company stock had little price volatility causing small trading profit).

3. The policy-based RL with one-dimensional model achieved better performance than the value-based RL model because of the availability of continuous actions to the policy-based model which allows the system to trade precise shares of stocks instead of the predetermined shares in the value-based model.
4. The original A2C model with the feed-forward neural network has lower performance but is still better than the Buy-and-Hold technique.

4.4.1 Models comparison results

The results of the annualized terms performance evaluation for different models are reported in Table 8. The results demonstrated that both model DBN-LSTM-DDPG-M and EIIE-CNN achieved very close result, while DBN-LSTM-DDPG-M outperformed all models' performances, and the EIIE-CNN achieved second place with slight difference. Then, the EIIE-RNN outperformed the DBN-LASTM-DDPG-S also with slight difference. The benchmarks strategies ("Best Stock," "Constant Rebalance Portfolio," "Buy and Hold") have given worst performance evaluation results among compared models.

4.4.2 Time complexity results

Figure 8 shows that the proposed multi-stock A2C algorithm (with DBN and LSTM) model consumes larger running time compared to the normal (A2C algorithm with feed-forward neural network) model. As expected, the running time increases as the training dataset size increases, but the parallel design of the framework makes the consumed time acceptable. The model's training is firstly done offline, for around thirteen hours (for thirty years historical data). Next, a daily online training, which takes only few minutes, is needed.

4.4.3 Robustness check results

The results of robustness check experiment are reported in Table 9. The performance evaluation metrics are examined with different randomly chosen stocks from the available twenty stocks. The results indicate that choosing the number of stocks between 5 and 10 gives better performance. Therefore, using five stocks seem a reasonable choice.

5 Conclusions and future work

This research proposed a multi-stock trading model, which has the ability to abstract dynamic financial information into useful knowledge, using a parallel synchronized reinforcement learning algorithm implemented using deep learning techniques. The trading strategy is dynamic and automatically generated by the model. It can trade multiple stocks simultaneously and in small proportions whenever needed (continue action domain) using the available trading money

efficiently. After an initial offline training phase, on large historical data, the model needs to run for only few minutes, on a multi-core CPU or GPU machine, to adapt to the daily changes in the market. This makes the model applicable for real-life usage. Our future work will focus on enriching the model with a crisis detector to allow it to take more risk-aversion decisions during crisis periods.

References

1. Hasbrouck., J.: 22 Modeling market microstructure time series, In: Handbook of Statistics, Vol. 14, pp. 647-692, ELSEVIER(1996). https://doi.org/10.1016/S0169-7161(96)14024-4
2. Pate, J., Shah, S., Thakkar, P.: Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. Exp. Syst. Appl. **42**(1), 259–268 (2015). https://doi.org/10.1016/j.eswa.2014.07.040. Elsevier
3. Cavalcantea, R.C., Brasileirob, R.C., Souza, V.L., Nobrega, J.P., Oliveirab, A.L.I.: Computational intelligence and financial markets: A survey and future directions. Exp. Syst. Appl. **55**, 194–211 (2016). https://doi.org/10.1016/j.eswa.2016.02.006
4. Gupta, P., Mehlawat, M.K., Mittal, G.: Asset portfolio optimization using support vector machines and real-coded genetic algorithm. J. Global Optim. **53**(2), 297–315 (2012)
5. Yang, B., Gong, Z.-J., Yang, W.: Stock market index prediction using deep neural network ensemble, In: 36th Chinese Control Conference (CCC), pp. 26-28, Dalian, China (2017). https://doi.org/10.23919/ChiCC.2017.8027964
6. Zhang, J., Shicheng, C., Yan, X., Qianmu, L., Tao, L.: A novel data-driven stock price trend prediction system. Exp. Syst. Appl. **97**, 60–69 (2018). https://doi.org/10.1016/j.eswa.2017.12.026
7. Chonga, E., Han, C., Parka, F.C.: Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. Exp. Syst. Appl. **83**, 187–205 (2017)
8. Lee, J., Kang, J.: Effectively training neural networks for stock index prediction: Predicting the S&P 500 index without using its index data. PLoS ONE **V15**(4), e0230635 (2020). https://doi.org/10.1371/journal.pone.0230635
9. Sezer, O., Ozbayoglu, M.: Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach. Appl. Soft Comput. **70** (2018) https://doi.org/10.1016/j.asoc.2018.04.024
10. Jiang, W.: Applications of deep learning in stock market prediction: recent progress, arXiv:2003.01859 (2020), Preprint submitted to Elsevier Journal
11. Murat, A.M., Omer, M.U., Sezer, B.S.: Deep learning for financial applications : A survey. Appl. Soft Comput. **93**, 106384 (2020). https://doi.org/10.1016/j.asoc.2020.106384
12. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Driessche, G., Graepel, T., Hassabis, D.: Mastering the game of go without human knowledge. Int. J. Sci. Nat. **550**, 354–359 (2017)
13. https://colah.github.io/posts/2015-08-Understanding-LSTMs/ Accessed (August 2020)
14. Watkins, C.J., Dayan, P.: Q-learning, Machine Learning, vol. 8, pp. 279–292. Springer, Berlin (1992)
15. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)
16. Schmid, Huber J.: Deep learning in neural networks: An overview. Neural Netw. V **61**, 85–117 (2015)
17. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Comput. **V18**(7), 1527–1554 (2006)
18. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. Adv. Neural Inf. Process. Syst. **V**(19), 153–160 (2006)
19. Hasselt, H.V.: Double Q-learning. Adv. Neural Inf. Process. Syst. **23**, 2613–2621 (2010)
20. Wang, Z., Freitas, N., de., Lanctot, M.: Dueling network architectures for deep reinforcement learning, In the International Conference on Machine Learning (ICML), (2015). arXiv preprint arXiv:1511.06581
21. Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D.: Rainbow: Combining Improvements in Deep Reinforcement Learning, Thirty-Second AAAI Conference on Artificial Intelligence (2017). arXiv preprint arXiv:1710.02298
22. Sutton, R.S., McAllester, D.A., Singh, S.P., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation, In: Advances in Neural Information Processing Systems, Vol. 12, pp. 1057–1063. (NIPS 1999) MIT Press, Cambridge, MA (2000)
23. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning, In: International Conference Learning Representations (2016). arXiv preprint arXiv:1509.02971
24. Schulman, J., Levine, S., Abbeel, P., Jordan, M.I., Moritz, P.: Trust Region Policy Optimization, In: 32nd International Conference on Machine Learning, Vol. 37, pp. 1889–1897, PMLR. http://proceedings.mlr.press/v37/schulman15.html (2015)
25. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms, Computing Research Repository (CoRR), 1707.06347 (2017). arXiv preprint arXiv:1707.06347
26. Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., Freitas, N.: Sample efficient actor-critic with experience replay,ICLR (2016). arXiv preprint arXiv:1611.01224
27. OpenAI, https://openai.com/, Accessed 1.7 April 2020
28. OpenAI Baselines: ACKTR & A2C, https://openai.com/blog/baselines-acktr-a2c/, Accessed 17 April 2020
29. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmil-ler, M.: Playing atari with deep reinforcement learning, In NIPS Deep Learning Work-shop (2013)
30. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning, In: 33rd International Conference on Machine Learning, Vol. 48, pp. 1928-1937, PMLR (2016)
31. Moody, J., Saffell, M.: Learning to trade via direct reinforcement. IEEE Trans. Neural Netw. **12**(4), 875–889 (2001). https://doi.org/10.1109/72.935097
32. Deng, Y., Bao, F., Youyong, K., Zhiquan, R., Qionghai, D.: Deep direct reinforcement learning for financial signal representation and trading. IEEE Trans. Neural Netw. Learn. Syst. **28**(3), 653–664 (2017)
33. Almahdi, S., Yang, S.Y.: An adaptive portfolio trading system: A risk-return portfolio optimization using recurrent reinforcement learning with expected maximum drawdown. Exp. Syst. Appl. **V**(87), 267–279 (2017). https://doi.org/10.1016/j.eswa.2017.06.023
34. Jiang, Z., Xu, D., Liang, J.: A deep reinforcement learning framework for the financial portfolio management problem, arXiv:1706.10059 (2017)
35. https://github.com/OLPS/OLPS, last accessed October 2020

36. Li, B., Sahoo, D., S. CH. Hoi.: Olps: A toolbox for online portfolio selection., J. Mach. Learn. Res. (JMLR), (2015)
37. Li, B., Hoi, S.C.H.: Online portfolio selection: A survey. ACM Comput. Surv. (CSUR) **V46**(3), 35 (2014)
38. https://github.com/ZhengyaoJiang/PGPortfolio , last accessed October 2020
39. Jiang, Z., Liang, J.: Cryptocurrency portfolio management with deep reinforcement learning., Intelligent Systems Conference., SAI Conferences,2017. Preprint: arXiv:1612.01277
40. Liang, Z., Chen, H., Zhu, J., Jiang, K., Li, Y.: Adversarial Deep Reinforcement Learning in Portfolio Management, arXiv:1808.09940 (2018)
41. Hegde, S., Kumar, V., Singh, A.: Risk aware portfolio construction using deep deterministic policy gradients, IEEE Symposium Series on Computational Intelligence (2018)
42. Wang, J., Zhang, Y., Tang, K., Wu, J., Xiong, Z.: AlphaStock: A Buying Winners-and-Selling-Losers Investment Strategy using Interpretable Deep Reinforcement Attention Networks, 25th ACM SIGKDD, pp.1900-1908 (2019)
43. Li, Y., Zheng, W., Zheng, Z.: Deep Robust Reinforcement Learning for Practical Algorithmic Trading, IEEE Access, pp.108014–108022 (2019)
44. Soleymani, F., Elodie, P.: Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder - DeepBreath". Exp. Syst. Appl. **156**, 113456 (2020)
45. Leem, J., Kim, H.Y.: Action specialized expert ensemble trading system with extended discrete action space using deep reinforcement learning. PLoS ONE **15**(7), e0236178 (2020). https://doi.org/10.1371/journal.pone.0236178
46. Mosavi, A., Ghamisi, P., Faghan, Y., Duan, P.: Shamshirband. Comprehensive Review of Deep Reinforcement Learning Methods and Applications in Economics. (2020). https://doi.org/10.20944/preprints202003.0309.v1
47. Charpentier, A., Elie, R., Remlinger, C.: Reinforcement Learning in Economics and Finance (2020) arXiv:2003.10014