Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Deep reinforcement learning based trading agents: Risk curiosity driven learning for financial rules-based policy

Badr Hirchoua [a,*], Brahim Ouhbi [a], Bouchra Frikh [b]

[a] LM2I Laboratory, Industrial Engineering and Productivity Department, National Higher School of Arts and Crafts (ENSAM), Moulay Ismail University (UMI), Meknes, Morocco
[b] LTTI Laboratory, Computer Science Department, Higher School of Technology (EST), Sidi Mohamed Ben Abdellah University (USMBA), Fez, Morocco

A R T I C L E I N F O

Keywords:
Deep reinforcement learning
Partially observable Markov decision process
Knowledge uncertainty
Financial data
Trading system
Financial engineering

A B S T R A C T

Financial markets are complex dynamic systems influenced by a high number of active agents, which produce a behavior with high randomness and noise. Trading strategies are well depicted as an online decision-making problem involving imperfect information and aiming to maximize the return while restraining the risk. However, it is challenging to obtain an optimal strategy in the complex and dynamic stock market. Therefore, recent developments in similar environments have pushed researchers towards exciting new horizons.

In this paper, a novel rule-based policy approach is proposed to train a deep reinforcement learning agent for automated financial trading. Precisely, a continuous virtual environment has been created, with different versions of agents trading against one another. During this multiplex process, the agents which are trained on 504 risky datasets, use the fundamental concepts of proximal policy optimization to improve their own decision making by adjusting their action choice against the uncertainty of states. Risk curiosity-driven learning acts as an intrinsic reward function and is heavily laden with signals to find salient relationships between actions and market behaviors. The trained agent based on curiosity-driven risk has steadily and progressively improved actions quality. The self-learned rules driven by the agent curiosity push the policy towards actions that yield a high performance over the environment. Experiments on 8 real-world stocks are given to verify the appropriateness and efficiency of the self-learned rules. The proposed system has achieved promising performances, made better trades using fewer transactions, and outperformed the state-of-the-art baselines.

1. Introduction

Stock markets are designed to elicit information from a complex and continuous real-world environment. Automated trading, also known as agent trading, is a developed method to submit many trading orders to an exchange (Huang, Huan, Xu, Zheng, & Zou, 2019). Trading in most financial markets happens between buyers and sellers in a continuous double auction with an open order book on an exchange. The trader attempts to buy on a low price and sell on a high one. To this aim, the targeted rules must meet some standards such as deciding “when to buy”, “when to sell”, “what the target price is”, and “how long to target”. Additionally, as the market indices keep randomly changing, the target prices are continuously adjusting. The market inputs such as volume in addition to stock price are real-time changing variables. This leads to the change of the targeted price. Basically, if the agent is able to predict that the market will go up, then it has the ability to buy immediately and sell once it is gone up. In high-frequency trading, where the stock market presents a dynamic unstable evolution, the chance to accurately predict the price converges to zero. Since the agent is dealing with a decision-making problem under uncertainty, it becomes difficult to estimate a probability distribution over the price and return. Hence, the agent needs more than just a price prediction model (Dash & Dash, 2016). Instead, a rule-based policy is required. The policy takes the price prediction as input and the decision as output.

The increase in the available data and the ease of access to the market information have expanded the usefulness of trading algorithms (Chaboud, Chiquoine, Hjalmarsson, & Vega, 2014). Diverse approaches attempt to handle the automated trading problem. Most successful methods are based on optimization (Goldkamp & Dehghanimohammadabadi, 2019) and Reinforcement Learning (RL) (Lei, Zhang, Li, Yang, & Shen, 2020), or on a combination of both of them (Deng, Kong, Bao, & Dai, 2015). The optimization methods require a more profound

* Corresponding author.
E-mail addresses: hirchoua.badr@gmail.com (B. Hirchoua), ouhbib@yahoo.co.uk (B. Ouhbi), bfrikh@yahoo.com (B. Frikh).

https://doi.org/10.1016/j.eswa.2020.114553
Received 23 September 2019; Received in revised form 5 August 2020; Accepted 30 December 2020
Available online 6 January 2021

knowledge of the problem and impose more assumptions. On the other hand, the RL methods learn accurate and precise thresholds and parameters by only interacting with the environment. Moreover, the RL inherents adaptive nature. If the conditions of an environment change, RL can usually adapt and anticipate the change. RL shows promising results on financial and economic problems (Hull, 2014; Lei et al., 2020), where there is a possibility to apply it (RL) to improve trading strategies (Eilers, Dunis, von Mettenheim, & Breitner, 2014), or to build trading systems (James, Alrajeh, & Dickens, 2015). In financial trading, the deep reinforcement learning (DRL) agent attempts to maximize the returns and minimize the risk simultaneously, by automating the human expert thinking that anticipates the movements of the financial market before making a decision.

The DRL (Spooner, Fearnley, Savani, & Koukorinis, 2018; Huang, 2018; Ganesh & Rakheja, 2018) process, presented in Fig. 1, is formulated as the following: at each time step, the agent receives the current state as an input and suggests an action. Depending on the chosen action, the agent receives a reward as well as the next state. The agent decides which action to take based on the learned policy. The goal of the policy is to maximize the cumulative reward over a time horizon. This is the profit from each transaction. In the context of the current problem, the agent is the trader which decides when and how to make trades on the market. The environment details are constantly changing and the agent has little control over these changes. Meanwhile, the environment states are partially, not fully, observable observable. Thus, dynamic decision-making is more challenging due to the lack of states details. Hence, the agent is obliged to explore unknown trajectories and simultaneously make correct real-time decisions.

Various financial market trading works are deep learning-based methods (Eilers et al., 2014; James et al., 2015; Huang, 2018). These methods use the historical market data to predict the price movements or trends. The trading algorithm is not bound to predict the price but directly outputs the actions to take in a given state. On the one hand, the trading rules including what action to take requires human-designed models or handcrafted features; and this is impossible due to the environment nature. On the other hand, the predicted price differs significantly from the real one, while the ultimate goal of the agent is to make a higher profit instead of a higher price-prediction accuracy. Comparing to human experts, RL learns the appropriate and optimal policies (Mnih et al., 2015). The quality of financial markets data, or lack thereof, is not sufficient to produce a successful trading strategy. This makes the problem of finding hidden signals more complicated. Despite the sophistication of DRL algorithms, a basic exploration policy is usually used. In financial trading, using basic exploration is infeasible. This infeasibility generates huge losses and worse performances.

In addressing the aforementioned problems, this paper introduces a novel rule-based policy for simultaneous real-time decision making and risk curiosity-driven learning. Moreover, different versions of the agent are trained on a continuous virtual environment, involving risky datasets. The trained agent, concerned by designing strategies using DRL, achieves a global control objective through this environment. During this struggling process, the agent handles the above shortcomings by using the fundamental concepts of the proximal policy optimization (PPO) (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017) to improve its own decision making and adjusting its actions behavior against the observed states. The PPO provides the agent with a model synthesized from past experiences. It also forces policy updates to be conservative if they move far from the accurate policy (choosing the appropriate actions). Besides, the PPO takes the solution to a higher level of accuracy by upgrading the off-policy to the on-policy, leading the agent to act in real time scale. Notwithstanding the fact that the model needs to be learned first, the PPO enables improved generalization across trajectories making it easy to exploit the additional learning signals. Thus, it leads to greater data efficiency. Another appeal of the trained agent is its ability to increase the number of simulations leading to an accurate exploration and experience accumulation. This, in turn, causes an extension of the model's performances.

In addition to the policy mechanism, the trained agent uses a flexible and scalable model to reason about the market risk. The agent seeks positive incomes while avoiding the adverse consequences of long trial-and-error procedure in the real environment, including making irreversible or poor decisions. Furthermore, the risk curiosity is integrated into the proposed algorithm yielded to a faster and controlled learning, in addition to improved generalization over simulations. The risk curiosity drives the agent learning in several axes. First, it provides an additional reward signal, which handles the reward sparsity. Second, curiosity-driven exploration attends to choose the convenient action to maximize the agent's reward. RL models use the regret mechanism to solve this problem, which is the expected loss due to not choosing the best action. Moreover, the agent curiosity adopts another form of regret function by learning a risk function-based that maps the partially observed state to a more likely state form. Finally, the curiosity anticipates the market behavior, making the agent more curious about whether the prediction on the up-or-down direction is right.

The remainder of this paper is organized into sections. The problem is reformulated in Section 2. In Section 3, the related works of this research are reviewed. In Section 4, the necessary backgrounds and the basic concepts and properties are introduced. First, the partially observable Markov decision process is introduced. Second, the policy learning methods are illustrated including deep Q-Learning, trust region policy optimization, and proximal policy optimization. Finally, the risk curiosity-driven learning is presented. In Section 5, the contribution concepts and properties of different methods parts are discussed. Section 6 includes the results of the novel approach which is validated through comparative analyses. Moreover, discussions are summarized and included in the same section. Section 7 concludes this work and presents the proposals for future ones.

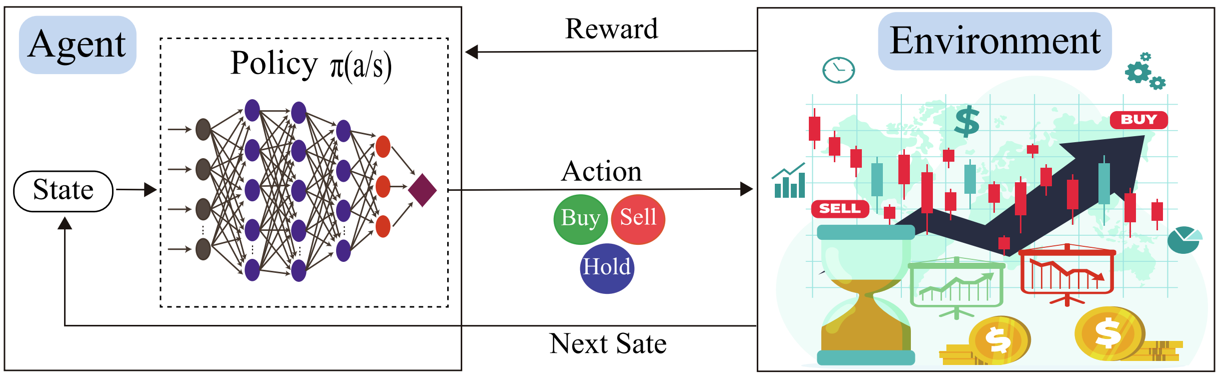

Fig. 1. The DRL framework.

2. Problem statement

The main purpose of this section is to identify and explain the problems addressed in this paper. This section makes clear the purpose of handling and improving financial trading goals.

The model attempts to learn the optimal, accurate, and profitable action at any time. It consists of states S, actions A, and rewards R. The interaction between the agent and the environment is presented in Fig. 1. The environment is the exchange containing different agents, both humans and algorithms. For any given stock, the agent can perform three different actions in the environment $A = \{Buy, \quad Sell, \quad Hold\}$. The number of actions, at any given time step, is 3^n, where n is the number of stock in the portfolio. Each state S combines the historical prices of all exchange events received up until that time, the account balance, and the number of owned stocks. The reward R is purely the profit from each transaction. In practice, the problem is how to learn a policy that maximizes the reward.

In the context of the current problem, there are various best strategies. As such, the agent training process needs to continuously explore the environment for more synthesized experiences. Unlike many real-world problems where agents see everything, crucial information is hidden from the trading agent. The agent is forced to take the best action with imperfect information. Moreover, due to the nature of the problem, the causes-and-effects are not instantaneous, meaning that actions taken early may not pay off for a long time. Thus, the agent resonates about long term planning. The need to balance both long-term and short-term goals and harmonize to unforeseen circumstances poses a considerable challenge for this system that has often tended to be hard and risky (Sastry & Thompson (2019)). However, mastering the financial trading problems demands breakthroughs at several levels including:

- **Data complexity:** the financial markets data is non-stationary and has nonlinear patterns as well as low signals. This leads to the highly non-stationary time series.
- **Partially-observed state:** the states' details are unobservable, where the agent observes a derivation of the actual state. Strong actions require making inferences based on incomplete data.
- **Long time horizons:** some actions performed by the agent have minor impact on the market, while other ones can affect the trading algorithm strategically. Each agent has three actions yielding to different states context. A basic learned strategy may end before few transactions, while other profitable strategies continue to be profitable for a long time horizon.
- **High-dimensional continuous observation space:** Financial markets are complex dynamic systems containing dozens of active agents. Therefore, each agent's action influences the observation sequences continuously and differently according to the agent parameters.
- **Sparse reward:** the reward sparsity issue is represented by the long-time delay between actions and their associated rewards. Value functions offer an effective solution to this problem by allowing the system to estimate the goodness of an action before the delayed reward arrives. The proposed system considers algorithms that optimize a parameterized policy and uses value functions to help estimate the way the policy is better improved.
- **Risk management:** the financial market suffers from randomly changing prices. The action taken by the agent must be controlled by a risk management mechanism. In other words, the agent must handle the risk of taking a worse action.

To mitigate data uncertainty and noise, handcraft features such as stochastic technical indicators (Neely, Rapach, Tu, & Zhou (2014)) can be used to condense the market conditions. However, the downside of technical analysis is its poor generalization ability. The trading positions (long or short) are heavily changing, and this might cause huge losses. Accordingly, the past good actions and the corresponding positions are, meanwhile, required to be explicitly modeled and learned in the policy learning process. The data distribution over observations and rewards is consistently changing as the agent learns.

Due to these challenges, deep reinforcement learning-based trading agent has emerged as a grand challenge for researchers. To help intelligent agents further explore these problems, this work integrates the PPO and risk curiosity-driven learning in the continuous virtual environment to produce a strong rule-based policy, revealing promising performances over real-world data sets.

3. Related work

This section includes and briefly describes some of the research works that have addressed the same problems handled in this work. Specifically, the general idea and some shortcomings, if any, for these methods are presented. Also, these methods involve many assumptions. These assumptions, in turn, consist of a number of hypotheses that are briefly introduced in this paper. First, DRL based financial trading works are discussed. Second, the relevance of the described works, lying in risk curiosity-driven learning problems, are compared to financial trading-based sentiment analysis.

3.1. Deep reinforcement learning based trading agents

Financial markets are complex dynamic systems with a high number of active agents (investors, traders, and hedgers) influenced by one another and by external information (news, economic data, and events). This produces a behavior with high randomness and noise which is very difficult to predict (Machado, Neves, & Horta, 2015). Thus, various tools and methods have focused on the financial industry (Eilers et al., 2014; Lei et al., 2020). Precisely, RL has shown promising results on the financial and economic problems (Hull, 2014; Li, Zheng, & Zheng, 2019).

Eilers et al. (2014) present a decision support algorithm which uses the powerful ideas of RL in order to improve the economic benefits of the basic seasonality strategy. However, these approaches require large amounts of training data, but the resulting policies are not immediately generalized to novel tasks in the same environment. They also reduce the behavioral flexibility constitutive of general intelligence. James et al. (2015) introduce a RL trading algorithm based on least-squares temporal difference (LSTD) to estimate the state value function.

Deng et al. (2015) introduce a typical DRL framework for financial signal processing and online trading. They frame the learning process as a bilevel optimization (BO) and solve it by the online gradient descend (OGD) method with fast convergence. After identifying the technical-indicator-free trading system, the framework extends the fuzzy learning, which reduces the uncertainty in the original time series. Existing DRL algorithms such as deep Q-learning (DQL) method (Mnih et al., 2015; Silver et al., 2016) make remarkable achievements in the perfect information environment with discrete actions. Hence, these algorithms are not scalable for continuous actions space, especially in high-frequency trading (Carapuço, Neves, & Horta, 2018).

Moody and Saffell (2001) take advantage of the policy search to learn and practice trading. Meanwhile, Deng, Bao, Kong, Ren, and Dai (2016) extend this method by using deep neural network policy. Precisely, they use Fuzzy Deep Direct Reinforcement Learning (FDDR), Deep Multilayer Perceptron (DMLP), and RL methods to predict 300 stocks from SZSE index data and commodity prices.

Di Persio and Honchar (2017) compare three Recurrent Neural Network models to evaluate which variant of RNN performs better. Notably, they evaluate a basic RNN, a Long Short Term Memory (LSTM), and a Gated Recurrent Unit (GRU). The LSTM outperforms other variants with a 72% accuracy on a five-day horizon. Jiang, Xu, and Liang (2017) propose an ensemble of identical independent evaluators for portfolio management. This framework trains a neural network in an online stochastic batch learning scheme, which is appropriate for both pre-trade training and online training. Si, Li, Ding, and Rao (2017) propose a multi-objective intraday financial signal representation and trading method. This method utilizes a Multi-objective Deep Reinforcement Learning (MODRL) methodology followed by a reinforcement learning framework (with ad hoc LSTMs) to extract market features and make continuous trading decisions. Lu (2017) implements and tests an agent inspired trading using deep (recurrent) reinforcement learning, LSTM, and Natural Evolution Strategies (NES) on the trading of GBP/USD.

Serrano (2018) proposes an asset banker RL algorithm for algorithmic trading. It uses a random neural network, Genetic Algorithm (GP), and RL to generate the trading signals. Chen, Chen, and Huang (2018) train an agent-based RL algorithm with a 1-dimensional CNN on the Taiwan stock index futures (TAIFEX) dataset.

Jeong and Kim (2019) provide an advanced trading application in the context of expert systems and derive an asset trading rule to determine actions for assets and the number of shares for the actions taken. Specifically, they combine a DQL with a novel deep neural network structure consisting of two branches for learning action values and the number of shares to take, respectively. Azhikodan, Bhat, and Jadhav (2019) demonstrate that DRL is capable of learning the tricks of stock trading by predicting the trend in stock value. Their system implements deep deterministic policy gradient-based neural network model along with a deep recurrent convolutional neural network (RCNN) for sentiment analysis model. Li et al. (2019) implement a trading agent based on DRL to autonomously perform trading decisions through a modified deep Q-network (DQN) and actor-critic (A3C) approach. They utilize the stacked denoising autoencoders (SDAEs) and LSTM to design a robust mechanism that makes the trained agent more efficient.

Lei et al. (2020) propose a time-driven feature-aware combined deep reinforcement learning model (TFJ-DRL) to improve the action decision making and financial signal representation learning in algorithmic trading.

The critic-actor deterministic policy gradient outputs continuous actions, using two neural networks. The first neural network trains a Q-function estimator as its reward function, and the second (network) as the action function (Silver et al., 2014; Lillicrap et al., 2015). However, training two neural networks (the critic and the actor) is found to be difficult and unstable.

Du, Zhai, and Lv (2016) has used a policy gradient Q-learning based and found that the policy search algorithm enabled a simpler problem representation than the value function-based search algorithm. Specifically, DQL techniques can be applied in RL trading systems to approximate the action-value function (Jin & El-Saawy, 2016). Despite the fact that the value-function-based methods perform well for a number of problems, Moody and Saffell (2001) indicate that it is not suitable for the trading problem. This is due to the trading environment which is too complex to be approximated in a discrete space. In other words, the value-function-based methods are appropriate for the offline problems (Tesauro, 1994), but not for dynamic real-time trading (Moody & Saffell, 2001; Deng et al., 2015). Precisely, the definition of the value function in the Q-learning always implies a reply buffer for recording the future discounted returns, whereas the nature of trading requires to count the profits in real-time scale. A hallmark of an intelligent agent is its capacity to expeditiously adapt to new situations (Legg & Hutter, 2007). Progress has been made in developing capable agents for directly map raw observations to values or actions (Mnih et al., 2016; Schulman, Levine, Abbeel, Jordan, & Moritz, 2015).

In addition, recent successes are mastered by model-free methods (Lillicrap et al., 2015). However, the standard planning methods suffer from model errors resulting from function approximation (Talvitie, 2014). These errors compound during simulations causing poor agent performance. There are currently no clear planning solutions or methods that are robust against model imperfections which are inexorable in such environments. Last, DRL provides intelligent solutions to manage the risk issue (Hull, 2014), where the training process focuses on choosing profitable action, as well as reducing the risk due to not choosing the best action.

Table 1 provides a summary of recent works related to our research.

3.2. Risk curiosity-driven learning

Reinforcement learning-based trading agents suffer from reward sparsity, which is the time difference between an action and its associated reward. Thus, it is hard to attribute rewards to specific actions. Furthermore, due to the environment nature, the state details are unavailable. Whereas using more granular data, the agent runs into another problem. The learning process must refer to the historical details without adding extra complexities in different learning parts.

Curiosity-driven learning has many forms and techniques. The public mood has been found to be a key element for stock market prediction, so it can be discussed as a curiosity mechanism (Wu, Yu, & Chang, 2014). Moreover, combining language models with sentiment analysis probably identifies the main events in news articles, which is also combined with historical stock index and currency exchange values for prediction (Jin et al., 2013). Xing, Cambria, Malandri, and Vercellis (2019) formalize the public mood into market views since it can be integrated into the modern portfolio theory. Their framework starts by training two neural networks models: the first one generates the market views, and the second one creates the model performance benchmark. Even if the experimental results increase the profitability (5% to 10% annually), but in the high-frequency trading, the time response taken by the two networks is critical. Nevertheless, the stock market benchmarks are unsafe, because they are changing dynamically. Besides, the general process of sentiment analysis ignores the market behavior (states context), so the agent is not able to execute trade orders based on this theoretical index. In addressing the aforementioned problems and in the favor to make the agent more ambitious to the targeted goal, a curiosity-based risk and return is used in this work.

In order to handle the market risk and increase awareness of small probability events, Chow, Ghavamzadeh, Janson, and Pavone (2017) present an efficient RL algorithm for risk constrained Markov Decision Process (MDP). Risk problems are taken as percentile risk-constrained MDP. Specifically, the percentile risk constrained MDP is approached by computing the gradient of the Lagrangian function. Therefore, the policy gradient and actor-critic algorithms are presented in three steps. After estimating the gradient, the policy in the descent direction is updated. Then, the Lagrange multiplier is updated in the ascent direction. Almahdi and Yang (2017) suggest risk-return portfolio optimization based on recurrent reinforcement learning method in order to optimize financial investments. Buehler, Gonon, Teichmann, and Wood (2019) present a DRL framework for hedging a portfolio of derivatives with a strong point of handling the market frictions including risk limits. Vella and Lon (2016) use higher order fuzzy systems (i.e. an adaptive neuro-fuzzy inference system). They also introduce the Type-2 Fuzzy sets to handle increased uncertainty, mostly induced by the market microstructure noise in the high-frequency trading sphere. Their main objective is to improve the risk-adjusted performance with minimal increase in the design and computational complexity.

Despite the studies mentioned above, deep reinforcement learning-based trading agent remains an open problem. Therefore, maximizing the profit in an uncertain environment is the core of the proposed approach. Due to the described limitations of existing methods, and motivated by the combination of PPO and risk curiosity used inside the novel environment, this work presents a novel trading strategy approach. The idea is to focus more on improving the action quality, especially the rules-based policy expressing the action to take in a given state instead of concentrating on the price prediction model.

The necessary backgrounds are introduced in the next section. First, the partially observable Markov decision process is introduced. Second, the policy learning methods are illustrated, including deep Q-learning, trust region policy optimization, and proximal policy optimization.

Table 1
Summary of Related Work on deep learning, RL, and DRL in financial trading.

Paper	Dataset	Period	Technique	Prediction Type	Metrics
Eilers et al. (2014)	DAX, S&P 500	2000–2012	ANN	Seasonality, monthly	Profit
James et al. (2015)	EUR/USD	2014	LSTD	Minute	Profit
Deng et al. (2015)	China Stock Index Future (IF)	2013–2014	BO + OGD	Daily	Profit and Loss (P&L), Sharpe Ratio (SR), Profitable days
Deng et al. (2016)	SZSE	2014–2015	FDDR, DMLP + RL	Minute, Daily	P&L, SR, profitable rate, Trading times, return
Jiang et al. (2017)	80 tradable cryptocurrency	-	CNN + RNN + LSTM	Minute	Portfolio value, SR, Maximum Drawdown (MDD)
Di Persio and Honchar (2017)	Google	-	RNN, LSTM, GRU	Daily, Weekly	Log loss, accuracy
Si et al. (2017)	IF-IH-IC	2016–2017	MODRL + LSTM	Minute	P&L, SR
Lu (2017)	GBP/USD	2017	RL + LSTM + NES	-	SR, Downside, Deviation ratio, Total profit
Serrano (2018)	Derivative market/ Bond	-	RL, DMLP, GA	-	Learning and genetic algorithm error
Chen et al. (2018)	Taiwan TAIFEX	2017	RL agent + CNN	-	Accuracy
Li et al. (2019)	APPL, IBM, PG, S&P 500, ES, IF	2008–2018	DQN + A3C + SDAEs + LSTM	Minute, Daily	Return, SR
Azhikodan et al. (2019)	NASDAQ-GE, NASDAQ-GOOGL	-	NN + RCNN	-	Accuracy, Stock held
Jeong and Kim (2019)	S&P500, KOSPI, EuroStoxx50, HSI	1987–2017	DQL + DNN	Daily	Total profit, Correlation
Lei et al. (2020)	S&P500	1999–2018	TFJ + DRL	Daily	Profit, Return, SR, Transaction times.
Park et al. (2020)	US-ETF, KOR-IDX	2010–2017	DQL	-	Cumulative return, SR, Sterling ratio, Average turnover
The proposed approach	S&P500, FB, Baba, GooG, SFix, Gold, AApl, BTC	1960–2019	DRL - PPO	-	Cumulative return, Profit, # transactions, Risk

Finally, the risk curiosity-driven learning is presented.

4. Preliminaries

In this section, the basic concepts and properties of the partially observable Markov decision process are first introduced. Second, the policy learning methods are briefly reviewed including deep Q-Learning, trust region policy optimization, and proximal policy optimization. Finally, the risk curiosity-driven learning is presented.

4.1. Partially observable Markov decision process

A Partially Observable Markov Decision Process (POMDP) (Kaelbling, Littman, & Cassandra, 1998) is a generalization of a MDP. The POMDP framework, presented in Definition 1, models diversity of dynamic real-world decision processes to produce a policy that fulfills a given task. A POMDP models an agent decision process which cannot directly observe the underlying state. In order to maintain the probability distribution over states, the general framework of MDP's involving incomplete information and discrete state space (Åström, 1965) must resonate upon the set of observations, observation probabilities, and the underlying MDP. The targeted aim in the POMDP model is to maximize the obtained reward while reasoning about the state uncertainty. The sequence of optimal actions yields the optimal policy.

Definition 1. A POMDP models the relationship between an agent and its environment. Formally, a POMDP is a 7-tuple $< S, A, T, R, \Omega, O, \gamma >$, where:

- S is a set of states,
- A is a set of actions,
- T is a set of conditional transition probabilities $T(s^{'}|s,a)$for the state transition $s \rightarrow s^{'}$,
- $R : S \times A \rightarrow \mathbb{R}$: reward function,
- Ω is a set of observations,
- O is a set of conditional observation probabilities $O(o|s^{'},a)$,
- $\gamma \in [0,1]$ is the discount factor.

The state space $S = \{s_0, s_1, s_2, \ldots\}$ describes the agent's world, where the transition between states depends only on the information encoded in the current state. Formally, the states have the Markov property (Eq. (2)) allowing the agent to deal only with the current state. This makes POMDPs tractable. The action space contains three possible actions $A = \{Buy, \quad Sell, \quad Hold\}$. Each action simultaneously fulfills two purposes. First, it influences the evolution of the environment in term of new states. Second, based on the current state and action, the agent gathers the next region of information. The a_t denotes the action taken in time step t.

The observation space $\Omega = \{o_0, o_1, o_2, \ldots\}$ contains all observations (a single observation at a time) received from the environment after taking an action. The transition function $T(s^{'}|s,a)$ encodes the consequences as a probability distribution. Besides, after taking the action a from the state s, the transition function returns the probability of moving to the state $s^{'}$.

$$T(s^{'}|s,a) = Pr(s^{'}|s,a), \tag{1}$$

where $Pr(s^{'}|s,a)$ is the probability of the environment transitioning to $s^{'}$ from s by performing the action a. Based on this notation, the Markov property is illustrated as: $\forall s \in S, \forall a \in A$,

$$Pr(s_{t+1}|s_0,a_0,s_1,a_1,\ldots,s_t,a_t) = Pr(s_{t+1}|s_t,a_t), \tag{2}$$

where t denotes the time step in which a particular state has been reached.

Similarly to the transition function, the observation function O represents the probability of obtaining observation o upon reaching a given state s' by taking action a.

$$O(s', a, o) = Pr(o|s', a) \tag{3}$$

In order to find the best strategy to maximize the accumulated reward $R(s, a, s')$ within a time constraint, the agent encodes a reward function upon states and actions. In other words, the agent's goal is to choose actions that maximizes its expected future discounted reward:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, a_t\right)\right] \tag{4}$$

4.2. Deep Q-learning

Scaling DRL to complex sequential decision-making problems starts by the DQL algorithm (Mnih et al., 2015). The deep reinforcement learning components, which are represented with a deep neural network (such as the policy $\pi(s, a)$), show a promising performance for approximating the action values for a given state s_t (which is fed as input to the network). At each step, the agent selects an action with respect to the action values and adds the transition $(s_t, a_t, r_{t+1}, s_{t+1})$ to a replay memory buffer based on the current state. Generally, the standard policy gradient methods execute one gradient update per data sample. Thus, the high-frequency system requires immense gradient updates. Basically, the DQL selects an action and improves its utility function, reflecting on how beneficial taking a given action is? Given a state, the decision policy learned by DQL calculate the next action to take in order to improve the Q-function using the new experience.

The DQL uses the following Bellman equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\ [R(s, a) + \gamma maxQ(s', a') - Q(s, a)], \tag{5}$$

where $Q(s, a)$ is the current Q value, $R(s, a)$ is the reward of taking action a at the state s, α is the learning rate, γ is the discount rate, and $maxQ(s', a')$ is the maximum expected future reward given the new state s' and all possible actions at the new state.

DQL is an important milestone, but several limitations of this algorithm affect its performance in different environments including the high-frequency trading environment. Especially, large states space makes it intractable to learn Q value estimates (Arulkumaran, Deisenroth, Brundage, & Bharath, 2017). In the next subsection, the trust region policy optimization is introduced and demonstrated that it can solve the policy learning problem.

4.3. Trust region policy optimization

With supervised learning, using gradient descent on the cost function surely provides excellent results with little hyperparameter tuning. But, the optimization process in deep reinforcement learning is not as obvious – the agent has many partial observed states that are hard to debug. Also, they require substantial effort in order to get better results.

The vanilla policy gradient (VPG) maximize the expected total reward by repeatedly estimating the gradient $\hat{g}$ represented in Eq. (6). Usually, the general policy optimization starts by defining the policy gradient loss $L^{PG}(\theta)$ illustrated in Eq. (7).

$$\hat{g} = \mathbb{E}_t\left[\nabla_\theta log\pi_\theta\left(a_t|s_t\right)\hat{A}_t\right]. \tag{6}$$

$$L^{PG}\left(\theta\right) = \mathbb{E}_t\left[log\pi_\theta\left(a_t|s_t\right)\hat{A}_t\right], \tag{7}$$

where π_θ is a stochastic policy. In other words, it is a neural network that takes the observed states from the environment as an input and suggests action to take as an output. At each time step t, $\hat{A}_t$ is an estimator of the advantage function, which is based on the discount sum of rewards and the baseline estimate:

$$\hat{A}_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} - V\left(S_t\right) \tag{8}$$

The discount sum of rewards (the return) is basically a weighted sum of all rewards that the agent earns during each time step in the current episode. The discounter account γ reflects that the agent cares more about the reward collected quickly, versus the same reward picked lately. The advantage function is calculated after the episode sequence has been collected, meaning that all episode rewards are already known. So, there is no guessing involved in calculating the discount value. The baseline or the value function is an estimate of the sum of rewards from the current point onward. Starting from the current state, the agent tries to guess the final return for this episode.

The advantage estimate value determines how much better the action was, based on the expectation of what would normally happen. More formally: " was the taken action better than expected or worse?".

The advantage function $\hat{A}_t$ is actually satisfying the objective function (Eq. (6)). In the first case, where the advantage function is positive, the agent has taken better actions in the sample trajectory. Remarkably, the agent increases the probability of selecting these actions again when it encounters in the same state. In the other case, where the advantage function is negative, the agent reduces the likelihood of the selected actions. Due to the fact that the agent keeps running gradient descent on one batch of collected experience, the parameters update in the policy neural network will be running so far outside from the range where the data have been collected. As a result, the agent destroys the policy if it keeps running gradient descent on a single batch of collected experience.

The key solution to this issue is by updating the policy closer to the old policy. In other words, the policy update is forced to be conservative, referring to the current policy. Schulman et al. (2015) introduce the trust region policy optimization (TRPO) objective function which needs to be maximized under a constraint on the size of the policy update. Specifically,

$$maximize_\theta \quad \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] \tag{9}$$

The optimization process in TRPO is identical to the VPG, where it guarantees that the updated policy stays near to the current policy, through adding a Kullback–Leibler (KL) constraint to the optimization objective. Precisely, this is subject to:

$$\mathbb{E}_t\left[KL\left[\pi_{\theta_{old}}(|s_t), \pi_\theta(|s_t)\right]\right] \leqslant \delta \tag{10}$$

The KL effectively ensures the closeness between the updated policy and the region where the old policy acts perfectly.

4.4. Proximal policy optimization

DQL fails on many simple environments and is insufficiently understood. VPG methods have bad data capability and stability. TRPO is relatively hard and is not consistent with uncertain environments which include noise and parameter sharing (between the value function and policy network). In this work, the PPO objective function, which enables multiple epochs of mini-batch updates, is chosen. Unlike the DQL approach learning from the stored offline data, the PPO learns online. Instead of storing the past experiences, the agent learns directly from the environment. Moreover, the KL used in TRPO adds additional overhead to the optimization process and often leads to very undesirable training behavior, especially in complex environments.

In order to uncover the KL problem, the PPO includes this extra constraint directly into the optimization objective, by defining a variable $r_t(\theta)$, which is a probability ratio:

$$r_t\left(\theta\right) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{11}$$

Given a sequence of sampled trajectories (states and actions), the $r_t(\theta) > 1$ if the action is more likely now than it was in the old version of the policy; whereas $r_t(\theta) \in [0,1[$ if the action is less likely now than it was before the last gradient step. The TRPO objective function is now written as:

$$L^{CPI}\left(\theta\right) = \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}\left(a_t|s_t\right)}\widehat{A}_t\right] = \mathbb{E}_t\left[r_t\left(\theta\right)\widehat{A}_t\right], \tag{12}$$

where the superscript *CPI* refers to conservative policy iteration. The PPO tries to optimize the following objective function:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(r_t\left(\theta\right)\widehat{A}_t, clip\left(r_t\left(\theta\right), 1-\epsilon, 1+\epsilon\right)\widehat{A}_t\right)\right], \tag{13}$$

where the superscript *CLIP* stands for the clipping objective function. In other words, this clipping prevents policy from having incentive to go far from the current policy. The PPO objective function is an expectation operator, where the calculation is performed over a batch of trajectories. The expectation operator is taken over the minimum of the default objective function (Eq. (12)) for normal policy gradients, which pushes the policy towards actions that yield a high positive advantage over the baseline. The second term is very similar to the first one, except that it contains a truncated version of the $r_t(\theta)$ ratio by applying a clipping operation between $[1-\epsilon, 1+\epsilon]$.

Fig. 2 shows one term (a single time step) of the surrogate function L^{CLIP} as a function of the probability ratio r. The red circle on each plot shows the starting point for the optimization (where $r = 1$), for positive (left) and negative advantages (right). Note that L^{CLIP} sums many of these terms.

It is worth noting that the advantage estimate can be positive or negative, and these changes affect the objective function (Eq. (13)). The loss function flattens out when $r_t(\theta)$ gets too high because the action is more likely under the current policy than it was under the old one. In this case, the agent prefers not to overdo the action update too much. Then, the objective function gets clipped to limit the effect of the gradient update. Next, if the action had an estimated negative value, the objective flattens when $r_t(\theta)$ goes near zero. This scenario corresponds to actions that are less likely now than the old policy. It also has the same effect of not overdoing a similar update, which might otherwise reduce these action probabilities to zero.

If the advantage function is negative, it ends in the region where $r_t(\theta) > 1$. In this case, the agent undoes the last gradient step by making the action less probable with an amount proportional to how much the agent failed in the first place. In addition, this is the only region where the unclipped part of the objective function has a lower value than the clipped version, as well as those returned by the minimization operator.

Basically, the PPO objective function does the same as TRPO by forcing the policy updates to be conservative if they move far from the current policy (good policy). The different and strongest points are that the PPO uses a very simple and fast objective function, in addition to not requiring the calculation of all the additional constraints or the KL divergences.

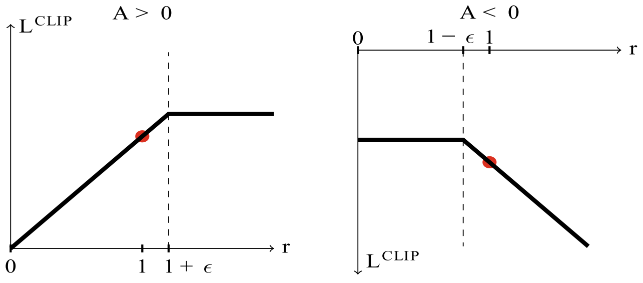

Fig. 2. Single term plot in L^{CLIP} (Schulman et al. (2017)).

4.5. *Risk curiosity-driven learning*

The trading rules are involving a massive amount of hidden experiences. Human experts intuition and experiences are the alternative keys since the environment parameters are constantly changing. The major part of the solution proposed in this paper is to define an intrinsic reward function, which is known as a curiosity (intrinsic motivation). The core idea is to minimize the action loss in order to successfully predict the next state and to encourage the agent to perform actions that reduce the uncertainty. Thus, this uncertainty grows in the areas when the agent spends less time or the area with more complex and dynamic risk. In order to model the curiosity, a risk-driven learning takes place. The agent uses this function to add more information as a feature's representation with remarkably less noise in the desired embedding space, which is compact and informationally dense. The concept of risk is hard to pin down into stock analysis and evaluation. As a major interest of traders, trading strategy optimization deals with the risk-return tradeoff.

Beta (β) (Sharpe (1964); Lintner (1965)) is a measure of the systematic risk or volatility in comparison to how correlated is the stock versus some kinds of benchmarks. In other words, the level of volatility varies in direct proportion to market beta. Beta is estimated as the covariation between a security return with the market return, which reflects the tendency of a security's returns to respond to swings in the market. Moreover, it measures volatility in relation to the market. Specifically:

$$\beta = \frac{Cov(R_i, R_M)}{Var(R_M)}, \tag{14}$$

where $Cov(R_i, R_M)$ is the covariance of the stock and the market respectively, and $Var(R_M)$ is the variance of the market. For example, if a market has $\beta = 1.5$, then the stock is expected to move 1.5 times upward or downward, as compared to the market. On the one hand, a stock that deviates very little from the market is not risky as much the opposite scenario. On the other hand, it also does not increase the theoretical potential for greater returns. The agent must choose to follow less risk and less return, or follow the return without caring out about the risk.

β risk-driven learning as a curiosity helps the agent understand whether a stock is moving as planned and how volatile it is, compared to the market. Risk-driven learning is beneficial to know how one underlying action is in relation to another one. This information allows the agent to choose a hedge to offset the risk of existing actions, or even find an interesting pairs trade.

R-Squared (R^2) is a measure of how well Beta is of best fit. It represents the linear association between a stock's performance and the market benchmark index. The R^2 is calculated between the predicted price and the actual market one. The total sum of squared, reflecting how much variation in the predicted price, is divided by the sum of squared due to regression that quantifies how much the observed responses vary from the actual price. Explicitly:

$$R^2 = \frac{\sum_i(\widehat{y}_i - \overline{y})^2}{\sum_i(y_i - \overline{y})^2}, \tag{15}$$

where for each time step i, $\widehat{y}_i$ is the estimated price and $\overline{y}$ is the mean of the market. The R^2 expresses the strength of the correlation between the agent's act and the actual market. The correlation is strong if R^2 is close to 1, and weak if R^2 is close to 0.

Risk curiosity encourages the agent to perform actions that reduce the uncertainty in the agent's ability to predict the consequence of its own action (uncertainty grows higher in areas where the agent spends less time, or in areas with complex dynamics).

5. Deep reinforcement learning based trading agents

In this section, the novel approach is proposed and the detailed learning process is also discussed.

In order to understand the proposed approach, it is useful to look at the learning process from a human expert perspective. The intelligent learning process starts by analyzing the feedback related to previous actions, where human experts analyze the past and the current factors including historical prices. Furthermore, the experts improve their behavior based on the feedback received from the environment. Specifically, through analyzing the positive profit (gains) and trading losses, the expert removes unprofitable strategies and try to find new ones, which in turn must be investigated for profitability and should be adopted or rejected. Compared to the expert's process, the proposed approach follows the same analogy. The greater the profit, the better the decision regarding the quality of the state, where the profit is the direct reward for an executed action. In this case, the decision process will be associated with a positive experience and high profit, respectively. Given a considerable learning time, this decision procedure leads to favor the possible alternatives with positive linkages. Whereas, the chosen actions with bad experiences will be avoided.

The agent is trained entirely from scratch using only self-trading without referring to any external signal. In other words, the learning process is based only on environment interaction, without using any handcrafted features. The agent's behavior is generated by a deep neural network that receives states as input and outputs an action. The neural network architecture, which applies a mapping from states observation sequences to actions, uses the novel fundamental concepts of proximal policy optimization. This allows the agent to be more creative in finding new strategies, by imitating the expert's process which is used to train deep reinforcement learning agents. The policy network is parametrized by the deep neural network shown in Fig. 3. The neural network components are chosen using basic models evolution over a small sampled data. First, an initial population of models containing different hidden layers, neurons size, and activation functions, is initialized. Then, the unstable models, in terms of the provided rewards, are eliminated. Last, the model, which performs better compared to other models, is chosen. Besides, during the backpropagation step, the gradients keep becoming smaller until they vanish. However, no gradients mean no learning. A remedy to this problem is to use the ReLU activation function, which does not squeeze information and has much lower run time, making it more computationally efficient. The first three layers have 64, 32, 8 units respectively and use the ReLU activation function. The last layer is a linear output layer with 3 units, one for each action. The neural network uses Adam optimizer (Kingma & Ba (2014)) with a learning rate of 0.001. The parameters used in this neural network are summarized in Table 2.

In order to improve the training process, a continuous virtual environment has been created with different versions of agents trading against one another, akin to how experts experience the trading process. New stock markets datasets have dynamically been added to this virtual environment, by branching from the existing datasets; each version of the agent then learns from these new datasets. To encourage diversity in the virtual environment, various risky datasets are added to encourage the agents to defeat different risk levels. The agents start out with random parameters and use around 504 different training datasets to learn better parameters. Fig. 4 illustrates 24 examples of these datasets, which are very risky. At each transaction, the agent gets positive rewards when it makes a positive income and negative rewards when it loses money. The agent applies the PPO algorithm to update the parameters of the neural network, making actions that occur soon before positive reward more likely and those soon before negative reward less likely. This new form of training takes the ideas of agent training and policy improvement further, creating a process that continually explores the huge environment states. Thus, this process ensures that each improved version performs well against the uncertainty of the states and defeats earlier versions. As the learning progresses and new versions of agents are created, new counter-strategies emerge and are able to defeat the earlier strategies. While some new agents perform a strategy that is merely a refinement of a previous one, others identify drastically new strategies. For example, basic strategies such as executing trader orders targeting the stock peaks. These risky strategies have been discarded as training progressed, leading to better strategies. For instance, gaining the nearest action strength and increasing its chance to be chosen again. This process is similar to the trader experts for discovering new strategies and defeating previously favored approaches. The policy update rule is an efficient and novel on-policy algorithm which does not use any experience replay or imitation learning, where the policy update is forced to be conservative referring to the current/good policy.

Fig. 5 shows the continuous virtual environment training a population of evolutive agents together. This environment provides a diverse set of opponents to trade with and is also used to evolve the internal rewards and hyperparameters of agents and learning process. Each circle represents an agent in the population, with the final circle representing the strength policy. The policy operates at two different level scales: the risk and the trading position. Then, the policy neural network suggests

Table 2
The parameter setup for the neural network.

Parameter	value
Max rollouts per state	~ 2
Discount (γ)	0.99
Learning rate	0.001
Loss	Mean Square Error

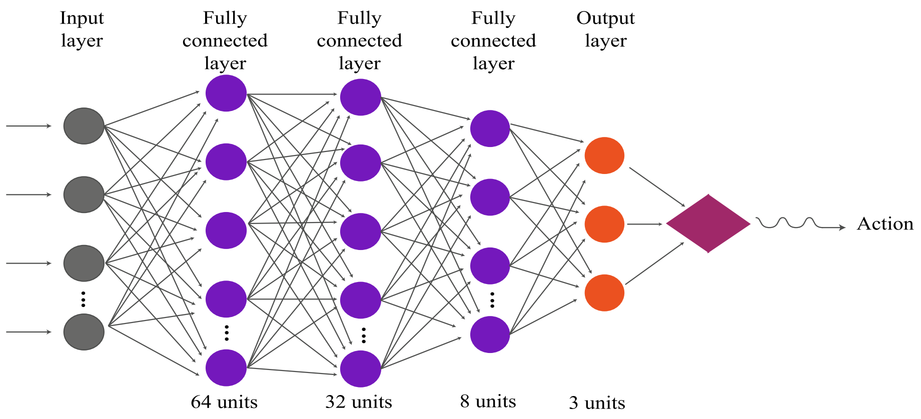

Fig. 3. The policy neural network.

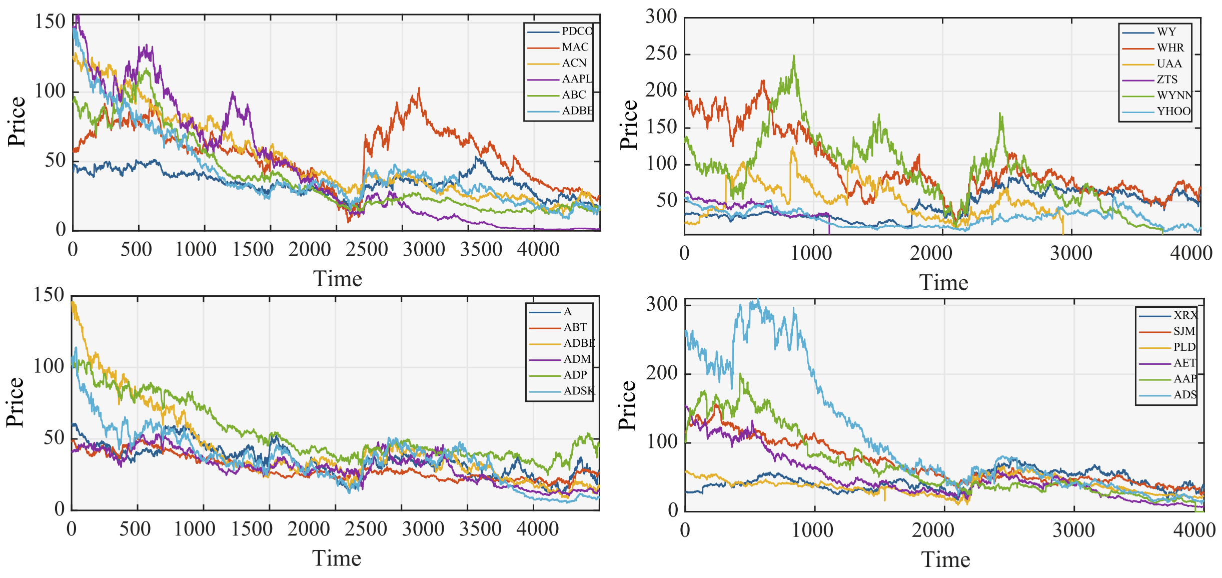

Fig. 4. 24 example of different training data (the total used data is 504).

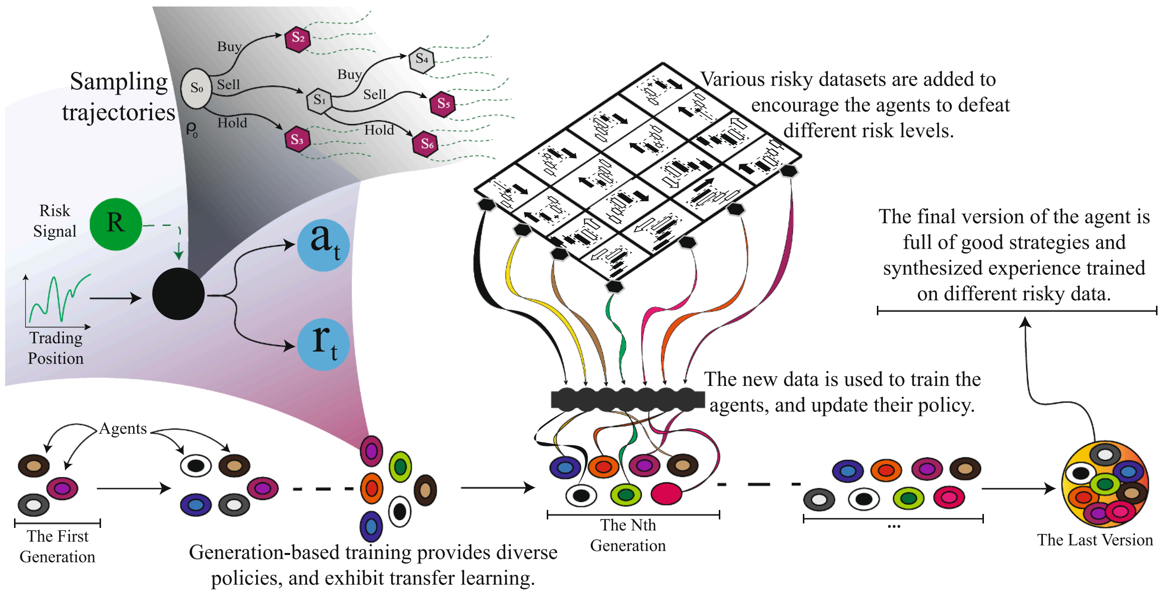
Sampling trajectories
Buy
Sell
Hold
Risk Signal
R
Trading Position
a_t
r_t
Various risky datasets are added to encourage the agents to defeat different risk levels.
The final version of the agent is full of good strategies and synthesized experience trained on different risky data.
The new data is used to train the agents, and update their policy.
Agents
The First Generation
Generation-based training provides diverse policies, and exhibit transfer learning.
The Nth Generation
...
The Last Version

Fig. 5. The continuous virtual training environment.

the action to take and calculate the reward associated with this taken action. It starts out completely random with no knowledge of the world and simply trades against other agents. This means it climbs the ladder of skill level until it is able to reach the performance of mastering the task. These agents exhibit transfer learning, applying skills learned in one setting to succeed in another never-before-seen one. In one case, the agent trained on the self-trade using a good data set is faced with risky data while being perturbed by other agent results. The agent manages to rectify its own behavior despite never seeing the risky data or observing another agent success. The final version of the trained agent has learned a rich representation of different states and is full of good strategies and synthesized experience.

This work uses the vine method (Schulman et al. (2015)), which generates multiple trajectories from each state in the rollout set. Precisely, as shown in Fig. 6, the vine estimation procedure sample $s_0 \sim \rho_0$ and simulate the policy π_{θ_i} to generate a number of trajectories. Next, along these trajectories a rollout subset of N states is chosen $(s_0, s_1, \dots, s_N)$. In the context of current problem, since the action space contains just

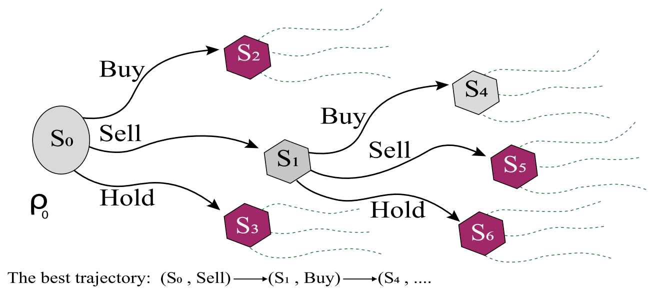

Fig. 6. The system generates a set of "trunk" trajectories, and then generate "branch" rollouts from a subset of the reached states. For each of these states s_n, the system performs the known three actions and perform a rollout after each action. This process attempts to differentiate between the best trajectories as well as the bad ones.

three actions. The system can generate a rollout according to every possible action from a given state. The contribution to $L_n(\theta_{old})$from a single state s_n is as follows:

$$L_n\left(\theta\right) = \sum_{k=1}^{3} \pi_\theta\left(a_k|s_n\right)Q_\pi\left(s_n, a_k\right), \tag{16}$$

where $Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1},a_{t+1},\dots}\left[\sum_{l=0}^{\infty}\gamma^l r(s_{t+l})\right]$. Fig. 6 illustrates the sampling method, revealing the best trajectory. Given a state s_0, the algorithm generates a number of trajectories from this state and tries every possible action which generates good and bad trajectories, and thus, making the good trajectories and actions more probable.

The proposed algorithm consists of two major training threads, which can be decoupled from each other. In the first one, the current policy interacts with the environment and generates episode sequences for which the agent immediately calculates the advantage function using the fitted baseline estimate for the state values. For a considerable number of episodes, the second thread collects that experience and optimizes it using clips PPO objective function. In addition, the PPO updates the policy network while ensuring the deviation from the old policy is relatively small. The agent avoids making too aggressive moves to assure that the new policy performs better. Thus, excessively updating policy using samples of behavior policy could make it stray too far from the real behavior resulting in the policy degradation. In this sense, proper steps should be taken to ensure a safe search. However, Fig. 7 illustrates the general training overview. Rollout workers gather experience through interacting with the environment and generate episodes sequences. Then, they sample actions from the policy given the current observation and send the data to the optimizer to perform gradient updates. The optimizer publishes the parameter versions to model parameters.

The final objective function used to train the agent is given as follow:

$$L^{PPO}\left(\theta\right) = L_t^{CLIP+VF+S}\left(\theta\right) = \mathbb{E}_t\left[L_t^{CLIP}\left(\theta\right), c_1L_t^{VF}\left(\theta\right), c_2S\left[\pi_\theta\right]\left(s_t\right)\right], \tag{17}$$

where c_1 and c_2 are coefficients, and $L_t^{VF}$is a squared-error loss $(V_0(s_t) - V_t^{targ})^2$, which is concerned by updating the baseline network. This error especially estimates how good it is this state. More specifically, what the average amount of discounted reward that the agent expects to get from this point onward is. The value estimation network shares a large portion of its parameter space with the policy network. Thus, whether the system is trying to estimate the value of the current state or to take the best current action, it needs similar features extraction pipelines from the current state observation, so these parts of the network are simply shared. The $S\left[\pi_\theta\right](s_t)$ denotes the entropy term charged of ensuring that the agent does sufficient exploration during training. In contrast to discrete action policies that output the action choice probabilities, the PPO policy head outputs the parameters of a Gaussian distribution for each available action type. The policy uses these particular distributions as a sample to get a continuous output value from each action head.

Deep reinforcement learning-based trading agent training algorithm is shown below (Algorithm 1). As the training goes, each state price $\widehat{y}_i$is fitted into the risk function (Eq. (15)), which returns the R^2 value indicating the quantity of risk from this state onward. The agent training process is based on fixed-length trajectory segments sampled by the vine method (Fig. 6). At each iteration, N workers collect parallelly T timesteps of data. Then, the agent constructs the surrogate loss on these NT timesteps of data, and optimizes it with minibatch Adam (Kingma & Ba, 2014), for K epochs. The agent training process uses multiple epochs of Adam optimizer to perform each policy update. A significantly greater exploration becomes necessary, where the agent must explore enough trajectory to accumulate more experience in a shorter time. This causes a faster adjustment and produces interesting results.

The algorithm repeatedly performs the following steps:

- Use the vine procedure to collect a set of state-action pairs.
- Run policy $\pi_{\theta_{old}}$ by averaging over samples and construct the estimated advantages.
- Finally, Approximately solve the optimization problem to update the policy's parameter vector θ.

Algorithm 1: Deep reinforcement learning based trading agent training algorithm.

```
1: for iteration = $1, 2, \dots$ do
2: for worker = $1, 2, \dots, N$ do
3: Run policy $\pi_{\theta_{old}}$ over a set of collected trajectories for T timesteps.
4: Estimate advantages $\widehat{A}_1, \dots, \widehat{A}_T$.
5: end for
6: Compute policy update, by taking K steps of minibatch SGD (via Adam (Kingma and Ba (2014))), using:
   $L^{PPO}(\theta)$ (Eq. (17)).
7: $\theta_{old} \leftarrow \theta$
8: end for
```

The central idea of the PPO is to avoid having too large policy update, which improves the stability of the agent training by limiting the policy update at each training step. Moreover, the PPO uses multiple epochs of stochastic gradient ascent to perform each policy update. Specifically for each iteration, after sampling the environment with π_{old} (line 3) and starting running the optimization (line 6), the policy π is exactly equal to π_{old}. So, at first, none of the updates is clipped which guarantees learning something from these examples. However, as the policy $\pi$is updated using multiple epochs, the objective starts hitting the clipping limits, the gradient goes to 0 for those samples, and the training will gradually stops.

In Section 6, the validity of the proposal is tested and proved. Moreover, the proposal's advantages are demonstrated through a comparative analysis.

6. Results and discussion

In this section, the validity of the proposal is tested and proved.

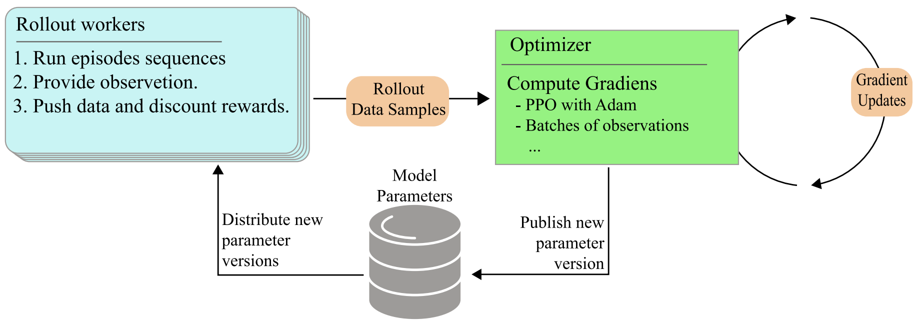
Rollout workers
1. Run episodes sequences
2. Provide observetion.
3. Push data and discount rewards.
Rollout Data Samples
Optimizer
Compute Gradiens
- PPO with Adam
- Batches of observations
...
Gradient Updates
Model Parameters
Distribute new parameter versions
Publish new parameter version

Fig. 7. Training Overview.

Moreover, the proposal's advantages are also demonstrated through a detailed analysis.

In the field of financial theory, financial market trading has been conceiving much attention. However, a large body of works applies deep reinforcement learning to video games and robotics, but there is relatively little focus on how to apply these algorithms to financial trading. Furthermore, no methodology has been assumed to accurately predict the market movement dynamically. Nonetheless, recent developments in similar environments have pushed researchers towards exciting new horizons.

In order to demonstrate the effectiveness and efficiency of the proposed approach, the closing prices of 8 stocks are used for evaluation to determine if the best action to take at a given state is to buy, sell, or hold. On the one hand, the proposed system is evaluated in terms of total profit as well as in the cumulated reward using the closing prices of 8 stocks presented in Fig. 8. On the other hand, this system is compared to DQL system on the Gold stock in terms of total profit, cumulated reward, and on 12 episodes. Each episode contains very long sequences (states and actions). Notice that the Gold stock is chosen because it represents a low risk (0.7) comparing to other stocks. Moreover, this system is compared to two different methods. The first consists of the seasonality events (Eilers et al., 2014), where the second uses the deep recurrent Q-network (Huang, 2018). In addition, this system is compared to price prediction-based trading systems. Finally, the curiosity-driven risk is illustrated to justify the gains and the number of transactions in each stock.

Table 3 provides the agent performance for each currency pair in terms of the number of trades and the total profit over these trades. The agent produced a larger gain overall stocks. The obtained results over the total profit (buying and selling) are plotted in Fig. 9. Together these results provide important insights into agent performances because the market prices are heavily changing. However, the agent behavior derived by risk curiosity and the conservative policy update is pushing the trading process to attend the best results. In other words, the partially observed states are transformed into more likely states using the curiosity-driven risk, and the price changing is overtaken by updating the policy in the trusted region.

In order to understand the theoretical and practical value of the proposed system, Fig. 10 shows the systems' cumulative rewards over different stocks. The obtained results reflect that the agent achieves the targeted goal by maximizing the expected cumulative reward under states uncertainty. Since every DRL system concerns to maximize the reward, the PPO acts carefully in keeping the high rewards, as well as taking the appropriate actions. Additionally, the action selection mechanism handles the risky actions (buy or sell in the wrong moments and transactions) by holding the trading. Moreover, the agent keeps learning in the background in order to improve its action quality in high-risk states.

In RL, the agent attempts to learn the optimal policy while interacting with its environment. Over various domains, DQL shows useful behavior (Jeong & Kim, 2019), but it becomes ineffective in big state space environments. Conversely, the random behavior presented in the high-frequency trading needs more than just a price prediction because as the training goes, the prediction values get diminished and flattened. The differences between DQL and the proposed system over 12 episodes in a single stock are highlighted in Fig. 11. Evidently, the DQL presents a weak behavior on all episodes, contrary to the proposed system which acts perfectly and achieves a high profit. Fig. 12 exposes that the proposed system surpasses the DQL in terms of maximizing the profit. Besides, it outperforms the DQL in many axes. On the one hand, the agent is acting in real time scale, contrary to the DQL which needs to store the

Table 3
Trading Statistics.

Stock	Profit	Number of trades
BTCUSDT	221,482	3009
SP500	105,108	1292
Google	13,204	398
Apple	4710	816
Facebook	3307	299
Gold	1911	385
Baba	777	209
SFix	114	137

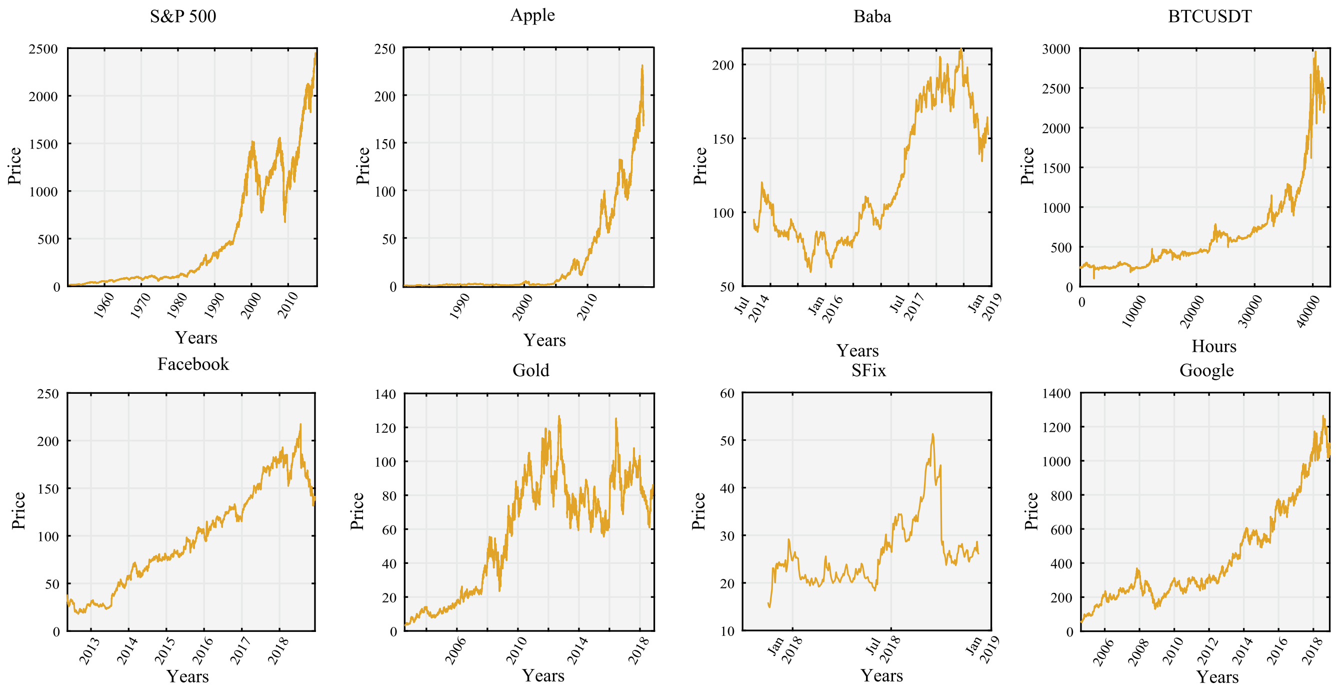

Fig. 8. Real prices of the 8 stocks.

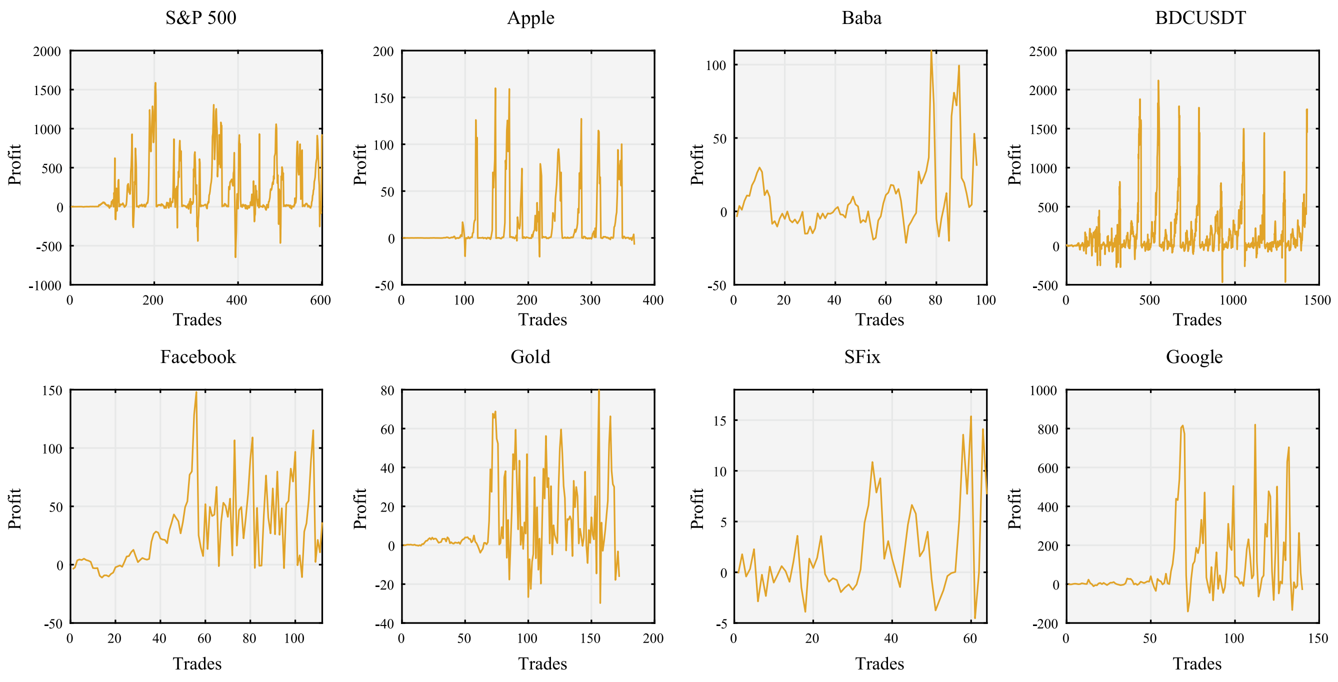

Fig. 9. Total profit over 8 stocks.

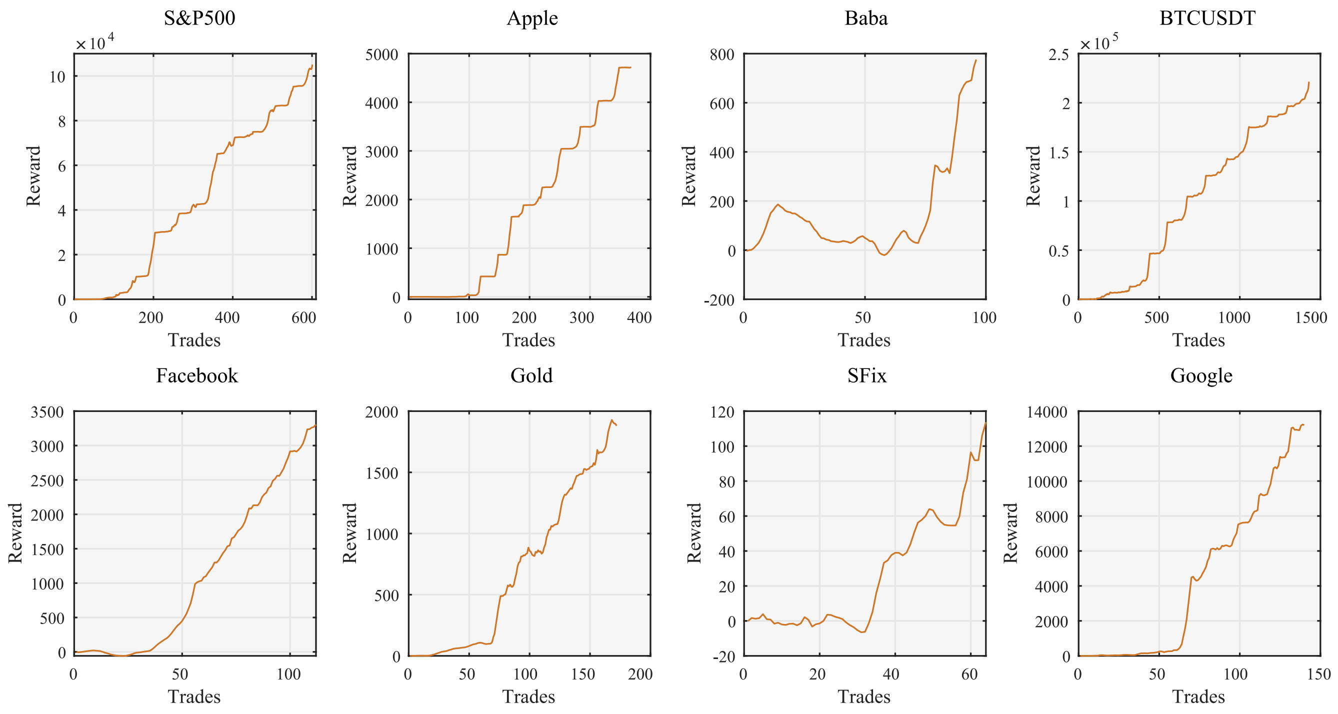

Fig. 10. Rewards over different learning step.

previous experiences on a reply buffer. On the other hand, the fast convergence, specifically, the proposed system starts making positive income from step 57, as opposed to the DQL which attends the same positive income from step 245. Moreover, the DQL attends the max reward starting from 2746, which the proposed system achieves from the 173. Fig. 13 illustrates the cumulative rewards over learning steps for both systems, and Table 4 summarizes the discussed results.

With an eye toward clarifying the risk curiosity, Table 5 shows the risk value for every stock. The higher the value of the risk, the less risky the market is. Jointly to the results from Table 3, it is apparent that every stock which makes an extremely profit has less risk, comparing to those which have less profit. More formally, while the R^2 is close to 1, the system would make a huge profit. On the other side, if R^2 is close to 0, then the profitability is weaker.

It is worth mentioning that the proposed approach is successfully achieving the best scores on discrete observations and partially observed states. Overall, the agent has used a neural network to approximate the ideal function that maps the observations to the best action to take in a given state. The trained agent performs well by successfully applying the main capabilities and experiences it gains during training. Performing

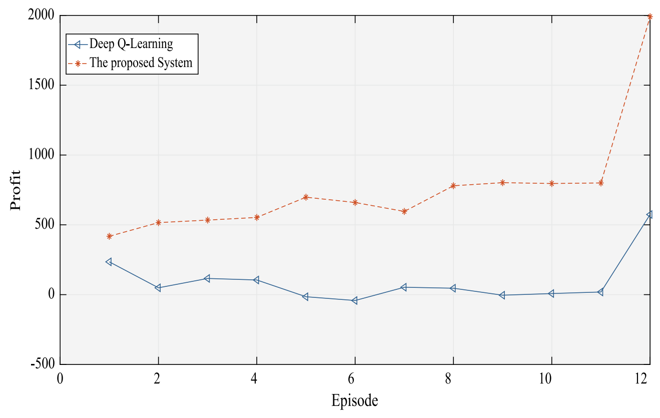

Fig. 11. DQL compared to the proposed system over 12 episodes.

the PPO over trust-region founded along with the training process helps the agent make profitable results. Additionally, the risk curiosity pushes the learning process to its limits, specifically during the training phase if the agent detects that the market is risky. Then, it reduces the potential of trading by minimizing the number of transactions. Besides, the agent keeps learning in the background to improve its future experiences for the risk curiosity mechanism, as well as handling the problem of taking actions in an uncertain environment. The achieved capabilities demonstrated by the presented results expose the success in the financial knowledge capitalization field over the high-velocity environment.

Most of the existing works attempt to predict stock price direction using multiple classifiers, either by focusing on a specific financial market application or by focusing on a family of machine learning algorithms (Ballings, Van den Poel, Hespeels, & Gryp (2015)). Table 6

Table 4
The proposed system performance Vs DQL system.

System	Profit	Convergence	Positive income	max episodes	Act
The Proposed System	1910 $	Fast	57	173	online
DQL system	574 $	Slow	245	2746	offline

Table 5
The R^2 risk for every stock.

Stock	R^2Risk
BTCUSDT	0.74
SP500	0.75
Google	0.53
Apple	0.53
Facebook	0.57
GOLD	0.70
BABA	0.30
SFIX	0.49

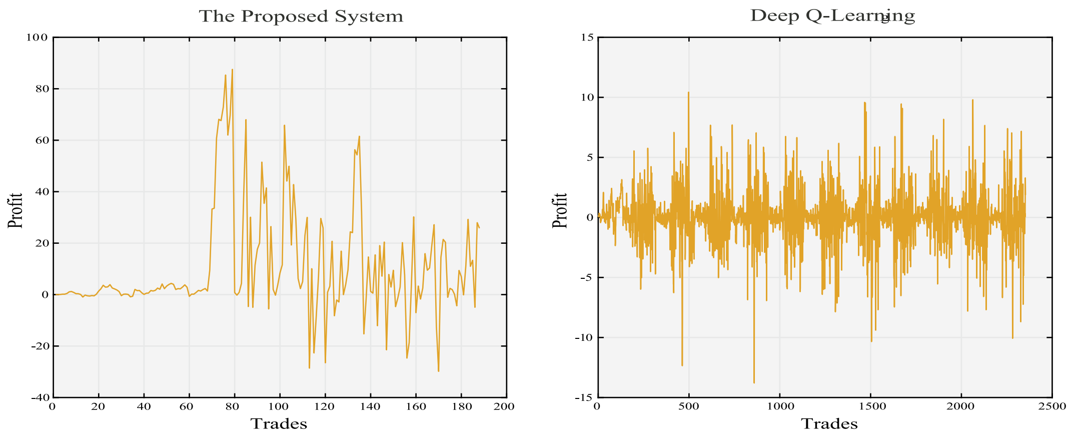

Fig. 12. DQL compared to the system over total profit.

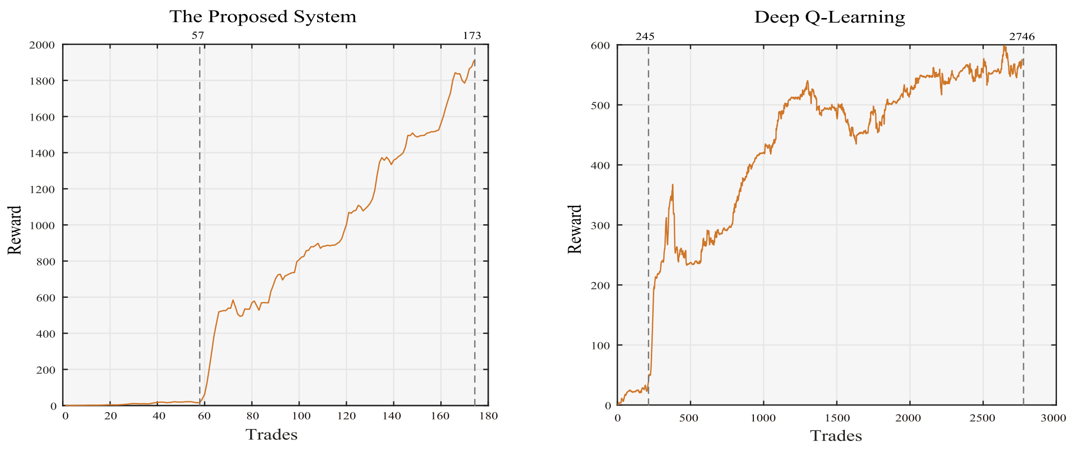

Fig. 13. Rewards over one stock for both systems.

compares the proposed model with Eilers et al. (2014) and Huang (2018) methods. Moreover, it represents how price prediction methods are weak in trading stock markets. The first column shows the best results produced by all systems, where the proposed method make the best income compared to the other methods, and the third column shows the number of trading transactions. The proposed system focuses more on the market risk, rather than caring more about seasoning events (Eilers et al. (2014)) which can be riskier either. In addition, the deep recurrent Q-network attends weak results comparing to the proposed system.

The key idea of the majority of price prediction works is as follows: First, a predictive model is trained (e.g., a neural network) on historical data to forecast the asset's price change. Then, they feed these forecasts into a trading module to derive the trading actions. For example, the system buys the financial asset in case the forecast surpasses a certain threshold (Liu, Si, Zhang, & Zhou, 2018; Luo, You, Xu, & Peng, 2017). Despite its losses, this two-step approach has various shortcomings that lead to weak performance. On the one hand, the predictive model optimization such as the minimization of the prediction error is not necessarily straightforward with the ultimate goal of the model, which attempts to maximize the return. On the other hand, the trading module usually accepts only the forecast as input and disregards other valuable information. Finally, the environment complexity imposes exogenous constraints, such as the hidden details of other agents. Reinforcement learning promises to overcome these limitations. Specifically, the prediction and the rules-based policy are integrated into one single step and jointly optimized in line with the risk-curiosity. Hereby, the trained agent learns by interacting with the environment allowing it to incorporate the aforementioned constraints of POMDP into its decision-making process.

In comparison to the aforementioned methods, the proposed approach yields better and more accurate performances. However, according to the above results, these methods retain certain shortcomings resulting in distortion and loss which may certainly degrade their credibility. Alternatively, the key idea of the proposal is to focus on policy updates especially when the risk-curiosity mentions a high risk. The foremost objective is to carefully learn a rule-based policy based on uncertainty rather than reducing the potential of trading in risky situations. Practically, based on the state uncertainty, the risk indices are first identified, and the obtained risk values are then considered when choosing the alternatives' overall actions. Moreover, if the risk is high under exploration, the agent reduces the potential of transactions as well as it rectifies the learning trajectory according to vine sampling method. In other words, the agent attempts to learn strong actions in risky and uncertain states.

The capability of the suggested risk curiosity function cannot be achieved when using sentiment analysis methods. Furthermore, the introduced policy learning fundaments give summarized insights and experiences about the environment performances. Last but not least, the offered risk curiosity approach helps in achieving high results by considering different data collections. This additional feature is necessary for the trained agent and drives the learning process across uncertain states and situations. Besides, comparing to Eilers et al. (2014) which considers seasonality events to execute trading orders, this system handles the uncertainty, meanwhile, it detects the safe regions where the agent can make considerable gains. Therefore, these regions can uncover important events as well. Even if this case was partially considered, it remains one of the contributions of the proposed model. In view of this, the self-learned rules driven by the risk curiosity are important and strong in the financial field. The techniques behind this project could be useful in solving other problems, involving very long sequences of actions based on imperfect information and risky decision-making situations, which appear in many real-world challenges, such as touristic domain (Li, Chen, Wang, & Ming (2018)), applied energy like wind speed area (Jiang, Yang, & Heng (2019)), or energy consumption (Xiao, Dong, & Dong (2018)), where it is essential to address complex edge cases.

Table 6
Comparison of the proposed system with four price prediction models, seasonality events approach, and deep recurrent Q-network, on S&P 500 stock.

System	Best	# Transactions
K-Nearest Neighbor	202.95	-
Gradient Boosting Model	351.54	-
Random Forest	225.15	-
Support Vector Machines	210.22	-
Eilers et al. (2014)	133.99	314
Huang (2018)	257.19	331
The proposed system	**1588.22**	601

7. Conclusion

In this paper, a novel DRL based trading agent approach has been proposed. This approach consists of a continuous virtual environment containing different versions of agents trading against one another. For policy learning, every agent relies on the fundamental concepts of the PPO, which have pushed the policy towards actions that yield a high performance. In contrast to existing methods, the final version of the trained agent has explicitly demonstrated the proficiency of the rule-based policy guided by the risk-curiosity in maximizing the return while restraining the risk simultaneously. In addition, the agent has shown its own creativity in a different dimension by evaluating actions contextually in the environment without refereeing to any external signal, which strengthens its ability to uncover the uncertainty. Despite the high dynamicity of the environment, the performance achieved by the agent accumulates significant experience in a small period of time, which can be extended into real-world situations. Risk curiosity-driven learning is heavily laden with signals to find salient relationships between actions and market behavior and has steadily and progressively improved actions quality. The consistency and reliability of the proposed system have been demonstrated by comparisons with other related methods. The presented results indicate that the executed trades are more prominent and generate a considerable income.

In future researches, the proposed model will be extended to respond to portfolio formulation. Precisely, a factor-investing model will be developed to find the predictive power of multiple factors in a single step by discovering their appropriate weights and eliminating correlated/collinear factors. The central hypothesis reveals that the weights should not be equal by design. Instead, they should be associated with the quality of each factor. The highest weight corresponds to the most significant factor in decreasing order. First, each state used in the current work is extended by adding multiple factors associated to each stock. Next, the action space is rectified, where the policy network outputs the stocks weights and an applicable delay. In other words, each ranking episode is applicable over a specific period. Therefore, for each episode, the related weights are optimally calculated, and the delay period is defined. The network inputs consist of the defined state alongside the past period delay. The concept of transfer learning is also incorporated alongside the ranking to avoid the inappropriate experience-ranking scheme. Interestingly, this new model is a multi-task agent, by which stocks from the defined universe are ranked, the period of application the ranking schema is defined, and the transfer learning is guaranteed to avoid the market crisis.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Almahdi, S., & Yang, S. Y. (2017). An adaptive portfolio trading system: a risk-return portfolio optimization using recurrent reinforcement learning with expected maximum drawdown. *Expert Systems with Applications, 87*, 267–279. https://doi.org/10.1016/j.eswa.2017.06.023

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine, 34*(6), 26–38. https://doi.org/10.1109/MSP.2017.2743240

Azhikodan, A. R., Bhat, A. G. K., & Jadhav, M. V. (2019). Stock trading bot using deep reinforcement learning. In H. S. Saini, R. Sayal, A. Govardhan, & R. Buyya (Eds.), *Innovations in computer science and engineering* (pp. 41–49). Singapore: Springer Singapore. https://doi.org/10.1007/978-981-10-8201-6_5.

Ballings, M., Van den Poel, D., Hespeels, N., & Gryp, R. (2015). Evaluating multiple classifiers for stock price direction prediction. *Expert Systems with Applications, 42* (20), 7046–7056. https://doi.org/10.1016/j.eswa.2015.05.013

Buehler, H., Gonon, L., Teichmann, J., & Wood, B. (2019). Deep hedging. *Quantitative Finance, 19*(8), 1271–1291. https://doi.org/10.1080/14697688.2019.1571683. DOI: 10.1080/14697688.2019.1571683 arXiv:https://doi.org/10.1080/14697688.2019.1571683.

Carapuço, J., Neves, R., & Horta, N. (2018). Reinforcement learning applied to Forex trading. *Applied Soft Computing, 73*, 783–794. https://doi.org/10.1016/j.asoc.2018.09.017

Chaboud, A. P., Chiquoine, B., Hjalmarsson, E., & Vega, C. (2014). Rise of the machines: Algorithmic trading in the foreign exchange market. *The Journal of Finance, 69*(5), 2045–2084. https://doi.org/10.1111/jofi.12186. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/jofi.12186.

Chen, C. T., Chen, A., & Huang, S. (2018). Cloning strategies from trading records using agent-based reinforcement learning algorithm. In *2018 IEEE international conference on agents (ICA)* (pp. 34–37). https://doi.org/10.1109/AGENTS.2018.8460078

Chow, Y., Ghavamzadeh, M., Janson, L., & Pavone, M. (2017). Risk-constrained reinforcement learning with percentile risk criteria. *The Journal of Machine Learning Research, 18*(1), 6070–6120.

Dash, R., & Dash, P. (2016). An evolutionary hybrid fuzzy computationally efficient egarch model for volatility prediction. *Applied Soft Computing, 45*, 40–60. https://doi.org/10.1016/j.asoc.2016.04.014

Deng, Y., Bao, F., Kong, Y., Ren, Z., & Dai, Q. (2016). Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems, 28*(3), 653–664. https://doi.org/10.1109/TNNLS.2016.2522401

Deng, Y., Kong, Y., Bao, F., & Dai, Q. (2015). Sparse coding-inspired optimal trading system for hft industry. *IEEE Transactions on Industrial Informatics, 11*(2), 467–475. https://doi.org/10.1109/TII.2015.2404299

Di Persio, L., & Honchar, O. (2017). Recurrent neural networks approach to the financial forecast of google assets. *International Journal of Mathematics and Computers in Simulation, 11*, 7–13.

Du, X., Zhai, J. & Lv, K. (2016). Algorithm trading using q-learning and recurrent reinforcement learning. Positions 1, 1.

Eilers, D., Dunis, C. L., von Mettenheim, H. J. & Breitner, M. H. (2014). Intelligent trading of seasonal effects: A decision support algorithm based on reinforcement learning. Decision Support Systems 64, 100–108. http: http://www.sciencedirect.com/science/article/pii/S0167923614001523. https://doi.org/10.1016/j.dss.2014.04.011.

Ganesh, P. & Rakheja, P. (2018). Deep reinforcement learning in high frequency trading. arXiv preprint arXiv:180901506; arXiv:1809.01506.

Goldkamp, J. & Dehghanimohammadabadi, M. (2019). Evolutionary multi-objective optimization for multivariate pairs trading. Expert Systems with Applications 135, 113–128. http: http://www.sciencedirect.com/science/article/pii/S0957417419303811. doi: 10.1016/j.eswa.2019.05.046.

Huang, B., Huan, Y., Xu, L. D., Zheng, L., & Zou, Z. (2019). Automated trading systems statistical and machine learning methods and hardware implementation: A survey. *Enterprise Information Systems, 13*(1), 132–144. https://doi.org/10.1080/17517575.2018.1493145. DOI: 10.1080/17517575.2018.1493145 arXiv:https://doi.org/10.1080/17517575.2018.1493145.

Huang, C. Y. (2018). Financial trading as a game: A deep reinforcement learning approach. arXiv preprint arXiv:180702787; arXiv:1807.02787.

Hull, J. C. (2014). Options, futures, and other derivatives. *Pearson Education*.

James, C., Alrajeh, D. & Dickens, L. (2015). An investigation into the use of reinforcement learning techniques within the algorithmic trading domain. Imperial College London: London, UK.

Jeong, G., & Kim, H. Y. (2019). Improving financial trading decisions using deep q-learning: Predicting the number of shares, action strategies, and transfer learning. *Expert Systems with Applications, 117*. https://doi.org/10.1016/j.eswa.2018.09.036, 125–138.

Jiang, P., Yang, H., & Heng, J. (2019). A hybrid forecasting system based on fuzzy time series and multi-objective optimization for wind speed forecasting. *Applied Energy*. https://doi.org/10.1016/j.apenergy.2018.11.012

Jiang, Z., Xu, D. & Liang, J. (2017). A deep reinforcement learning framework for the financial portfolio management problem. arXiv preprint arXiv:170610059.

Jin, F., Self, N., Saraf, P., Butler, P., Wang, W. & Ramakrishnan, N. (2013). Forex-foreteller: Currency trend modeling using news articles. In Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining. New York, NY, USA: ACM; KDD '13 (pp. 1470–1473). http: http://doi.acm.org/10.1145/2487575.2487710. DOI: 10.1145/2487575.2487710.

Jin, O. & El-Saawy, H. (2016). Portfolio management using reinforcement learning.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence, 101*(1), 99–134. https://doi.org/10.1016/S0004-3702(98)00023-X

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980; arXiv:1412.6980.

Legg, S., & Hutter, M. (2007). Universal intelligence: A definition of machine intelligence. *Minds and Machines, 17*(4), 391–444. https://doi.org/10.1007/s11023-007-9079-x. DOI: 10.1007/s11023-007-9079-x.

Lei, K., Zhang, B., Li, Y., Yang, M. & Shen, Y. (2020). Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading. Expert Systems with Applications, 140, 112872. http: http://www.sciencedirect.com/science/article/pii/S0957417419305822. doi: 10.1016/j.eswa.2019.112872.

Li, S., Chen, T., Wang, L., & Ming, C. (2018). Effective tourist volume forecasting supported by pca and improved bpnn using baidu index. *Tourism Management*. https://doi.org/10.1016/j.tourman.2018.03.006

Li, Y., Zheng, W., & Zheng, Z. (2019). Deep robust reinforcement learning for practical algorithmic trading. *IEEE Access, 7*, 108014–108022. https://doi.org/10.1109/ACCESS.2019.2932789

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. & Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:150902971; arXiv:1509.02971.

Lintner, J. (1965). Security prices, risk, and maximal gains from diversification. *The Journal of Finance, 20*(4), 587–615. https://doi.org/10.1111/j.1540-6261.1965.tb02930.x. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.1965.tb02930.x.

Liu, J., Si, Y. W., Zhang, D., & Zhou, L. (2018). Trend following in financial time series with multi-objective optimization. *Applied Soft Computing, 66*, 149–167. https://doi.org/10.1016/j.asoc.2018.02.014

Lu, D. W. (2017). Agent inspired trading using recurrent reinforcement learning and lstm neural networks. arXiv preprint arXiv:170707338.

Luo, L., You, S., Xu, Y., & Peng, H. (2017). Improving the integration of piece wise linear representation and weighted support vector machine for stock trading signal prediction. *Applied Soft Computing*. https://doi.org/10.1016/j.asoc.2017.03.007

Machado, J., Neves, R. & Horta, N. (2015). Developing multi-time frame trading rules with a trend following strategy, using ga. In Proceedings of the companion publication of the 2015 annual conference on genetic and evolutionary computation. New York, NY, USA: Association for Computing Machinery; GECCO Companion '15 (pp. 765–766). http: doi: 10.1145/2739482.2764885. DOI: 10.1145/2739482.2764885.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., Silver, D. & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In Proceedings of the 33rd international conference on international conference on machine learning (Vol. 48, pp. 1928–1937). JMLR.org; ICML'16.

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. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533. https://doi.org/10.1038/nature14236. DOI: 10.1038/nature14236.

Moody, J., & Saffell, M. (2001). Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks, 12*(4), 875–889. https://doi.org/10.1109/72.935097

Neely, C. J., Rapach, D. E., Tu, J., & Zhou, G. (2014). Forecasting the equity risk premium: The role of technical indicators. *Management Science, 60*(7), 1772–1791. https://doi.org/10.1287/mnsc.2013.1838. DOI: 10.1287/mnsc.2013.1838 arXiv: https://doi.org/10.1287/mnsc.2013.1838.

Park, H., Sim, M. K. & Choi, D. G. (2020). An intelligent financial portfolio trading strategy using deep q-learning. Expert Systems with Applications, 158, 113573. http: http://www.sciencedirect.com/science/article/pii/S0957417420303973. doi: 10.1016/j.eswa.2020.113573.

Sastry, R., & Thompson, R. (2019). Strategic trading with risk aversion and information flow. *Journal of Financial Markets, 44*, 1–16. https://doi.org/10.1016/j.finmar.2018.12.004

Schulman, J., Levine, S., Abbeel, P., Jordan, M. & Moritz, P. (2015). Trust region policy optimization. In International conference on machine learning (pp. 1889–1897). arXiv:1502.05477.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:170706347; arXiv:1707.06347.

Serrano, W. (2018). Fintech model: The random neural network with genetic algorithm. Procedia Computer Science, 126, 537–546. http: http://www.sciencedirect.com/science/article/pii/S187705091831264X. doi: 10.1016/j.procs.2018.07.288; knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 22nd International Conference, KES-2018, Belgrade, Serbia.

Sharpe, W. F. (1964). Capital asset prices: A theory of market equilibrium under conditions of risk. *The Journal of Finance, 19*(3), 425–442. https://doi.org/10.1111/j.1540-6261.1964.tb02865.x. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.1964.tb02865.x.

Si, W., Li, J., Ding, P. & Rao, R. (2017). A multi-objective deep reinforcement learning approach for stock index future's intraday trading. In 2017 10th International symposium on computational intelligence and design (ISCID) (Vol. 2, pp. 431–436). DOI: 10.1109/ISCID.2017.210.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature, 529*(7587), 484–489. https://doi.org/10.1038/nature16961. DOI: 10.1038/nature16961.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. & Riedmiller, M. (2014). Deterministic policy gradient algorithms. In Proceedings of the 31st international conference on international conference on machine learning (Vol. 32, pp. I-387–I-395). JMLR.org; ICML'14.

Spooner, T., Fearnley, J., Savani, R. & Koukorinis, A. (2018). Market making via reinforcement learning. In Proceedings of the 17th international conference on autonomous agents and multiagent systems (pp. 434–442). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems; AAMAS '18.

Åström, K. (1965). Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications, 10*(1), 174–205. https://doi.org/10.1016/0022-247X(65)90154-X

Talvitie, E. (2014). Model regularization for stable sample rollouts. In Proceedings of the thirtieth conference on uncertainty in artificial intelligence (pp. 780–789). Arlington, Virginia, USA: AUAI Press; UAI'14.

Tesauro, G. (1994). Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation, 6*(2), 215–219. https://doi.org/10.1162/neco.1994.6.2.215. DOI: 10.1162/neco.1994.6.2.215 arXiv:https://doi.org/10.1162/neco.1994.6.2.215.

Vella, V., & Lon, Ng W. (2016). Improving risk-adjusted performance in high frequency trading using interval type-2 fuzzy logic. *Expert Systems with Applications, 55*, 70–86. https://doi.org/10.1016/j.eswa.2016.01.056

Wu, J. L., Yu, L. C., & Chang, P. C. (2014). An intelligent stock trading system using comprehensive features. *Applied Soft Computing, 23*, 39–50. https://doi.org/10.1016/j.asoc.2014.06.010

Xiao, L., Dong, Y., & Dong, Y. (2018). An improved combination approach based on adaboost algorithm for wind speed time series forecasting. *Energy Conversion and Management, 160*. https://doi.org/10.1016/j.enconman.2018.01.038, 273–288.

Xing, F. Z., Cambria, E., Malandri, L., & Vercellis, C. (2019). Discovering bayesian market views for intelligent asset allocation. In *Machine learning and knowledge discovery in databases* (pp. 120–135). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-10997-4_8.