Journal of Data Intelligence, Vol. 3, No. 3 (2022) 333–349

CONTINUOUS EVOLUTIONARY DEEP REINFORCEMENT LEARNING ENVIRONMENT: AUGMENTED TRANSFER LEARNING-BASED GENETIC ALGORITHM

BADR HIRCHOUA
National Higher School of Arts and Crafts (ENSAM), Hassan II University (UH2), Casablanca, Morocco
badr.hirchoua@gmail.com

IMADEDDINE MOUNTASSER
National School of Applied Sciences (ENSA), Sidi Mohamed Ben Abdellah University (USMBA), Fez, Morocco
imountasser@gmail.com

BRAHIM OUHBI
National Higher School of Arts and Crafts (ENSAM), Moulay Ismail University (UMI), Meknes, Morocco
ouhbib@yahoo.co.uk

BOUCHRA FRIKH
National School of Applied Sciences (ENSA), Sidi Mohamed Ben Abdellah University (USMBA), Fez, Morocco
bfrikh@yahoo.com

Stock markets trading has risen as a critical challenge for artificial intelligence research. The way stock markets are moving and changing pushes researchers to find more sophisticated algorithms and strategies to anticipate the market movement and changes. From the artificial intelligence perspective, such environments require artificial agents to coordinate and transfer their best experience through different generations of agents. However, the existing agents are trained using hand-crafted expert features and expert capabilities. Notwithstanding these refinements, no previous single system has come near to dominating the trading environment. We address the algorithmic trading problem utilising an evolutive learning method. Precisely, we train a multi-agent reinforcement learning algorithm that uses only self trades generated by different generations of agents. The evolution-based genetic algorithm operates as an evolutive environment that continually adapts the agent's internal strategies and tactics. Also, it pushes the system forward to generate creative behaviours for the next generations. Additionally, a deep recurrent neural network drives the mutation mechanism through the attention that dynamically encodes the memory mutation size. The winner, which is the last agent, achieved promising performances and surpassed traditional and intelligent baselines.

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

1. Introduction

Stock markets elicit information from a continuous, unpredicted, and complicated real-world environment. Automated trading, also named agent trading, is one of the recent influential subfields in finance, which can be observed as the strategies that automatically execute sequential trading decisions. Generally, the goal is to execute profitable actions that maximize

the returns and reduce risk simultaneously. It is a developed method to submit many trading orders to an exchange [1]. Automated trading surpasses human traders on many levels, such as the stability of trades, less affection by emotional factors, and more creative and adaptative behaviour. Hence, it becomes an essential point in modern finance theory. The trading is undertaken in a continuous double auction between sellers and buyers in most financial markets. The trader, both human and algorithm, endeavours to sell at a high cost and buy low. Hence, the learned behaviour needs to satisfy some criteria, such as selecting when to go long and when to go short. Besides, as the market signals change almost randomly, the target prices and actions continuously adapt. The exchange information such as volume and prices are real-time shifting variables; thus, it shifts the targeted price.

If the agent can predict the stock market trend, it can draw powerful and profitable rules, including decisive actions and less risky positions. However, the ability to predict the market trend converges to zero, considering the stock market dynamicity and instability. The agent acts in an uncertain sequential decision-making environment. Therefore, modelling the price and return connection by estimating a probability distribution is impossible. Consequently, such an environment requires a rule-based policy rather than a price prediction model [2]. This policy inputs the price prediction and outputs the decision that maximizes the return.

Several proposals try to examine the automated trading problem. The robust approaches are based on Reinforcement Learning (RL) [1, 3]. The RL agents can learn precise and correct parameters by simply interacting with the environment. Furthermore, the agent can efficiently adapt and expect environmental changes. RL shows encouraging performances in the financial industry [1, 2, 3]. Deep reinforcement learning (DRL) tries to maximize returns and decrease the risk in financial trading simultaneously.

The DRL [4, 5] process, highlighted in figure 1, is formed as follows: at each time step, the agent inputs the current state and submits an action. The environment emits the corresponding reward and the next state based on the selected action. The agent selects the action to execute based on its policy. The policy's goal is to maximize the cumulative reward over time. The reward is essentially each trade's profit. In the current problem, the trader is the agent who selects when and how to execute trades in the environment. The environment details and features are continually shifting where the agent has limited control over most changes. Besides, the environment states are partially observable, where the agent can consult only a derivation of the actual state. Hence, sequential dynamic decision-making under uncertainty is more challenging. Therefore, the agent is forced to analyze unknown trajectories and execute accurate real-time decisions simultaneously.

The agent tries to learn the proper, optimal, and practical action. The market environment includes various agents, both algorithms and humans. The agent can execute three separate actions $A = \{Sell, \quad Buy, \quad Hold\}$. Each state S_t encodes the past prices of all transactions received until the time t. The reward R is each transaction's profit. Practically, the dilemma is to learn a policy that maximizes the accumulative reward.

The training process pushes the agent to explore the environment for synthesized experiences. The agent is obliged to select the profitable action based on imperfect information. Moreover, the causes and effects are not immediate, where early actions are not pay off for a long time. Therefore, the agent has to resonate about long-term planning, stabilize long- and short-term purposes, and adapt to unexpected circumstances. However, mastering the trading

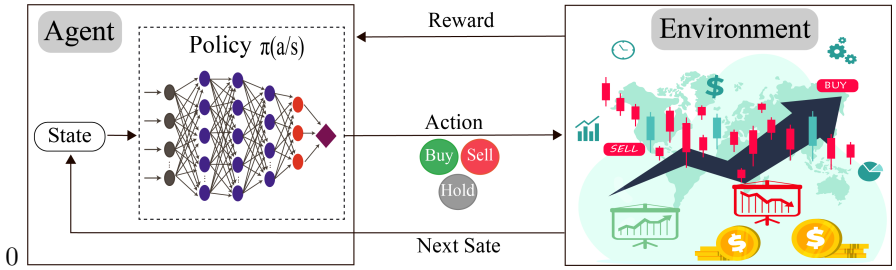

Fig. 1. The Deep Reinforcement Learning Framework

industry requires breakthroughs in diverse stages. First, the financial data is non-stationary, with low signals and nonlinear patterns. Next, the states' details are partially observable. However, decisive actions demand building inferences based on incomplete features. Some actions have a lesser impact on the environment, while others strategically affect the trading strategy. The action space contains three actions yielding to distinctive states' contexts; therefore, the agent must resonate about a long time horizons issue. Financial markets are complex dynamic environments holding multiple active agents. Hence, each action affects the observation series differently and continuously. Consequently, the system has to consider the high-dimensional continuous observation space. Finally, the reward sparsity is highlighted by the time lag connecting actions and their rewards.

Given these challenges, deep reinforcement learning-based trading has risen as a tremendous challenge for researchers. This work integrates the DQL and evolutive genetic algorithms to improve further agents' ability to explore these obstacles and generate a solid rule-based policy.

This paper is an extended version of our conference paper [6]. Precisely, we have extended the training data size, added two new critical events in the evaluation stage, and increase the number of generations to seven instead of only six. This paper combines new and existing general-purpose methods for artificial neural networks (ANN), deep reinforcement learning, transfer learning, and multi-agent learning to address the theoretic challenges and complexity of stock markets. The proposed system starts by creating an evolutive continuous training environment, where different agents collaborate, transfer knowledge, and trade against one another. The evolutive environment consists of a bench of generations containing different agents. Each version of the agent, both original or mutated, learns from several datasets to push the agents to overcome varying risk levels and encourage diversity in the evolutive environment. The created environment provides distinctive competitors to trade against and to develop agents' internal rewards and hyperparameters through the learning process.

In the first generation, the agents start entirely random with no experience. It scales the ladder of experience level until it masters the task. After terminating the learning phase, these agents are evaluated to select the stronger ones for mutation. The selection phase chooses the strongest agents in terms of internal reward. The agents' exhibit transfer learning in each generation, employing experiences learned under particular constraints to handle a never-before-seen one. The agent uses the deep Q-Learning (DQL) algorithm to update the policy parameters, reinforcing actions that happen exactly before positive reward. The final agent has collected an intense representation of the world states and has synthesized experience

good strategies. In the next generation, the selected agents are mutated. The mutation phase creates all possible combinations between the available agents. The proposed system mutates agents based on a dynamic experience size parametrized by a neural network that receives all agents' parents from the beginning as inputs and selects the mutation size as outputs.

The remainder of this paper is arranged into sections. In Section , the related works of this word are reviewed. In Section , the necessary preliminaries and the basic concepts are introduced. The partially observable Markov decision process, deep Q-Learning, and genetic algorithm are introduced. In Section , the contribution ideas and parts are explained. Section combines the results of the proposed approach, which is confirmed through comparative studies. Moreover, discussions are reviewed and covered in the same section. Section summarises this work and displays the proposals for future ones.

2. Related Work

Stock markets are dynamic environments containing many active agents. These agents are affected by each other behaviour and by outside information such as news. This generates a behaviour with substantial noise and randomness that is highly challenging to predict [7]. Hence, several algorithms and studies have concentrated on the financial industry [1, 2, 3]. On the one hand, we have proposed in our previous work [1] a new rule-based policy approach by which we train deep reinforcement learning agents for automated financial trading. Precisely, we have created a continuous virtual environment where we have trained different versions of agents. On the other hand, we have introduced a new DRL method based on an encouragement window [2]. The advantage function consisted of the discounted sum of rewards and the baseline estimate. Interestingly, our introduced encouragement window is based entirely on the past rewards fitting a dense synthesized knowledge rather than noisy and unclear signals. This proposal has extensively increased actions' quality by adjusting the action choice versus states' uncertainty.

Strassburg et al.[8] apply the genetic algorithm to optimize the technical trading rules. Zhang et al. [9] improve the recurrent reinforcement learning through a genetic algorithm to enhance the trading results. Their system uses the advantage of GA's capacity to select an optimal combination of fundamental, technical, and volatility indicators to develop out-of-sample performance. Youngmin et al.[10] develop a hybrid trading system for determining trading rules utilizing rough set analysis and a genetic algorithm (GA). In other words, they have used a novel rule discovery mechanism based on GA to solve optimization problems. Chang et al.[11] Combine GA and Markov decision process (MDP) to introduce a new analytical framework for trading. They have tried to make timing decisions by utilizing the prediction features and real-time analysis capabilities of the MDP. The parallel search capabilities of GA are employed to identify the profitable investment strategy.

Eilers et al. [12] introduce a decision support method that employs RL to enhance the profits of the direct seasonality plan. Cumming et al. [13] propose an RL algorithm for dynamic trading that estimates the value function of different states using the least-squares temporal difference. Deng et al. [14] develop a framework-based DRL to process the financial signal and trade online. They utilize bilevel optimization (BO) to structure the training procedure. Then, the online gradient descend (OGD) technique solves the optimization structure with fast convergence. Deng et al. [15] expand the theory of the policy search by employing DRL.

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

Paper	Dataset	Period	Technique
Eilers et al. [12]	DAX, S&P 500	2000-2012	ANN
Deng et al. [14]	China Stock Index Future (IF)	2013-2014	BO + OGD
Deng et al. [15]	SZSE	2014-2015	FDDR, DMLP+RL
Jiang et al. [16]	80 tradable cryptocurrency	-	CNN+RNN+LSTM
Di Persio et al. [17]	Google	-	RNN, LSTM, GRU
Lu et al. [18]	GBP/USD	2017	RL + LSTM + NES
Serrano et al. [19]	Derivative market/ Bond	-	RL, DMLP, GA
Chen et al. [20]	Taiwan TAIFEX	2017	RL agent + CNN
Li et al. [21]	APPL, IBM, PG, S&P500, ES, IF	2008-2018	DQN + A3C + LSTM
Azhikodan et al. [22]	NASDAQ-GE, NASDAQ-GOOGL	-	NN + RCNN
Jeong and Kim [23]	SP&500, KOSPI, EuroStoxx50, HSI	19872017	DQL + DNN
Lei et al. [3]	S&P 500	1999-2018	TFJ + DRL
Park et al. [24]	US-ETF, KOR-IDX	2010-2017	DQL
Hirchoua et al. [1]	SP&500, Facebook, Baba, Google, Gold, SFix, Apple, BTC	1960-2019	DRL - PPO
Hirchoua et al. [6]	Facebook, Baba, Google, Gold	2002-2020	DRL - DDQN
Ours	SP&500, Facebook, Baba, Google, Gold, SFix, Apple, BTC	1960-2019	DRL - DDQN

In addition to the RL methods, the authors consider the Fuzzy Deep Direct RL (FDDR) and Deep Multilayer Perceptron (DMLP).

Di Persio et al. [17] have done a comparative study over a primary Recurrent Neural Network, a Gated Recurrent Unit (GRU), and Long Short Term Memory (LSTM) to point out the most reliable variant. The results show that the Long Short Term Memory has surpassed the other variants by 72% accuracy. Jiang et al. [16] consider the problem of portfolio management. They have trained a neural network via online stochastic batch learning. This mechanism is proper for online and pre-trade training. Serrano et al. [19] introduce an asset banker algorithmic trading based on reinforcement learning. It generates the trading signal by employing a neural network alongside the genetic algorithm and RL. Chen et al. [20] consider an RL agent with a convolutional neural network (CNN) policy.

Jeong et al. [23] attempt to define the appropriate actions sequence alongside the number of shares for each particular action. Precisely, they merge a DQL with a policy form containing two components for learning the two requested outputs (actions and the number of shares). Azhikodan et al. [22] confirm that DRL-based methods are competent in recognizing the patterns of stock markets. They perform a deep deterministic policy gradient and a deep recurrent CNN for the opinion mining. Li et al. [21] develop a DRL trading agent to autonomously execute actions via an adjusted DQN and actor-critic (A3C) approach. Lei et al. [3] introduce a time-driven feature-aware integrated DRL model (TFJ-DRL) to enhance the action choice and the signal presentation. Lu et al. [18] use deep RNN, long-short term memory, and Natural Evolution Strategies (NES).

Recent works considering the same problem are highlighted in Table 1.

3. Preliminaries

This section presents the essential preliminaries about the partially observable Markov decision process and the double deep Q-learning network (DDQN) algorithm.

3.1. *Partially Observable Markov Decision Process*

A Partially Observable Markov Decision Process (POMDP) [25] presented in definition 1, is a generalization of a Markov Decision Process (MDP). The POMDP can model various dynamic real-world decision processes. It attempts to provide a policy that fulfils different tasks. A POMDP framework models environment with imperfect information. In other words, it models the agent decision process that does not observe the underlying state completely.

The MDP that includes discrete states with imperfect information needs to resonate against the observations, probabilities, and the underlying MDP to control the states' probability distribution. The POMDP model aims to maximize the system's reward while analyzing states' uncertainty and complexity. The sequential optimal actions produce the targeted optimal policy.

Definition 1 *A POMDP models the agent decision process and its interaction with the environment. Precisely, a POMDP is a 7-tuple* $< S, A, T, R, \Omega, O, \gamma >$*, where:*

- S *is the space of states,*
- A *is the space of actions,*
- T *is the conditional transition probabilities* $T(s|s,a)$ *for the state transition* $s \rightarrow s$*,*
- $R : S \times A \rightarrow \mathbb{R}$*: is the reward function,*
- Ω *is the space of observations,*
- O *is the conditional observation probabilities* $O(o|s,a)$*,*
- $\gamma \in$ *[0,1] is the discount factor.*

The space of state $S = \{s_0, s_1, s_2, \ldots\}$ defines the agent's world. Note that the transition among states confides only in the current state information. Precisely, the states are Markovian (Equation (2)), enabling the agent to focus only on the current state. This Markov property makes POMDPs tractable. In the context of the current problem, the action space involves three actions $A = \{Sell, \quad Buy, \quad Hold\}$. Every action hits two directions. On the one hand, it impacts the environment progression in terms of new states. On the other hand, the agent collects the next observation when considering a couple of states and actions. The a_t signifies the action executed in time move t.

After the agent executes an action, the environment emits the following observation sampled from the observation space $\Omega = \{o_0, o_1, o_2, \ldots\}$. Simultaneously, after executing an action a in a state s, the transition function highlights the probability of hitting the state s'. Precisely, this function $T(s'|s,a)$ holds the action's effects as a probability distribution.

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

where $Pr(s'|s,a)$ reflects the environment probability of hitting s' from s through executing the action a. Given this, the Markov property is represented as: $\forall s \in S, \forall a \in A$,

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

where t indicates the time in which the encoded state has been hit.

The observation function O highlights the probability of receiving the observation o upon hitting the state s' after executing the action a.

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

In order to learn the optimal strategy that maximizes the accumulated reward $R(s, a, s')$, the agent formalizes a reward function based on actions and states. More formally, the agent selects sequences of actions that maximize the expected future reward:

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

3.2. *Deep Q-Learning Network*

The Q-learning [26] seeks to learn a greedy deterministic policy. Precisely, given the current state, it tries to execute the optimal action. Q-learning attempts to construct a policy that maximizes the accumulated reward. It encodes a function to determine the state-action combination: $Q : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{R}$, More formally,

$$Q(s_t, a_t) = R(s_t, a_t) + \gamma \max_a(Q(s_{t+1}, a)) \tag{5}$$

The agent executes the action a_t at state s_tm and receives the reward $R(s_t, a_t)$. The action-value function represented by $Q(s_t, a_t)$ is the result of choosing the action a_t at the state s_t.

The resulted experience in terms of $\{s_t, a_t, R_t, s_{t+1}\}$ is saved in the reply buffer to train different agents. IN other words, the stored Q-values for all possible combinations of state s and action a. However, as the limitation of computation resources, we only store the last N experiences in practice.

The resulted experience in terms of $\{s_t, a_t, R_t, s_{t+1}\}$ is saved in the reply buffer to train different agents. In other words, the agents learn through the stored Q-values of different combinations of states and actions. Practically, the previous experiences are kept due to the limitation of resources.

Mnih et al. [27] scales the DRL to complex sequential decision-making problems. Deep Q-network (DQN) is an enhanced variant of the Q-learning, where a deep network estimates the Q-function $Q_\theta(s, a)$, and θ is the model's parameters. The objective the function of the DQN algorithm is:

$$\min_\theta \ J(\theta) = \min_\theta (y_t - Q_\theta(s_t, a_t))^2, \tag{6}$$

where y_t is calculated as the following:

$$y_t = \begin{cases} r_t, & if\ episode\ ends\ at\ t+1 \\ r_t + \gamma \max_a Q_\theta(s_{t+1}, a'), & otherwise \end{cases} \tag{7}$$

Different components $\{s_t, a_t, R_t, s_{t+1}\}$ are randomly sampled from the replay buffer. The second equation of the Equation 7 is formulated as the following:

$$y_t^{DQL} = r_{t+1} + \gamma Q_\theta(s_{t+1}, argmax_a Q_\theta(s_{t+1}, a)), \tag{8}$$

Generally, the DQL adopts the following Bellman equation:

$$Q(s,a) = Q(s,a) + \alpha\ [R(s,a) + \gamma maxQ(s',a') - Q(s,a)\ ], \tag{9}$$

where $Q(s,a)$ represents the Q value, γ is the discount rate, and α is the learning rate. Based on the new state s' and all possible actions from this state onward, the $maxQ(s',a')$ is the maximum expected future reward.

The DQL algorithm is a significant milestone in DRL, although various limitations influence its performance in the trading environment. Principally, extensive states space causes it intractable to learn and estimate the Q value.

3.3. *Genetic Algorithms*

Genetic algorithms (GAs) are a sub-field of evolutionary computation. They imitate natural selection alongside biological evolution principles to generate diverse solutions. GAs can optimize complex problems in an evolutive manner. Based on historical experiences, GAs enable the agents to explore the environment accurately, even if they start by a random configuration.

Generally, GA starts with a random population of agents. A predefined fitness function then evaluates agents in each population. Two agents are elected as parents based on the fitness function to reproduce one or more children. These new offspring agents are produced by utilizing a recombination driver, such as mutation and crossover. This process is iterated continuously until finding the best solution or reaching a predefined generations number. The essential pseudo-code of a GA is as follows:

1. Generate an opening random population of agents.
2. Estimate the fitness of the agents.
3. Generations = 0.
4. **While** convergence constraints are not met **do:**
   (a) Elect two agents as parents for reproduction.
   (b) Generate new offspring agents utilizing mutation and crossover drivers.
   (c) Evaluate the fitness of the newly produced agents.
   (d) Adopt the best new agents instead of weak ones.
   (e) Generations = Generations + 1
5. **End While**
6. **return** the Optimal Population.

In the next section, we introduce the proposed approach and explain the training process in detail.

4. Evolutive Deep Reinforcement Learning Based Trading Agents

To address the theoretic challenges and complexity of stock markets, we combine new and existing general-purpose methods for neural networks, deep reinforcement learning, transfer learning, and multi-agent learning. The proposed system consists of two main stages. We have created an evolutive continuous training environment, with diverse agents collaborating, transferring knowledge, and trading against one another, akin to the specialists' process. The evolutive environment consists of a bench of generations. Each generation contains a collection of different agents. Diverse datasets have dynamically been injected into different generations by branching from the existing ones; each version of the agent, both original or mutated, then learns from these new datasets. We have inserted several risky datasets to push the agents to overcome varying risk levels and encourage diversity in the created environment.

Figure 2 underlines the evolutive continuous environment training generations of evolutive agents collectively. It delivers distinctive enemies to trade against and develop agents' intrinsic rewards and hyperparameters through the learning process. Each circle denotes an agent in the generation, and the final one describes the strength agent's policy. The policy neural network points to taking action and receiving the reward associated with this action executed in the environment.

In the first generation, the agents initiate entirely random having no experience of the environment and trade. It scales the ladder of experience level until it masters the task. After terminating the learning phase, these agents are evaluated to select the stronger ones for mutation. The selection phase determines the most robust internal reward/profit agents. The agents display transfer learning in each generation, employing experiences learned under particular constraints to handle a never-before-seen one. The agents from the next generations are challenged with new risky datasets. Each agent controls its behavior, notwithstanding never observing the new data. The agent employs the DQL algorithm to update the policy parameters and reinforce actions that happen exactly before a positive reward. The final agent has composed an intense representation of the world states and has synthesized experience and good strategies.

In the next generation, the selected agents are mutated. The mutation phase creates all possible combinations of selected agents. It mutates two agents simultaneously, admitting that the mutation of more than two agents is presented in the constructed combinations. The proposed system mutates agents based on a dynamic experience size. In other words, the memory size mutation is described by a neural network that receives all agents' parents from the beginning as inputs and selects the mutation size as outputs. Besides, the proposed system allows the strongest agent from the previous generation to cross directly to the next generation. This behavior guarantees that the most potent agent can always be selected against weak ones, which gives the system the required flexibility to handle the decreasing or exceptional performances. In other words, the system keeps the most substantial DNA to anticipate the heavy changes in data and agents' behaviors. Each new generation selects 50% of the last percentage. Precisely, the first generation selects 50% of agents for mutation; the second determines 25%, the third 12.25%.

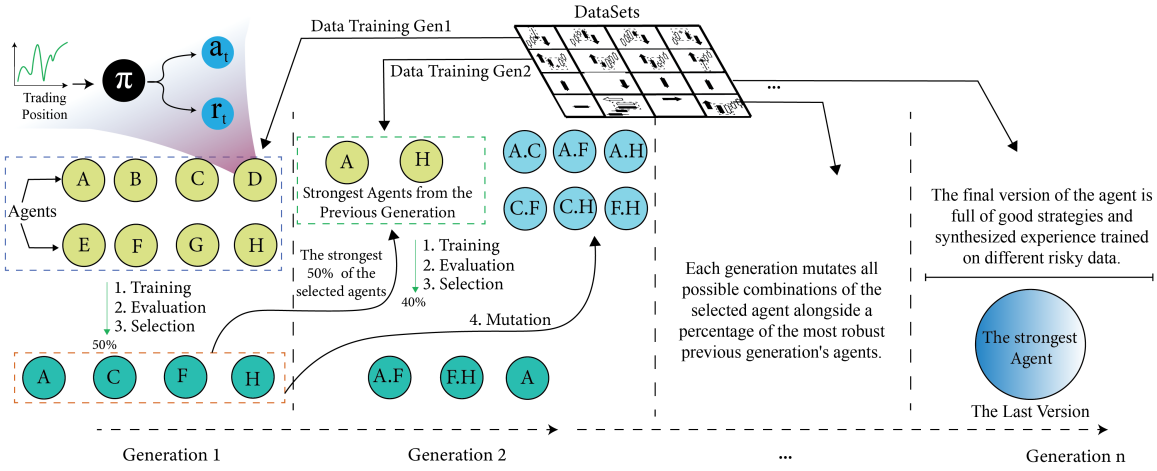

Fig. 2. The evolutive continuous training environment.

This evolution of training mechanism pushes the agents' training and policy enhancement purposes further, creating a method that continually and stably explores the immense environment states. Hence, it guarantees that each enhanced version through generations performs strongly against the environment uncertainty and crosses previous variants. As the evolution proceeds and new variants of agents are mutated, new counter-strategies appear and conquer the earlier ones. Some distinct agents deliver a strategy slightly an adjustment of a previous one; others discover drastically new policies. The unsafe tactics have been rejected as training proceeded, leading to stabler strategies.

Algorithm 1 Evolutive training process

1: First Generation: Generate a population of agents / traders using DQL;
   Population $= \{A_1, A_2, A_3 \ldots\}$
2: **repeat**
3: Train the population;
4: Select the half of the last percentage of agents following the maximum reward;
5: Identify the strongest agent A_s;
6: Generated the childs $\{C_1, C_2, C_3 \ldots\}$ by crossing/mutating their parents;
7: Update the population:
   Population $= \{A_s\} + \{C_1, C_2, C_3 \ldots\}$
8: Pass to the next generation;
9: **until** Convergence constrains

The proposed system mixes the power of reinforcement learning, deep learning, big data, and the advantage of genetic algorithms to find the optimal solution. Algorithm 1 highlights the evolutive training process. It starts by training a population of agents using a data set. This corresponds to the creation phase of the initial population. Next, the selection phase chooses the best agents. The higher the reward provided by an agent, the better the agent. Furthermore, the crossing phase merges two agents to produce only one. It forms a new agent

that has part of his parents' history and, therefore, part of all parents' history. For each generation, we repeat the process of selection and crossing using different datasets to push the training process further. The process ends if the generation size is reached or ends up with only one agent.

5. Results and Discussion

This section tests and proves the validity of the proposal. Furthermore, it demonstrates the proposal's advantages over a complete analysis.

5.1. *Performance Rating Measures*

The system performances are evaluated using different measures. The Sharpe ratio (S_p) [28] measures the excess return of a portfolio related to the risk-free rate divided by its standard deviation. It evaluates the performance of an investment considering the risk taken. The ratio corresponds to the average return above the risk-free rate, divided by the asset or portfolio of assets volatility. The following expression gives it:

$$S_p = \frac{E(R_{pi} - rfi)}{\sigma_p} \tag{10}$$

where R_{pi} is the return on the portfolio at the time i. The risk-free rate, corresponding to the return on a portfolio of investments with guaranteed capital, is denoted by rfi, and the standard deviation of the portfolio's return (thus referring to its volatility) is given by σ_p.

The Sharpe measure takes into account the total risk of returns. More recently, the Sortino ratio [29], which is a choice reward-risk evaluation measure, is proposed to consider the asymmetry in returns and not just the market risk. Given the return of the portfolio R_{pi} at the time i, the minimum acceptable return MAR, and the Downside risk (DR) $\sqrt{E(Min(R_{pi} - MAR), 0)^2)}$, the Sortino ratio of a portfolio is defined as follow:

$$SOR_p = \frac{E(R_{pi} - MAR)}{\sqrt{E(Min(R_{pi} - MAR), 0)^2)}} \tag{11}$$

The max drawdown (MDD) indicates the risk of a portfolio chosen according to a particular strategy. The MDD measures the most significant decline in the value of a portfolio. It corresponds to the maximum historical loss incurred by an investor who has bought at the highest and sold at the lowest during a given period.

From the perspective of the performance of hedge funds, it is more convenient to use the maximum possible loss from high to low as an alternative risk indicator. Given different non-normal distributions, the latter is positively/negatively skewed when the most extreme returns extend towards its right/left tail. The following expression is used to measure the degree of skewness, taking $\bar{r}$ as the mean: Considering different non-normal distributions, if there are more extreme returns spreading to the right/left tail of a distribution, it is positively/negatively skewed. The degree of skewness is measured using the following formula, where $\bar{r}$ is the mean:

$$Skewness = \sum \left(\frac{(R_{pi} - \bar{r})}{\sigma_p} \right)^3 \times \frac{1}{n} \tag{12}$$

Table 2. The parameter settings.

Parameter	value
Generations size	6
Generation 1 size	20
State Length	30
Number Of Episodes	25
Discount (γ)	0.99
Learning rate	0.001

Table 3. The strongest agent in each generation.

Performance Indicator	Gen 1	Gen 2	Gen3	Gen 4	Gen 5	Gen 6	Gen 7
Profit & Loss	21 274	24 458	24 033	29 286	106 021	168 611	175 846
Annualized Return	9.56%	10.47	10.98%	4.48%	3.01%	5.63%	25.36%
Annualized Volatility	22.75%	22.77%	12.37%	12.01%	15.33%	12.19%	38.52%
Sharpe Ratio	0.463	0.51	0.936	0.417	0.208	0.488	1.037
Sortino Ratio	0.683	0.796	1.11	0.557	0.272	0.56	1.663
Maximum Drawdown	22.67%	24.92%	10.10%	66.97%	20.78%	19.34%	13.68%
Max Drawdown Duration	121 Days	83	9 Days	237 Days	27 Days	65 Days	47 Days
Profitability	69.23%	100.00%	40%	41.18%	39.33%	52.94%	80%
Ratio Average Profit/Loss	1.123	inf	13.167	2.113	1.798	1.62	2.818
Skewness	0.645	0.882	- 0.567	-0.289	-0.371	-0.619	0.228

Automated trading has been attracting much attention. However, we need a substantial measure to evaluate the system improvement across the generation of agents. In the following subsection, we show and discuss the system performances.

5.2. *Experimental Results*

In order to demonstrate the proposed approach's applicability, we have used the closing prices of 8 stocks for evaluation. Contrary to our conference paper [6] where we have considered only 4 stocks, in this paper we consider 504 training dataset with 8 evaluation stock data as we have done in our previous work [1]. Figure 3 shows the used data. Also, Table 2 summarizes the environment parameters. On the one hand, the proposed system is evaluated in terms of Profit & Loss, Annualized Return, Annualized Volatility, Sharpe Ratio, Sortino Ratio, Maximum Drawdown, Profitability, Ratio Average Profit/Loss, and Skewness. Table 3 highlights the evolution of the strongest agents over generations. The last agent shows strong behavior, mastering every stock and generating a huge profit. We have followed the strongest agents through generations and observed their behavior. Interestingly, the mutation size neural network has progressively improved the mutation mechanism, balancing the mutation size and the amount of transferred knowledge.

We have used divers datasets to train each generation. Precisely, the facebook, Gold, SFix, Apple, SP&500, Baba, BTCUSDT, and Google datasets are used. The last agent is tested against the Facebook dataset to demonstrate its performance and the enforcement of the transferred learning mechanism. Table 4 illustrates the last agent's achievements. It outperforms all strongest agents from the past generations in all used metrics.

Table 5 compares the highlighted results form our paper [1]. It compares our agent with

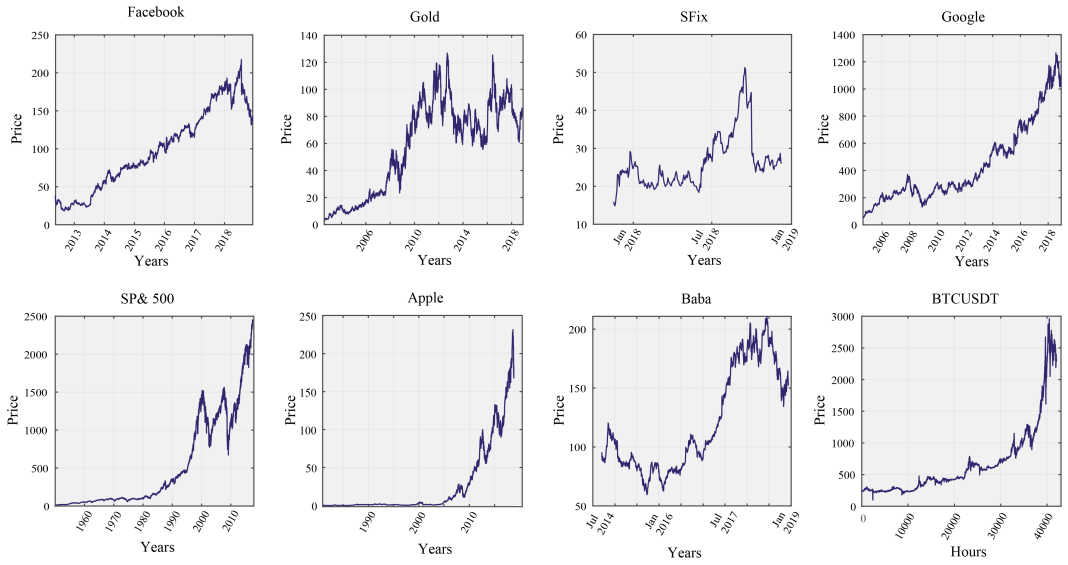

Fig. 3. Real prices of the 4 stocks.

Table 4. The final agent performances.

Performance Indicator	The Last Agent
Profit & Loss	191 951
Annualized Return	13.50%
Annualized Volatility	25.51%
Sharpe Ratio	0.783
Sortino Ratio	1.202
Maximum Drawdown	44.76%
Max Drawdown Duration	266 Days
Profitability	85.71 %
Ratio Average Profit/Loss	3.933
Skewness	0.357

Table 5. The last agent against four price prediction models, seasonality events detection [12], and deep recurrent Q-network [30], on S&P 500 stock.

System	Profit
Gradient Boosting Model	351.54
K-Nearest Neighbor	202.95
Support Vector Machines	210.22
Random Forest	225.15
Huang, Chien Yi [30]	257.19
Eilers et al. [12]	133.99
Ours	572.60

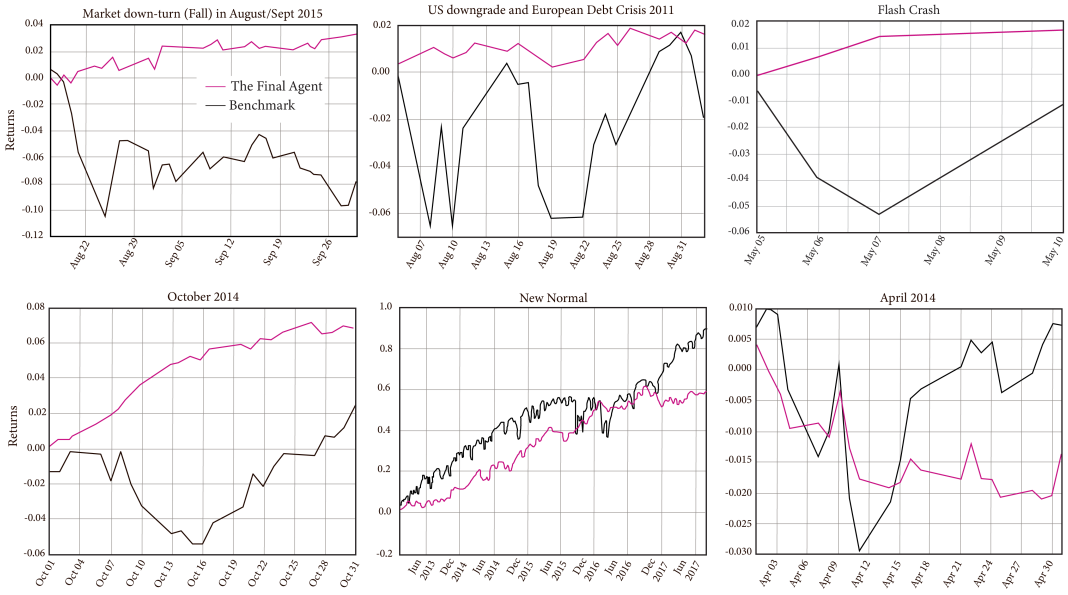

Fig. 4. Interesting Times performances.

Eilers et al. [12] and Huang [30] systems. Moreover, it describes how the price prediction models are limited in trading stock markets. Our agent makes the best profit compared to the other systems. The proposed system concentrates more on improving its behavior through transfer learning, rather than worrying more about seasoning events [12].

Compared to the methods mentioned above, the final agent generates more reliable and precise performances. Nevertheless, according to the earlier experiments, these systems maintain several shortcomings generating huge losses. The fundamental purpose of our system is to concentrate on the size of transferred learning. Practically, based on the neural network output, the mutation size is identified for every generation to balance the merged knowledge size through evolution.

The final agent has faced a list of historical six events that significantly have impacted markets. Precisely, we have considered the US downgrade and European Debt Crisis 2011, April and October 2014, Market down-turn (fall) in August/Sept 2015, new normal, and flash crash. Figure 4 shows the final agent performances. It produces a significant profit in

different events and fails with relatively small losses. Interestingly, The agent has learned from the April 14 event to master the October 14 fall event. Precisely, the model had beaten the October 14 fall by experiencing the April 14 fall, where the agent has shown relatively weak performances. In April 14 crisis, the benchmark has recovered for the first time on April 07 and for the second time on April 11. In the first recovery, the agent has stabilized its returns before the drawdown on April 09. The model stability highlights this behaviour compared to the start of the crisis. On the other hand, the market has started recovering on October 16, while the model has experienced a slight shift in its returns, becoming positively stable to avoid the April 2014 scenario. The agents that have encountered April 2014 have transferred their experience to the next episodes. The transferred knowledge has helped beat the October 2014 market crisis with higher gains.

6. Conclusion

In this article, we have addressed the algorithmic trading problem employing an evolutive-based generations learning method. The algorithm integrates existing and novel general-purpose techniques for deep reinforcement learning as a general mechanism driving agents and neural networks representing the agents' policies. Besides, the algorithm incorporates transfer and multi-agent learning to handle stock markets' theoretical issues and complexity. The evolution-based genetic algorithm has functioned as an evolutive bloc that constantly adjusted the agent's tactics and pushed the system forward to train new innovative behaviour for the following generations. Besides, a deep recurrent neural network has guided the mutation procedure via the attention mechanism that dynamically points out the memory mutation size. The final agent has achieved favourable performances and exceeded traditional and intelligent baselines.

In future research, the proposed model will be extended to use proximal policy optimization and trust region policy optimization instead of DQL. Moreover, we will inject a curiosity mechanism to add an extra index and improve the policy.

References

1. B. Hirchoua, B. Ouhbi, and B. Frikh, "Deep reinforcement learning based trading agents: Risk curiosity driven learning for financial rules-based policy," *Expert Systems with Applications*, vol. 170, p. 114553, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420311970
2. H. Badr, B. Ouhbi, and B. Frikh, "Rules based policy for stock trading: A new deep reinforcement learning method," in *2020 5th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech)*, 2020, pp. 1–6.
3. K. Lei, B. Zhang, Y. Li, M. Yang, and Y. Shen, "Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading," *Expert Systems with Applications*, vol. 140, p. 112872, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417419305822
4. T. Spooner, J. Fearnley, R. Savani, and A. Koukorinis, "Market making via reinforcement learning," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS 18. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2018, p. 434442.
5. P. Ganesh and P. Rakheja, "Deep reinforcement learning in high frequency trading," *arXiv preprint arXiv:1809.01506*, 2018.

6. B. Hirchoua, I. Mountasser, B. Ouhbi, and B. Frikh, *Evolutionary Deep Reinforcement Learning Environment: Transfer Learning-Based Genetic Algorithm.* New York, NY, USA: Association for Computing Machinery, 2021, p. 242249. [Online]. Available: https://doi.org/10.1145/3487664.3487698
7. J. Machado, R. Neves, and N. Horta, "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*, ser. GECCO Companion 15. New York, NY, USA: Association for Computing Machinery, 2015, p. 765766. [Online]. Available: https://doi.org/10.1145/2739482.2764885
8. J. Straßburg, C. Gonzàlez-Martel, and V. Alexandrov, "Parallel genetic algorithms for stock market trading rules," *Procedia Computer Science*, vol. 9, pp. 1306–1313, 2012.
9. J. Zhang and D. Maringer, "Using a genetic algorithm to improve recurrent reinforcement learning for equity trading," *Computational Economics*, vol. 47, no. 4, pp. 551–567, 2016.
10. Y. Kim, W. Ahn, K. J. Oh, and D. Enke, "An intelligent hybrid trading system for discovering trading rules for the futures market using rough sets and genetic algorithms," *Applied Soft Computing*, vol. 55, pp. 127–140, 2017.
11. Y.-H. Chang and M.-S. Lee, "Incorporating markov decision process on genetic algorithms to formulate trading strategies for stock markets," *Applied Soft Computing*, vol. 52, pp. 1143–1153, 2017.
12. D. Eilers, C. L. Dunis, H.-J. von Mettenheim, and M. H. Breitner, "Intelligent trading of seasonal effects: A decision support algorithm based on reinforcement learning," *Decision Support Systems*, vol. 64, pp. 100 – 108, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167923614001523
13. J. Cumming, D. Alrajeh, and L. Dickens, "An investigation into the use of reinforcement learning techniques within the algorithmic trading domain," *Imperial College London: London, UK*, 2015.
14. Y. Deng, Y. Kong, F. Bao, and Q. Dai, "Sparse coding-inspired optimal trading system for hft industry," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 2, pp. 467–475, 2015.
15. Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 3, pp. 653–664, 2016.
16. Z. Jiang, D. Xu, and J. Liang, "A deep reinforcement learning framework for the financial portfolio management problem," *arXiv preprint arXiv:1706.10059*, 2017.
17. L. Di Persio and O. Honchar, "Recurrent neural networks approach to the financial forecast of google assets," *International journal of Mathematics and Computers in simulation*, vol. 11, pp. 7–13, 2017.
18. D. W. Lu, "Agent inspired trading using recurrent reinforcement learning and lstm neural networks," *arXiv preprint arXiv:1707.07338*, 2017.
19. W. Serrano, "Fintech model: The random neural network with genetic algorithm," *Procedia Computer Science*, vol. 126, pp. 537 – 546, 2018, knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 22nd International Conference, KES-2018, Belgrade, Serbia. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S187705091831264X
20. C. T. Chen, A. Chen, and S. Huang, "Cloning strategies from trading records using agent-based reinforcement learning algorithm," in *2018 IEEE International Conference on Agents (ICA)*, July 2018, pp. 34–37.
21. Y. Li, W. Zheng, and Z. Zheng, "Deep robust reinforcement learning for practical algorithmic trading," *IEEE Access*, vol. 7, pp. 108 014–108 022, 2019.
22. A. R. Azhikodan, A. G. K. Bhat, and M. V. Jadhav, "Stock trading bot using deep reinforcement learning," in *Innovations in Computer Science and Engineering*, H. S. Saini, R. Sayal, A. Govardhan, and R. Buyya, Eds. Singapore: Springer Singapore, 2019, pp. 41–49.
23. G. Jeong and H. Y. Kim, "Improving financial trading decisions using deep q-learning: Predicting the number of shares, action strategies, and transfer learning," *Expert Systems with Applications*, vol. 117, pp. 125 – 138, 2019. [Online]. Available:

http://www.sciencedirect.com/science/article/pii/S0957417418306134
24. H. Park, M. K. Sim, and D. G. Choi, "An intelligent financial portfolio trading strategy using deep q-learning," *Expert Systems with Applications*, vol. 158, p. 113573, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417420303973
25. L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1, pp. 99 – 134, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S000437029800023X
26. C. J. C. H. Watkins, "Learning from delayed rewards," 1989.
27. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb 2015. [Online]. Available: https://doi.org/10.1038/nature14236
28. W. F. Sharpe, "Mutual fund performance," *The Journal of Business*, vol. 39, no. 1, pp. 119–138, 1966. [Online]. Available: http://www.jstor.org/stable/2351741
29. F. A. Sortino and L. N. Price, "Performance measurement in a downside risk framework," *The Journal of Investing*, vol. 3, no. 3, pp. 59–64, 1994. [Online]. Available: https://joi.pm-research.com/content/3/3/59
30. C. Y. Huang, "Financial trading as a game: A deep reinforcement learning approach," *arXiv preprint arXiv:1807.02787*, 2018.