Multi-DQN: an Ensemble of Deep Q-Learning Agents for Stock Market Forecasting

Salvatore Carta, Anselmo Ferreira, Alessandro Sebastian Podda, Diego Reforgiato Recupero*, Antonio Sanna

Dept. of Computer Science and Mathematics. Via Ospedale 72, Cagliari, Sardinia, Italy

Abstract

The stock market forecasting is one of the most challenging application of machine learning, as its historical data are naturally noisy and unstable. Most of the successful approaches act in a supervised manner, labeling training data as being of positive or negative moments of the market. However, training machine learning classifiers in such a way may suffer from over-fitting, since the market behavior depends on several external factors like other markets trends, political events, etc. In this paper, we aim at minimizing such problems by proposing an ensemble of reinforcement learning approaches which do not use annotations (*i.e.* market goes up or down) to learn, but rather learn how to maximize a return function over the training stage. In order to achieve this goal, we exploit a Q-learning agent trained several times with the same training data and investigate its ensemble behavior in important real-world stock markets. Experimental results in intraday trading indicate better performance than the conventional `Buy-and-Hold` strategy, which still behaves well in our setups. We also discuss qualitative and quantitative analyses of these results.

Keywords: Reinforcement learning, TD-learning, Q-learning, financial signal processing, neural networks for finance, trading.

1. Introduction

The ever-growing nature of machine learning research is becoming an oasis to be explored in more complex applications, such as stock trading. This task is known as a decision-making process, which helps traders to maximize some return-over-the-investment performance metrics, such as profit, economic utility,

*Corresponding author

Email addresses: `salvatore@unica.it` (Salvatore Carta), `anselmo.ferreira@unica.it` (Anselmo Ferreira), `sebastianpodda@unica.it` (Alessandro Sebastian Podda), `diego.reforgiato@unica.it` (Diego Reforgiato Recupero), `a.sanna106@studenti.unica.it` (Antonio Sanna)

risk-adjusted return, among others. The process of trading stocks is usually done in an online (dynamic) manner, and a successful action is composed of two steps: (i) knowledge of the market; and (ii) take the right decision. However, summarizing market conditions requires the investigation of time-series data from the past, which are usually noisy, unstable and contain lots of uncertainty. Additionally, changing decisions frequently in the stock market will be subject to transaction costs that minimize the profit. Finally, as identified by Zhang & Wu (2009), stock prices are often influenced by other factors such as political events, the behavior of other stock markets and even the psychology of the investors.

Despite these issues, the research in this area explored the development of artificial intelligence and machine learning through the years in order to deal with the uncertain behavior of stock markets and, as noticed by Dase & Pawar (2010), the literature and tools production in this field are growing since the end of the last century. Financial forecasting techniques are usually divided into two branches according to Henrique et al. (2019): (i) Technical Analysis (TA), which uses indicators calculated from the past to indicate future trends of the market; and (ii) Fundamental Analysis (FA), which seeks economic factors that influence market trends. Solutions in TA, such as the ones presented by Kamijo & Tanigawa (1990) and Kimoto et al. (1990), initially used artificial neural networks in order to predict the behavior (market prices go up or down) of the market in order to help traders to define the best online strategy. Other solutions used handcrafted features in Support Vector Machines as in Huang et al. (2005) and Kim (2003), Random Forests as in Manojlovi & Stajduhar (2015) and Ładyżyński et al. (2013), ensemble of the same classifiers as in Weng et al. (2018) and even fusion of different classifiers as in Asad (2015). Finally, the advances in machine learning such as the deep learning networks found a promising horizon in this research field. For example, Jagric et al. (2015) and Jagric et al. (2018) applied a novel optimized spiral spherical self-organized maps on the bank and insurance sectors, respectively. Other works from Chong et al. (2017), Gudelek et al. (2017), Hiransha et al. (2018), Barra et al. (2020) explored other network architectures and approaches. Although they are not the focus of this paper, FA and TA can be fused for a more robust and real-world setups.

Most of TA approaches consider the problem of stock market forecasting as a *classification problem*, where the classifiers use labeled training data from historical time series of prices in order to learn the market behavior. In the economic literature, such a problem is defined as a *sign prediction* problem, where classifiers must predict the behavior or prices, helping to decide if a *long* operation (*i.e.*, roughly, to bet that the price will rise in the future) or a *short* operation (*i.e.*, to bet that the price will fall in the future) should be done in the market in order to maximize the total profit in a given period. Other TA approaches are aimed at predicting the exact stock prices, which, in the economic literature, is termed as *predictive regression* as thoroughly discussed in Rapach et al. (2012).

However, in the sign prediction research, solving the stock market forecast-

ing problem with a simple and naïve classification approach only taking into consideration the past history and not also the chaotic, non-linear and non-parametric behavior of these data, has been shown to suffer from overfitting problems when put in real-world setups, a behavior identified in the works of Murphy (1999) and Tan et al. (2005). Although other TA solutions tried to minimize such a problem through predictive regression as shown by Rapach et al. (2012), Gudelek et al. (2017), and Hiransha et al. (2018), critics can be found to this strategy. For example, Si et al. (2017) discuss that it is difficult to consider transaction costs in predictive regression approaches and they also do not use previous decisions when forecasting.

A reasonable way of dealing with such issues is derived from a biological-inspired framework called *reinforcement learning* (RL), with one of the first relevant works in this field presented by Schultz et al. (1997). In this machine learning approach, the environment is viewed as a game that responds to an agent's action in a given state through rewards and new states. Then, the agent's objective is not to recognize patterns through labels, but to take actions in order to maximize rewards after some number of training iterations. In practical scenarios, the benefits of RL were already validated in a diverse set of applications, including video game playing as presented by Mnih et al. (2015) and Mnih et al. (2015), robots navigation as presented by Gu & Hu (2002) and Tsurumine et al. (2019), helicopter control as presented by Coates et al. (2017), among others. Although some works have validated successfully RL approaches to stock market prediction, as for example, in the works from Azhikodan et al. (2019), Shen et al. (2014) and Pendharkar & Cusatis (2018), the study on how to ensemble them is still an open field to be explored.

In order to address the aforementioned issue, in this paper we study the behavior of reinforcement learning ensembles on predicting the stock market trends. Our approach uses the same agent trained several times with the same training data, and we analyze its ensemble behavior in different markets. The advantages of ensemble methods were already shown in stock market forecasting in previous works of Weng et al. (2018) and Asad (2015), being thus considered the ideal strategy to real-world setups, as several approaches are applied to the problem at the same time, instead of just one. Additionally, joining different approaches will explore the complementary nature of the different classifiers used. This is particularly useful when we apply such ensembles on different markets but there are no guarantees that the best individual approach will be always the same in all possible markets. Moreover, when this strategy is coupled with a more effective feature selection technique and performing different combinations of actions in the market, we show that the proposed approach can outperform the Buy and Hold strategy when considering several return-over-the-investment metrics in several markets and trading periods.

The development of our approach is particularly motivated by the fact that stock markets have a natural chaotic behavior, thus they cannot be predicted by a simple trading strategy. Furthermore, such a behavior varies depending on the market considered and also the trading period. These issues make the stock trading task a very difficult task, especially considering that markets around

the world react with different intensities to periods of crisis. The aim of our approach is, therefore, to deal with such problems by proposing a flexible ensemble approach of reinforcement learning agents. Our approach is composed of multiple reinforcement trading agents that can deal with such uncertainty by trading with different trading agents with different experiences with the market and, therefore, being able to have complementary knowledge in predicting future data behavior. Additionally, we configure our trading strategy to perform diverse combination of actions in the market, with the goal to understand if the period and/or market analyzed are more suitable to some trading behaviors.

In summary, the contributions of this paper are:

1. An ensembling methodology for RL agents, which involves the use of different models trained at different iterations (epochs), with further analysis of the behavior of these ensembles through different agreement thresholds. We choose the Deep Q-learning algorithm in RL, initially presented by Mnih et al. (2015), applied to intraday stock market trading. We show that, even when coupled with a very basic network structure, the results of the ensemble has competitive performance against the `Buy-and-Hold` benchmark in some futures markets. To the best of our knowledge, this is the first work that explores mixing Deep Q-learning RL strategies in ensemble methods considering different training iterations to predict different future markets;

2. A solution that explores different combinations of decisions that can be suitable to specific markets and trading periods, and an analysis of the ensemble of such decisions;

3. A novel feature set, called *multi-resolution*, where different time resolutions of data are selected and joined in a more effective way, thus summarizing the relevant information that the agent must learn to achieve a better trading performance.

The remaining parts of this paper are organized as follows. Section 2 illustrates the background needed to understand RL and Q-Learning, also discussing how these techniques are used to perform stock market prediction. In Section 3 we discuss our proposed reward-based trading ensemble method, while Section 4 defines the metrics, datasets and methodology setups used for the experiments. In Section 5, we show and discuss our experimental results. Finally, Section 6 concludes this paper with final remarks and directions where we are headed.

2. Background and Related Work

In this section, we present the underlying theory and the related work to better guide the understanding of the problem and the solution we present in this paper. In particular, we introduce RL concepts along with their applications and related works in stock market trading.

2.1. Reinforcement Learning

Sutton & Barto (1998) present reinforcement learning as a self-taught process that can be represented by a Markov Decision Process (MDP). An MDP is a 4-tuple (S, A, P_a, R_a) where:

- S is a set of states;
- A is a set of actions (in the specific case of *reinforcement learning* this is the set of actions that an agent is allowed to do, thus in our case this set is finite);
- $P_a(s_t, s_{t+1})$ is the probability that an action a in state s_t will lead to the state s_{t+1}. Both states s_t and s_{t+1} are given by the environment, according to the action a done by the agent;
- $R_a(s_t, s_{t+1})$ is the immediate expected reward received in the transition from a state s_t to a state s_{t+1} doing an action a.

The main goal of the reinforcement learning process is, therefore, finding a policy π that returns the highest reward, or

$$\underset{\pi}{\operatorname{argmax}}\, \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_{\pi(s_t)}(s_t, s_{t+1})], \tag{1}$$

where $\pi(s_t)$ is the action a_t defined by a policy π in a given state s_t, and γ is a discount factor that motivates the policy to not postpone actions indefinitely.

To perform such a task, the agent can try different policies π in a training stage, but only one of those is the one with the highest reward possible, which is usually called the *optimal policy*. When applied to machine learning problems, the following analogies are usually made: (i) the state is a feature vector which is the input to the agent (*e.g.*, the 3D position of a robot); (ii) the actions are the outputs of the machine learning agent (*e.g.*, move right, left, backwards or forward); and (iii) the reward function is built at a training stage and higher rewards are related to correct actions (*e.g.*, at a given position, the robot is getting closer to the charging station). Thus, different from supervised and unsupervised learning, in RL the agent learns how to maximize a reward in an environment, instead of learning from annotations in supervised learning or learning how to cluster similar data as in unsupervised learning.

Depending on the learning objectives, typical RL algorithms are divided into critic-based and actor-based methods. Actor-based (*i.e.*, action learning) methods learn how to perform the policy from the data only, and continue to make the decisions from the policy. Therefore, the agent must learn a policy such that the action performed at each state is optimal to gain maximum reward in the future. Critic-based (*i.e.*, value function learning) methods, on the other hand, learn the policy by maximizing a *value function*, denoted as $V(s)$. This function is, in contrast to the short term reward $R(s)$, the expected long-term return with discounts, and $V_\pi(s)$ is denoted as the expected long-term return of the current state under policy π. Therefore, the goal of critic based methods is to find a policy through finding an optimal value function $V^*(s)$, which is greater than all the value functions of other possible policies, or

$$V^*(s) = \max_{\pi} V_\pi(s) \tag{2}$$

with

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}...], \tag{3}$$

where γ is defined as the discount factor, which weights the importance of future rewards versus immediate rewards. Some of the value functions more used are TD-Learning as shown by Tesauro (1995), SARSA as shown by Singh et al. (2000) and Q-learning as shown by Watkins (1989).

The Q-Learning algorithm, which is the RL approach of interest in this paper, is defined as a critic-based method that replaces the value function $V(s)$ with another function, called the *action-value* function $Q(x, a)$ which is disposed in a table. Such a function is calculated as:

$$Q(x, a) = \sum_y p_{xy}(a) \cdot [D(x, y, a) + \gamma \max_b Q(y, b)] \tag{4}$$

where x is the state given as input, y is any other state given by the environment according to agent's action, a and b are actions and p_{xy} is the probability of moving from state x to y given that action a was done. Finally, $D(x, y, a)$ is the immediate reward from taking action a and moving from state x to y and γ is the discount factor.

The update rule for finding the Q-function approximation is based on the following criterion:

$$Q(x, a)_{new} = Q(x, a) + \alpha[(D(x, y, a) + \gamma \max_b Q(y, b) - Q(x, a))], \tag{5}$$

where α is the learning rate, $Q(x, a)$ is the old Q-value at state x and $\max_b Q(y, b)$ is the maximum future reward considering all possible actions in the new state y. Such function is commonly known as the *Bellman Equation*.

After the Q-values are found and disposed in a table the agent will use a greedy strategy, choosing the optimal choice of action a^* in a testing stage as follows:

$$a^* = \underset{a}{\text{argmax}}(Q(x, a)). \tag{6}$$

In Q-learning, each $Q(x, a)$ is usually initialized to zero. Then, at each time t, the agent selects an action a_t, observes a reward r_t, enters a new state s_{t+1} that depends on the previous state x_t and the selected action a_t, and, finally, Q is updated as in Equation 5 and will guide the next action to be done in the algorithm. However, as the Q values are initialized to zero, the agent starts to explore the environment by taking random actions with high probability. This probability decreases with time, and then the agent starts to exploit the environment relying on the updated Q-values in the Q-Table. This procedure is commonly called ϵ-greedy strategy.

In the past years, neural networks were shown to overcome traditional machine learning methods in problems like computer vision, speech recognition,

time series prediction and many others, so they also showed promising applications in RL algorithms in doing a better search for higher rewards. In this case, the neural networks act as the agent, and will map actions on states to rewards. This branch of research in RL is commonly known as Deep Reinforcement Learning (DRL).

While neural networks normally learn coefficients (*i.e.* layer weights) in order to minimize a function that relates inputs to outputs (*i.e.*, the training error) in classification problems, in DRL they are used to recognize the state of the agent and learn weights in order to perform actions that return the highest rewards. For example, in a video game, deep neural networks like Convolutional Neural Networks (CNNs) can be used to recognize the state given as input by the environment, such as the current frame of the video game. Then, the CNN goal will be ranking the possible actions according to what the CNN learned before. For example, pressing button *right* returns -5 points, pressing *A* and *right* buttons returns 7 points, and so on. The benefits of using deep neural networks and the availability of ever increasing powerful hardware (*e.g.* NVidia GPUs) have expanded the use of DRL to more complex applications that showed to be problematic to previous RL algorithms. For example, updating a Q-Table becomes very inefficient when millions of different states are possible, such as in a 3-D video game.

In order to expand Q-learning to these difficult scenarios, Mnih et al. (2013) discussed the idea of Deep Q-Learning (DQL), which is replacing a Q-table (built as shown in Equation 4 and 5) with a neural network. Such network is used to learn weights in order to approximate the Q-function, no matter the input state. Once trained, the network receives a state as input and selects the action with highest Q-value. Q-values are any real values, which make the DQL a regression task.

The training of this network has, therefore, the goal to find weights w that predict Q-values as accurate as possible. Such a process works as follows: in the first iteration of the algorithm, all Q values for all actions are zero, and the agent starts sampling an action a randomly in an initial state x, receiving as feedback from the environment another state y. If y is not a terminal state, the algorithm calculates in Equation 4 discussed before what is called the *target function*, which is what the algorithm wants to approximate. Then, the weights update are done according to equation 7 as follows:

$$w_{t+1} = w_t - \alpha \bigtriangledown_w \mathbb{E}[(Q_{w_t}(x,a) - Q(x,a))^2], \tag{7}$$

where $\bigtriangledown_w \mathbb{E}[(Q_{w_t}(x,a) - Q(x,a))^2]$ is the gradient of the mean squared error between the q-value predicted by the network $Q_{w_t}(x,a)$ and the true $Q(x,a)$, which is calculated in Equation 4.

Several improvements have been proposed to the Deep Q-learning approach to improve the training, such as the Dueling Networks by Wang et al. (2016), Double DQNs by Hasselt (2010), Prioritized Experience Replay by Schaul et al. (2016), among others.

2.2. Reinforcement Trading

When applied to stock trading, some modifications should be done in order to apply RL to such an application. In Reinforcement Trading (RT), the price sequence is described as $p_1, ..., p_t, ...$ and the return at time t is defined as $r_t = p_t - p_{t-1}$. Trading decisions such as long, short and not trading in the market are RL actions and are represented as $A_t \in 1, 0, -1$. Therefore, according to Si et al. (2017), in RT, the profit R_t is calculated as :

$$R_t = A_{t-1} r_t - c|A_t - A_{t-1}| \tag{8}$$

where c is the transaction cost paid to the brokerage company when two different consecutive decisions are done, or $A_t \neq A_{t-1}$.

Hence, the objective of RT is maximizing the reward over a period T, or

$$\max_{\Theta} \sum_{t=1}^{T} R_t | \Theta, \tag{9}$$

where Θ is the family of parameters of the model. For example, if the model is a neural network, then $\Theta = \{w, b\}$, where w are the weights and b are the biases that should be trained to maximize the objective.

Another objective widely used in RT is the sharpe ratio. Moody & Saffell (1999) argued that, by considering such a metric, the model is aiming at maximizing risk-adjusted returns rather than maximizing profits. Such an objective can be defined as

$$\max_{\Theta} \frac{mean\{R_1, ..., R_t\}}{std\{R_1, ..., R_t\}} \Bigg| \Theta. \tag{10}$$

To maximize such objectives, RT agents are fed with feature vectors f_t, which consider the last m return values in the following manner:

$$f_t = [r_{t-m+1}, ..., r_t] \tag{11}$$

Several works in literature have explored concepts from reinforcement learning to stock trading applications. In one of the first approaches, Neuneier (1998) used a Q-Learning value-based RL approach to optimize asset allocation decision. Later, Mihatsch & Neuneier (1999) added in the Q-function the notion of risk. Gao et al. (2000) used as performance functions of Q-learning training the absolute profit and relative risk adjusted profit. Lee et al. (2007) presented four Q-learning agents that act serially in different steps of the trading procedure. Moody et al. (1998) proposed a recurrent neural network in the RL pipeline for trading, an approach known as Recurrent Reinforcement Learning (RRL). Deng et al. (2017) proposed to use deep neural networks with a novel back-propagation approach in RRL to commodity and stock markets forecasting. Finally, Si et al. (2017) expanded the use of deep networks and RRL to include a Long Short Term Memory (LSTM) layer and a multi-objective function to learn how to trade. For more details about the literature of reinforcement learning for trading, the reader can refer to Fischer (2018).

3. Proposed Approach

Although the use of RL agents has been extensively validated over the last years, a study on how to use them together for a more reliable trading system has not been extensively explored yet. In this paper, we propose to validate a multi-agent Deep Reinforcement Learning approach for stock trading, where agents are trained after successive and different iterations. We define these iterations as *epochs* from now on.

3.1. Trading strategy

We design our agents to perform a classic *intraday trading* strategy. This strategy consists of buying or selling a specific financial instrument (in our case, a stock *future*) by making sure that any open position is closed before the market closes in the same trading day. Specifically, we model our strategy such that an agent must perform *strictly one* of the following actions during a single trading day:

- a **long** action, which consists of buying the instrument, and then selling it before the market closes;
- a **short** action, which consists of selling the instrument (using the mechanism of the *uncovered sale*), and then buying it before the market closes;
- an **opt-out** action, which consists of deciding not to invest in that day.

The strategy goal requires to choose the action that maximizes the economic return (*i.e.*, the *reward*) of the day, given an assumption about the instrument price trend in that day (*i.e.*, whether the price will rise or fall). Thus, a *long* action is done whenever the agent assumes the price will rise in that day; conversely, a *short* action is done whenever the agent assumes the price will fall in that day; in the last case, an *opt-out* action is done whenever the agent is not enough confident about the market behavior, or it assumes the price will remain unchanged.

3.2. Summary of the approach

Given the trading strategy outlined in Section 3.1, we propose an approach composed of the following features:

1. **An ensemble of multiple agents trained at different epochs**. This results in performing intraday trading with agents that have different experiences about the strategy, arising from the inherent randomness of the training process, in order to maximize the reward in such a complex environment;
2. **A final decision, with different decision setups**. The final decision consists of an ensemble of decisions from different agents, and is parametric to different levels of agreement thresholds and agents configurations.

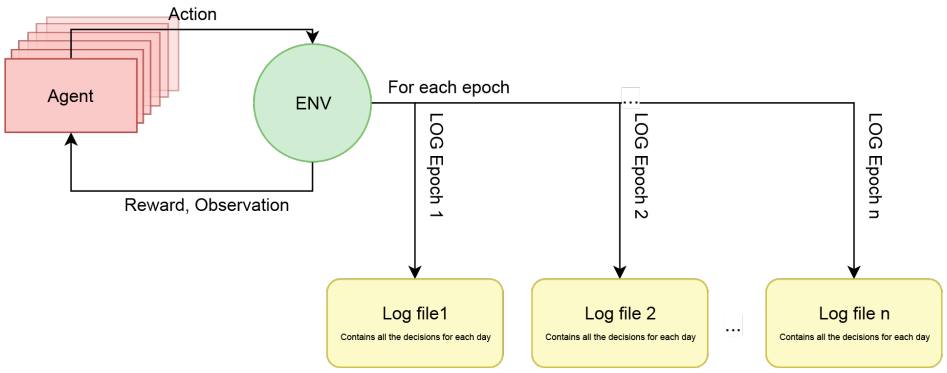


Figure 1: Proposed ensemble of deep reinforcement learning agents for stock trading. Multiple agents trained after multiple series of different iterations with the environment perform intraday stock trading, done by choosing between different combinations of actions. The final action to take is decided by an acceptable agreement of agents, which we call decision threshold in the scope of this paper.

For intraday trading, these decisions consist of performing *long*, *short* or *opt-out* actions. Our approach explores both fixed strategies (*i.e.*, one operation or opt-out) and mixed strategies (*i.e.*, using different combinations of longs, shorts and opt-outs).

3. **Use of *multi-resolution* samples**. This is a novel technique we propose in order to generate the input samples of our learning agents. This feature set consists of aggregating the market observations using different time resolutions and history depths.

The pipeline of the proposed approach is shown in Figure 1. Our proposed multi-agents are trained to maximize a reward function in the stock market environment, defined as follows:

$$Reward = \begin{cases} \dfrac{close - open}{open} & \text{if action is } long \\ -\dfrac{close - open}{open} & \text{if action is } short \\ 0 & \text{if action is } opt\text{-}out \end{cases} \tag{12}$$

with *open* being the opening price of the market in the considered trading day and *close* being the closing one.

After multiple different training iterations of the agent with the environment, the multi-agents perform different actions in the market. To keep trace of all the decisions taken by the agents, we store each decision history in a proper document, hereafter called *Log File*. Finally, given a wide combination of actions to take and a fixed agreement (or threshold) of decisions, our proposed approach performs the trading strategy. Further details of the proposed approach are provided in the following subsections.

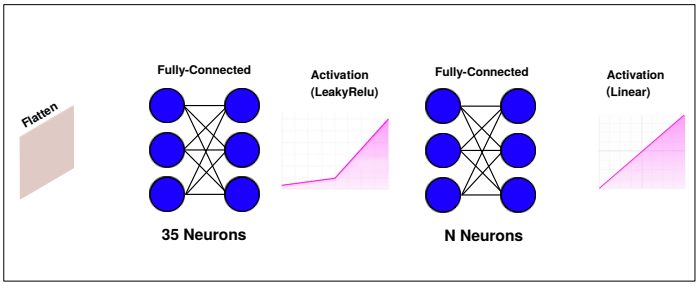


Figure 2: The neural network architecture of our proposed multi-DQN agents for stock trading.

3.3. Multiple DRL agents

Our approach is based on Double Q-Learning (DQL) agents from Hasselt (2010). In such an algorithm for DRL, Deep Q Networks (DQN) are used in order to learn the Q-table as previously described in Section 2. Such a parameterized table is learned through network weights, found through backpropagation of errors in a training stage. In a testing scenario, a DQN processes the input (state) and returns Q-values for each action to take. Then, the action with the highest Q-value will be chosen to be executed at that state. Therefore, for a state with n dimensions and an action space of m possible actions, the neural network is a function $R_n \rightarrow R_m$.

Some important features of DQL are the target networks, presented Hasselt (2010), and dueling networks, presented by Wang et al. (2016). The first one is used just to calculate the target from Equation 7. It is a network similar to the main network, and its weights w' are initially the same as the weights w of the main network. However, new weights are copied every t steps from the main network and remain the same in the other steps. The use of target networks is justified since the same parameters (weights) are used for estimating the target ($Q(x, a)$ in Equation 7) and the predicted Q-value ($Q_{w_t}(x, a)$ in Equation 7) in the classic Deep-Q learning. As a consequence, there is a strong correlation between the target and the network weights w. This means that, at each training step, both the predicted Q-values and the target change. This causes an unstable training procedure, as the weights update goal will be finding a predicted Q-value that gets close to an always moving target. The use of a target network yields a better and fast training, and is composed of two steps: (i) the main DQN network helps selecting the best action (the one with highest Q value); and (ii) use the target network to calculate the target Q value of taking that action at the next state.

Additionally, the DQNs of our approach consider the dueling networks from Wang et al. (2016). Such a concept helps the network to calculate each of the values ($D(x, y, a)$ and $\max_b Q(y, b)$) in Equation 4 separately and can also be used in target networks. This is done through individual fully connected layers positioned after the penultimate layers of the DQNs, which estimate each of these values with a final aggregation layer that is then used to calculate $Q(x, a)$.

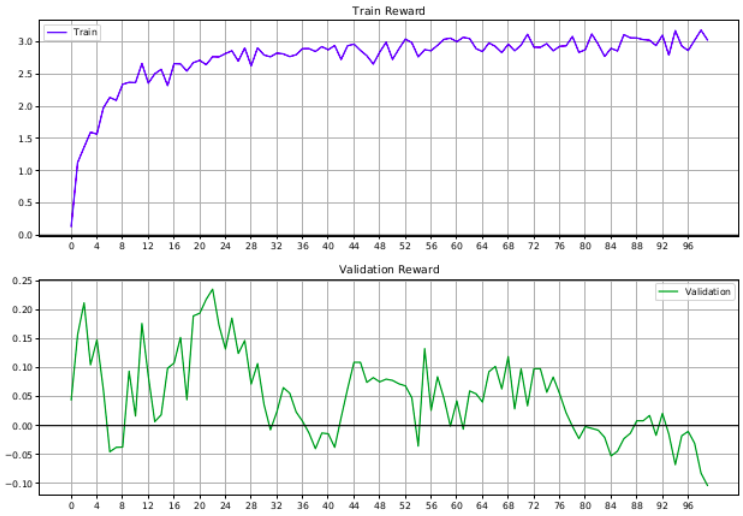


Figure 3: Different rewards achieved in multiple epochs (iterations) of training (top) and validation (bottom) steps of our multi-DQN agents in the stock trading scenario. Our proposed approach aims to use all of these agents, trained at different epochs, to perform stock trading in an ensembling setup.

To avoid backpropagation issues created by calculating Equation 4 in pieces, the aggregation layer includes subtracting the mean $\max_b Q(y, b)$ for all possible actions b. Therefore, this architecture helps accelerating the training, as the immediate reward $D(x, y, a)$ can be found without calculating the $\max_b Q(y, b)$ for each action at that state.

Figure 2 shows the architecture of the proposed main network used to stock trading, which includes the use of dueling and target networks in its final version. The main DQN is a very simple network composed of one flatten layer, one full connected layer with 35 neurons activated by the LeakyRelu from Maas et al. (2013), and N neurons fully connected layer with linear activation, where N is the number of decisions to take in the stock market.

Our approach considers several multiple DQN agents trained at different epochs (or iterations) in the environment. Each proposed ensemble consists then of different RL classifiers, one for each training epoch. Figure 3 shows a common behavior of these agents, represented by the training and validation rewards achieved in a run of the algorithm. The different behaviors of the agent when trained at different epochs is equivalent to consider them as multiple agents using the same strategy but with different experiences, which decisions are combined to design a final ensembling agent. In other words, the final agent works in an ensemble (or *late fusion*) fashion, which uses the agreement of all of its individual agents, at a given level, to perform its actions. The idea behind this approach is that these multiple agents can be complementary, as they have different experiences with the environment, making the approach more robust

against uncertainties of stock markets. Indeed, by ensembling the individual agents decisions through agreement thresholds, we require the final agent to perform a *long* or a *short* action only when such actions are decided by a given percentage of agents (*e.g.*, when there is an agreement of such decisions in at least $x\%$ of the individual agents). Otherwise, it opts to get out of the market. This is an intended behavior, since, for intraday trading, many observers suggest that entering the market not every day can lead, over the long term, to a more profitable strategy.

3.4. Multiple Combinations of Decisions

When applying DRL to predict stock prices, it is a good practice to have some previous knowledge about the market complexity in order to configure agents actions. To perform such a task, our ensemble of multiple DQN agents to stock trading can be configured with several combinations of actions that can be appropriate to different markets. We configured our approach with the following combinations of decisions:

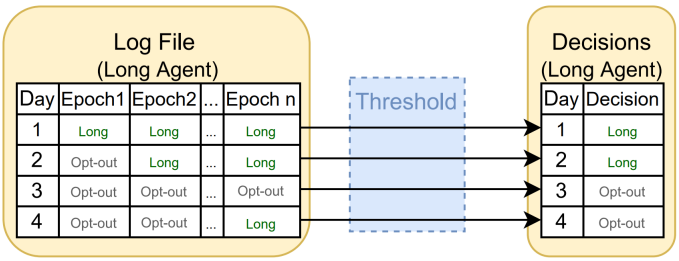
- `Only-Long` **agent**. This final agent is composed by multiple agents (trained at different epochs), all configured to perform intraday trading using *long* actions only, or getting out otherwise (the *opt-out* action is always allowed in our configurations) using training data in a training stage. To perform real-world trading, we first apply all these trained agents to predict an action to be done using as input the prices data from the past of a day we want to predict, generating a log file containing decisions for each new trading day according to each agent. The final agent decision is an ensemble of all the individual decisions, taken through an agreement threshold as illustrated in Figure 4(a).

- `Only-Short` **agent**. This final agent operates in a similar way to the `Only-Long` one, but its individual agents are configured to use just *short* actions or getting out of the market. In this case, decisions are also taken through the agreement threshold, as illustrated in Figure 4(b).

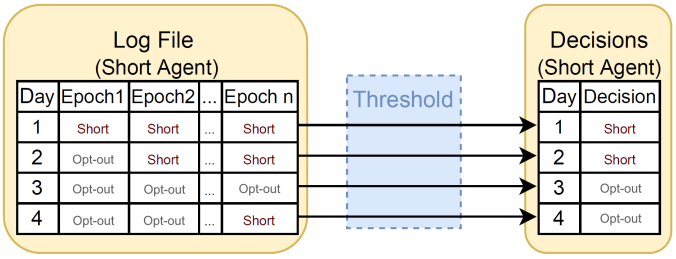
- `Long+Short` **agent**. This agent gets as input the results of the `Only-Long` and `Only-Short` agents seen above. Their final decisions are then combined as illustrated in Figure 4(c): the *opt-out* action is chosen when either there is no agreement between the two classes of agents or, in alternative, the two classes both recommend to getting out.

- `Full` **agent**. This final agent is composed by multiple individual agents that can perform both *long*s, *short*s and *opt-out*s, as illustrated in Figure 4(d). Its behavior is comparable to that of `Only-Long` and `Only-Short` agents, which uses the agreement threshold between agents trained at different epochs.

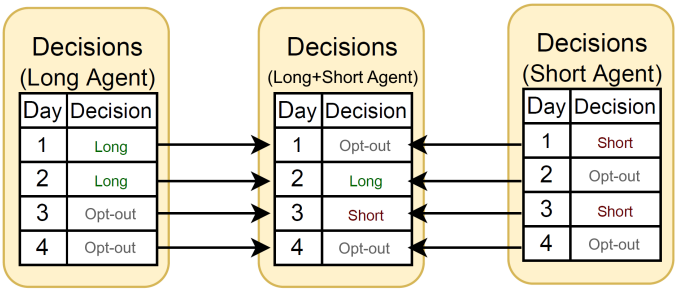
By taking one configuration as example, we can explain how our approach works in details. Given an *only long* configuration agent, we can train several RL



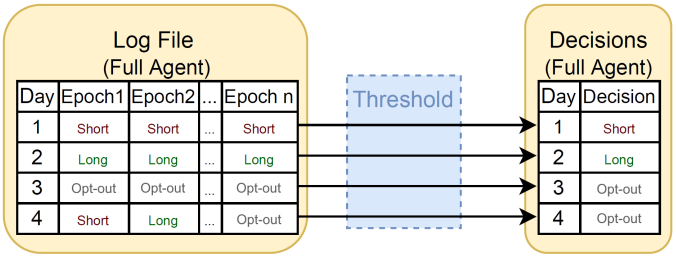
(a) `Only-Long` agent;



(b) `Only-Short` agent;



(c) `Long+Short` agent;



(d) `Full` agent.

Figure 4: Different configurations of decisions of the proposed multiple DQN agents for stock trading.

agents in order to learn to stay out of the market or performing long operations, depending on what returns the highest reward. These several agents are then applied to past data of trading days we want to predict, with several outputs from each agent. By choosing an appropriate agreement threshold, the agent can trade or not in the market according to frequency of the most frequent decision from the multiple RL agents, doing *opt-out* actions otherwise.

By configuring our agents this way, our scheme takes into the consideration the particularities of different markets. For example, the S&P500 tends to have a significant increasing trend in history, so *short* operations might not be so significant in terms of trading profits. Additionally, the agreement threshold can also be customized in order to make, for example, a more conservative or speculative trading behavior. For instance, in long term investments, we may opt to decrease the agreement threshold in order to avoid losing money in times of crises that might come. Such events may affect the data and, therefore, can also mislead most of the agents thus our approach can eliminate some of them in the final decision. On the other side, in short term investments, we expect that no crises are going to happen so we want to be sure that most of the agents are capturing the nature of prices behavior. Such features of our approach can be considered a step toward the design of *ad-hoc* automated stock traders to complex markets.

3.5. Multi-resolution samples

In this work, we propose a *multi-resolution* approach to provide the input samples to our learning agents. We start from the input dataset, which is a time series of market hourly observations. Then, for each observation time t of the market, we generate a vector $S^{(t)}$as the concatenation of the following terms:

$$S^{(t)} = H_l^{(t)} \parallel D_m^{(t)} \parallel W_n^{(t)} \tag{13}$$

where $H_l^{(t)}$ denotes the last l observations of the market at time t using the *hourly* resolution; $D_m^{(t)}$ denotes the last m observations of the market at time t after they are aggregated with a *daily* resolution; and, finally, $W_n^{(t)}$ denotes the last n observations of the market at time t after they are aggregated with a *weekly* resolution (note that l, m and n are parameters of the approach). Additionally, each observation in $H^{(t)}$, $D^{(t)}$ and $W^{(t)}$ is computed as the $(close - open)/open$ value for the considered time window $[(t-1); t]$ where, in this case, *open* being the observed price of the market at time $(t-1)$ (the opening price of that interval) and *close* being the observed price of the market at time t (the closing price of that interval).

4. Experimental Setup

In this section we detail all the procedures used in the experiments to validate our proposed approach in a stock trading scenario. We start discussing the considered markets, methodology, metrics and implementation aspects of the proposed approach.

4.1. Datasets

We choose to validate the proposed approach in two datasets widely used to test stock trading systems in the literature. The chosen markets are related to *futures*: the Standard & Poor's 500 ($S\&P500$), and the German stock index (DAX). These datasets were acquired from a brokerage company[1], and we used the hourly resolution datasets of prices initially, which structure can be found in Table 1. These markets price information are usually shown in terms of market points. The $S\&P500$ market has a point value of *50 USD*, and the DAX market has a point value of *25 EUR*.

From the original time series, we created new datasets of the remaining day and week resolutions that are also useful for our approach (Section 3.5) in the following manner: the *open* entry of the dataset is the first open of the day or week; the *close* entry is the last close of the day or week; the *max* is the maximum of the day or week and so on. With these datasets in hand, we can apply our multi-resolution approach described in Section 3.5, using $l = 40$ (last 40 hours), $m = 20$ (last 20 days) and $n = 8$ (last 8 weeks) as multi-resolution parameters. For our experiments, we have chosen $l = 40$ (last 40 hours), $m = 20$ (last 20 days) and $n = 8$ (last 8 weeks) as multi-resolution parameters. The choice of these resolution values lies in the idea of considering frequent market fluctuations for the hours immediately preceding the day to be predicted, while decreasing the granularity of observations as the time distance increases. In particular, these three values have been empirically determined, through preliminary experiments. Indeed, our exploration of the parameters showed that the choice of $l = 40 \pm 10$ allowed us to globally obtain better results, if compared to the choice of values outside this range; on the other hand, for the choice of m and n, we found that an increase of these values did not cause substantial variations in performance, where the choice of too low values penalized the results. Overall, this configuration leads to 68-dimensional vector samples.

Table 1: Datasets structure example.

Date	**Time**	**Open**	**High**	**Low**	**Close**
...	...	...	...	...	...
17/06/2011	19 : 00	7349	7363.5	7346	7351
17/06/2011	20 : 00	7352	7373	7350	7362
17/06/2011	21 : 00	7361	7376.5	7341.5	7374.5
...	...	...	...	...	...

We decided to perform in $S\&P500$ dataset two different experiments, where we analyse a short period of about two years (we call this sub-dataset $S\&P500$-S) and also a long period of about ten years (we call this sub-dataset $S\&P500$-L). This choice was made in order to investigate the learning agents behavior with

[1] https://www.tradestation.com/pricing/market-data-pricing/

more and less periods to train and test. We considered the same period of the $S\&P500$-L dataset for the DAX market. Details of such datasets are reported in Table 2.

Table 2: Futures market datasets considered in our experiments

Futures	**From**	**To**	**# Entries**	**# Long Entries**	**# Short Entries**	**Buy-and-Hold Strategy Return**
$SP500$-S	06/07/2015	06/05/2017	929	508	421	27,537.5 USD
$SP500$-L	17/09/2007	29/05/2017	3557	1946	1611	52,725 USD
DAX	18/09/2007	29/05/2017	2643	1409	1234	193,275 EUR

4.2. Experimental Methodology

We use a common approach in validating time-series data, which is called the *walk-forward* validation. In this experimental scenario, the semantic linking between the observation at time t and $t+1$ is taken into account to compose the same bunch of training, validation and test sets. This is different with respect to common cross-validation approaches like the leave-one-out cross validation or the k-fold cross validation, where data are randomly sampled in different folds, no matter when they were acquired. Such an approach is quite biased when applied to time series prediction, as features from late past and early future can be mixed in the same fold of data when using that strategy. The walk forward validation better fits this scenario, since the considered folds are temporally split and processed as training, validation and testing data. The number of walks selected for the experiments were 22 walks for $SP500$-L, 5 walks for $SP500$-S, and 21 for DAX (although the time periods of $SP500$-L and DAX are the same, there are missing days for DAX that causes the reduced number of walks). For each walk, we selected five years of data for training, six months for validating and six months for testing. For example, let us assume we have a dataset composed of 200 days that we want to divide into 6 walks of 100 days. One way is to consider the first 80 days of each walk as the training set and the remaining 20 days as the testing set, as shown in the left side of Table 3.

Table 3: Walk Forward Example

Data segment/walk	**Walk Forward**		
	Training	**Testing**	**Days**
1	$1 \rightarrow 80$	$81 \rightarrow 100$	100
2	$21 \rightarrow 100$	$101 \rightarrow 120$	100
3	$41 \rightarrow 120$	$121 \rightarrow 140$	100
4	$61 \rightarrow 140$	$141 \rightarrow 160$	100
5	$81 \rightarrow 160$	$161 \rightarrow 180$	100
6	$101 \rightarrow 180$	$181 \rightarrow 200$	100

4.3. Evaluation Metrics

In this subsection, we discuss how to measure the effectiveness of the proposed approach in real stock trading setups. For that, we choose some financial performance metrics, such as *Accuracy*, *Maximum Drawdown*, *Coverage*, *Sortino Ratio*, and *Equity Curve*. In the following we detail each of them.

4.3.1. Accuracy

Accuracy metric is commonly used in all classification problems since it provides an overview of the classification performance. It gives information about the number of instances correctly classified compared to the total number of them. More formally, given a set of X' trading days when the algorithm trades (*i.e.*, no *opt-out* operations were done), it is calculated as

$$Accuracy(X') = \frac{X'^{(+)}}{|X'|}, \tag{14}$$

where $|X'|$ stands for the number of trading days when decisions were made by the agent (*i.e.*, long or short operations were performed.), and $X'^{(+)}$ stands for the number of those correctly classified.

4.3.2. Maximum Drawdown (MDD)

Maximum drawdown estimates the downside risk in the time period taken into account, as stated in Magdon-Ismail & Atiya (2004). A low maximum drawdown is usually preferred because it shows that the risk of the investment is low. Formally, if P is the peak value before the largest drop and L is the lowest value before a new high value is established, its formal notation can be done as

$$MDD = \frac{(P - L)}{P}. \tag{15}$$

4.3.3. Coverage (COV)

Coverage metric reports the percentage of times the stock trading agent performs a long or short action (X') in the total number of days that the agent was working in the market (*i.e.*, $|X|$), as shown in Equation 16.

$$COV = \frac{(|X'|)}{|X|}. \tag{16}$$

Such a metric shows interesting insights about the algorithm, as the latter only takes a decision when an agreement of its multiple agents is reached at a pre-determined level, otherwise it gets out (*i.e.*, does not trade anything).

4.3.4. Sortino Ratio

Sortino Ratio (SR) is considered one of the best metrics to statistically assess a trading strategy performance, as it offers a view of its expected return in relation to the risk assumed. Its formula is described by following Equation 17:

$$Sortino\_Ratio = \frac{\bar{R} - r_f}{\sigma_d}, \tag{17}$$

where $\bar{R}$ is the expected average return of the strategy (aggregated on a daily, weekly, monthly or yearly basis), while r_f is the *risk-free rate* or *target rate*, *i.e.,* the minimum acceptable return (in common practice, it is assumed to be equal to the average return generated by a zero or negligible risk investment plan). Finally, σ_d is the *downside deviation* of the returns, *i.e.,* the standard deviation of negative returns only, which estimates the volatility (or risk) of the considered strategy.

The Sortino Ratio slightly differs from the alternative *Sharpe Ratio* metric, which instead requires to compute the standard deviation on all returns, and not just the negative ones. The latter, therefore, penalizes both the upside and downside volatility; for these reasons, Sortino Ratio is sometimes preferred on theoretical grounds as observed in (Chaudhry & Johnson (2008)).

We finally observe that, in our experiments, the Sortino Ratio is provided both on a daily and monthly basis, and that, as risk-free rate, we consider the expected return generated by the average rate of the *U.S. 10 Years Treasury Yields* in the considered period and for the same investment amount, when adopting a compound financial regime.

4.3.5. The Equity Curve

According to Schipper & Smith (1986), the Equity Curve, also known as *Profit and Loss Curve*, reports how much, in terms of money, a stock trading agent is earning or losing in the stock market in a graphically-represented time period. Positive slopes in such a graph highlight the effectiveness of the trading strategy, while negative slopes indicate that such a strategy generates negative returns. For instance, given an *Initial Investment II* to trade a *number of futures* that have a certain *entry price* and *exit price*, and also considering the related *trade commission*, we can calculate the points $EC = \{ec_1, ec_2, \cdots, ec_N\}$ that we need to plot in the *P&L* curve as shown in Equation 18.

$$\begin{aligned}
&ec_1 = II - ((entry\ price\ \times\ number\ of\ futures) - commission)\\
&ec_2 = II - ((exit\ price \times\ number\ of\ futures) - commission)\\
&\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad .\\
&ec_{N-1} = II - ((entry\ price\ \times\ number\ of\ futures) - commission)\\
&ec_N = II - ((exit\ price \times\ number\ of\ futures) - commission)
\end{aligned} \tag{18}$$

Finally, we also consider the total amount acquired in the whole stock trading process, also called *Return*. This metric sums up each point in the Equity Curve, informing us how much money was earned or lost in the full process.

4.3.6. Statistical Tests

To test the statistical relevance of the obtained experimental results, we exploit the *Student's t-test* from Gosset (1908). This test can determine if there is a significant difference between the means of two distributions. The t-test looks at several metrics, such as the *t-statistic*, the *t-distribution* and the degrees of freedom, to determine the statistical significance.

To apply the t-test in our scenario, we bootstrap several out-of-sample trading days, for several times. Then, we calculate distributions of Sortino Ratios to be used in the test, to point out statistical differences (if any) pairwise. In all tests, we set the confidence level to 95%. It means that, if the calculated p-value is below 0.05, then the null hypothesis, which says that there is not statistical significance between the two distributions, is rejected.

4.4. Technical Details

The approach proposed in this paper has been developed in *Python* using KERAS-RL library Plappert (2016) and was run in an *Intel i7-8700k* with a *64-bit* Operating System (*Linux Ubuntu*) with *64 GBytes* of *RAM* and a Nvidia TITAN XP GPU. To implement our approach, we apply our reward in Equation 12 in function step() of the KERAS-RL environment file. At each training sample, Equation 12 calculates the reward, which is then attached to Equation 4 to calculate a target q-function, and network weights are updated in order to approximate such a function as in Equation 7.

As discussed previously in Section 3.3, our multi-agent includes several DQN agents, equipped with target network and dueling architecture. The networks were trained through the ADAM optimization procedure by Kingma & Ba (2014), with a learning rate equals to 10^{-3}. We set the probability of explorations (the amount of random actions taken by the algorithm) as 20%. Additionally, our ensemble of multiple agents considers agents trained at 100 epochs. Finally, we tested different agreements to obtain a decision: from 60% to 100%, with a step size of 10%.

4.5. Baseline strategy

We use the common `Buy-and-Hold` (we sometimes refer to it as `BH`) strategy to compete against our approach. `BH` is a passive investment strategy where an investor buys stocks and holds them for a long time, with the goal that stocks will gradually increase in value over a long period of time. Such a strategy is largely used in literature as a baseline to evaluate the profitability of other investment strategies and, as such, is considered a valuable benchmark for our experimental results.

5. Results and Discussion

We now start discussing the results of the experiments we carried out to validate the behavior of our learning agents in some real-world trading scenarios.

We used the walk-forward methodology discussed in previous Section 4.2. To define the final ensembled agent to be adopted in the unknown testing periods, we use the same approach of Deng et al. (2017), where multiple runs are performed (we choose *five*) and the agent with the best validation accuracy is selected (Equation 14), in order to discard overfitting networks. This section reports the experiments done in $S\&P500$-S, $S\&P500$-L and DAX datasets respectively. For each experiment/market, we fixed the initial investments regardless of the trading strategy. Their values are:

- SP500-S: $102,087.50$ *USD*
- SP500-L: $74,675.00$ *USD*
- DAX: $201,762.50$ *EUR*

All trading strategies also adopt a stop-loss mechanism, set to $(open_d \pm 0.05 \cdot open_d)$, where *open* is the opening price of each trading day d (*i.e.,* a 5% offset depending on the taken daily decision, long or short). Notably, the source code of our solution is publicly accessible on our Github repository[2].

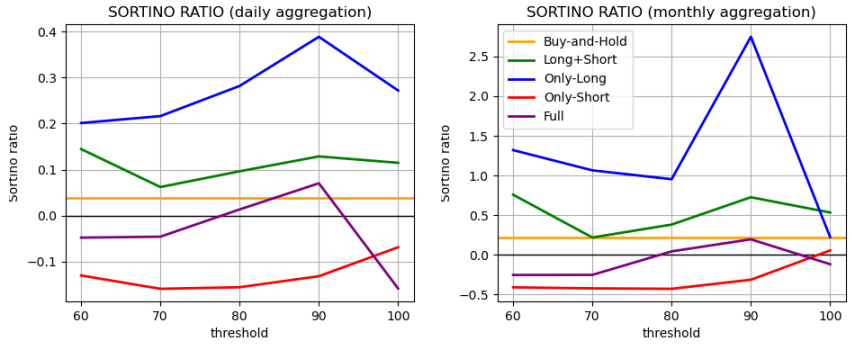
For each scenario, we first analyse the global profit-risk performance of each strategy against the `Buy-and-Hold` baseline both in terms of daily and monthly aggregated Sortino Ratio and different levels of agreement. Then, we report and discuss other four relevant metrics for evaluating each strategy: Return, Maximum Drawdown, Coverage and Equity curve of the best agreement threshold for each method. Notably, as described in previous Section 4.3.4, to compute the Sortino Ratio, we assume as the *risk-free rate* the average rate of U.S. 10 Years Treasury Yields in the considered trading period. We discuss the aforementioned results in the following subsections.

5.1. Results on the S&P500-S dataset (short period)

We start to validate our approach by considering a small fraction of the $S\&P500$ dataset, from 2015 to 2017, which is a relatively recent period. Figure 5 summarizes the results of the experiment.

Figure 5(a) shows that, in terms of Sortino Ratio, our `Only-Long` agent considerably outperforms the `Buy-and-Hold` baseline, which, in overall, has a positive trend in the considered scenario. Results in Figure 5(b) reinforce such an observation. Specifically, this agent provides better results (both on a daily and monthly basis) when the selected agreement threshold does not exceed 90%. However, we note that this agent is still slightly better than the baseline also with a 100% threshold, albeit with a substantial degradation in performance. We also observe that, in this scenario, our `Long+Short` agent represents the second best approach, showing to be a strategy that still overcomes the market behavior, but introducing short operations may result in a more risky solution, as the short term period analyzed shows an upward behavior of stock prices.

[2] `https://github.com/multidqn/deep-q-trading`



(a) Daily aggregated SR; (b) Monthly aggregated SR.

Figure 5: Results of the S&P500-S experiment in terms of (a) *Sortino Ratio* daily aggregated; and (b) *Sortino Ratio* monthly aggregated. The best performance is given by our `Only-Long` agent, which performs better than the `Buy-and-Hold` baseline for each considered agreement thresholds, both in daily and monthly basis.

Conversely, the proposed `Full` and `Only-Short` agents are not competitive in this scenario.

Figure 6 summarizes the other metrics we consider for the $S\&P500$-S experiment. In Figure 6(a), it can be observed that our `Only-Long` agent, when considering agreement thresholds in the range $60 - 90\%$, defeats `Buy-and-Hold` and has the global best performance in terms of return over the investment for this short term scenario.

Figure 6(b) confirms the above observations in terms of drawdown since, according to their MDD results, both `Only-Long` and `Long+Short` agents are significantly less risky than the `Buy-and-Hold` strategy. Following our hypothesis, the `Full` and `Only-Short` agents showed higher MDD than our best approaches in most of the agreement thresholds considered. Figure 6(c) gives a hint in such previous result as it shows the coverage of each agreement threshold. The `Only-Long` agent with 60% threshold is performing decisions in about 45% of the market days, whereas the `Long+Short` agent is deciding in about 50% of the market days. Comparing 6(b) and 6(c), we may also note that, although it seems intuitive that the risk (MDD) should increase when more actions are taken on the market, our agents (with the exception of the `Only-Short` agent) are sufficiently robust against that issue.

The trading improvements given by the `Only-Long` and `Long+Short` agents of the proposed approach can be seen in Figure 6(d), where their gains over time (equity curves) show a mostly positive trend, where the `Buy-and-Hold` approach suffers from a quite negative initial period. However, the curve from `Only-Long` strategy showed to be more stable in terms of behavior over time, and with an overall profit consistently higher than the others. From a reinforcement learning point of view, it is interesting to analyse the different behavior between the `Long+Short` and the `Full-Agent`. Indeed, the first one seems to

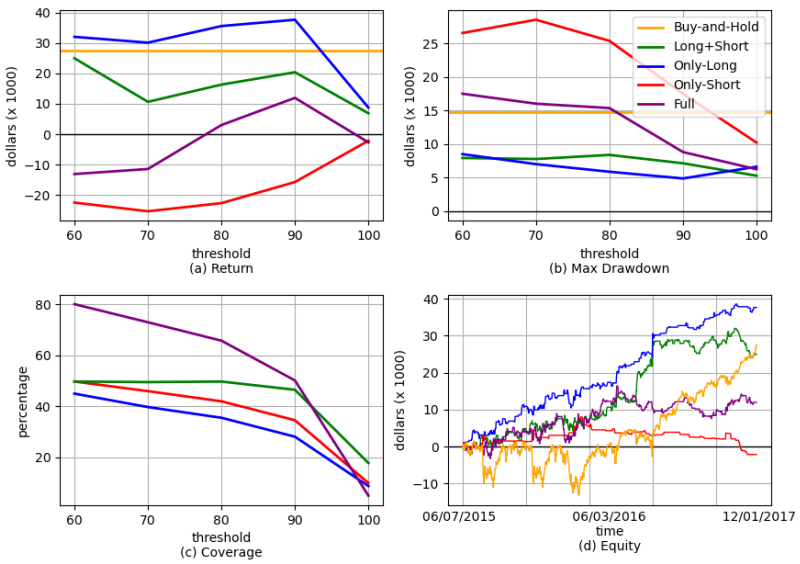


Figure 6: Results of the S&P500-S experiment in terms of Return (higher is better), Maximum Drawdown (lower is better), Coverage and Equity (higher is better).

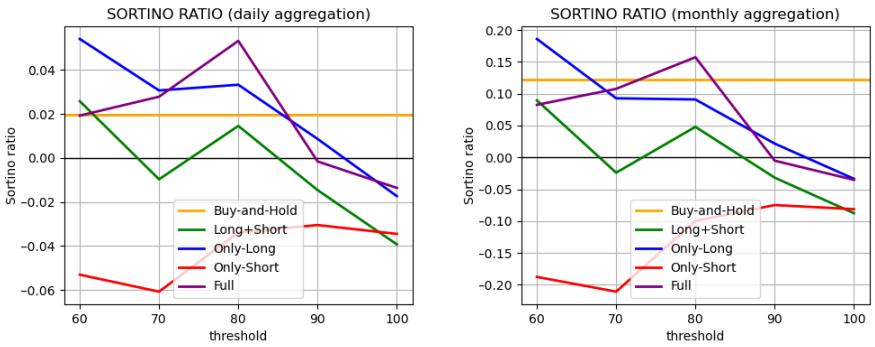
react better to the market recovery showed in the second half of the period, in which its `Only-Short` sub-agent (with the exception of the last few weeks) has a more prudent profile (*i.e.*, it chooses to opt-out in the majority of days), thus favouring the decisions of its `Only-Long` sub-agent. On the contrary, the `Full` agent, by the actual way it is designed, seems to be more aggressive in deciding to go long or short, in both phases, which – in this specific scenario – translates into a very good response to the first negative period, but in an essential inability to take advantage of the subsequent strong market recovery. This behaviour can be explained since, for the `Full` agent, performing long or short is potentially more profitable than opt-out (indeed, Figure 6 shows that this strategy has high coverage and risk for most thresholds). Conversely, in the `Only-Long` and `Only-Short` approaches, the agents learn that performing opt-out is better than doing a wrong long (or short) operations, as they are not enabled to perform the opposite operation. This is why the `Long+Short` combination is more conservative than the `Full` agent (it enters the market less times), but more sensitive to very sharp market trends.

Finally, Figures 6(b) and 6(c) also show that, in general, the choice of a higher agreement thresholds implies in a decrease of coverage which, contrary to what one might think, involves not only a decrease in risk, but also (considering up to a 90% threshold) an increase or retention of the expected return for all the considered strategies. Hence, from these results, we can conclude that our

`Only-Long` agent, with 90% agreement, provides the global best result in this trading scenario in terms of all the considered metrics.

5.2. Results on the S&P500-L dataset (long period)

We now consider a longer period of the $S\&P500$ dataset, which spans the years from 2007 to 2017. Figure 7 shows the experiment trading results in terms of Sortino Ratio.



(a) Daily aggregated SR; (b) Monthly aggregated SR.

Figure 7: Results of the S&P500-L experiment in terms of (a) daily aggregated *Sortino Ratio*; and (b) monthly aggregated *Sortino Ratio*. The `Only-Long` and `Full` agents perform better than the `Buy-and-Hold` baseline in both metrics, with, respectively, the agreement thresholds of 60 and 80%.

From Figure 7, we notice better performances from our `Only-Long` and `Full` agents, considering, respectively, agreement thresholds of 60 and 80%, if compared to the `Buy-and-Hold` baseline. Therefore, we find a point of novelty in the behavior of `Full` agent that, differently from the $S\&P500$-S scenario, provides here good results with lower thresholds (up to 80%). This fact can be justified by observing that: (i) a longer training period has allowed the agent to better learn how to balance the use of opt-out operations, whereas over short periods of trading, the use of a more aggressive strategy based on long and short actions can lead to potentially higher rewards; (ii) this strategy seems to respond better to negative market periods (as already emerged from the previous $S\&P500$-S experiment), and therefore takes advantage of the effects of the big economic crisis of 2008, as well as the subsequent market falls (in particular those of induced by the one-trillion dollar flash crash in 2010 and the 2012 credit crisis in Europe). Conversely, the `Long+Short` strategy proves to be weak in this broader scenario. This behaviour is mainly due to the very negative performance of the `Only-Short` agent which, despite the above mentioned crisis periods, and as described in the previous Section 5.1, tends to adopt a more conservative strategy in which it favours opt-out decisions rather than taking the risk of (potentially) making wrong short actions, which can be strongly penalized in terms of reward.

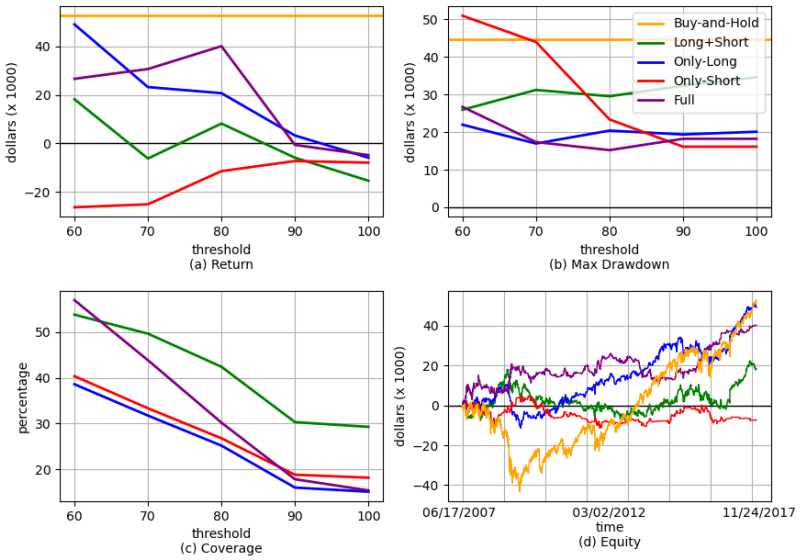


Figure 8: Results of the S&P500-L experiment in terms of Return, Maximum Drawdown, Coverage and Equity.

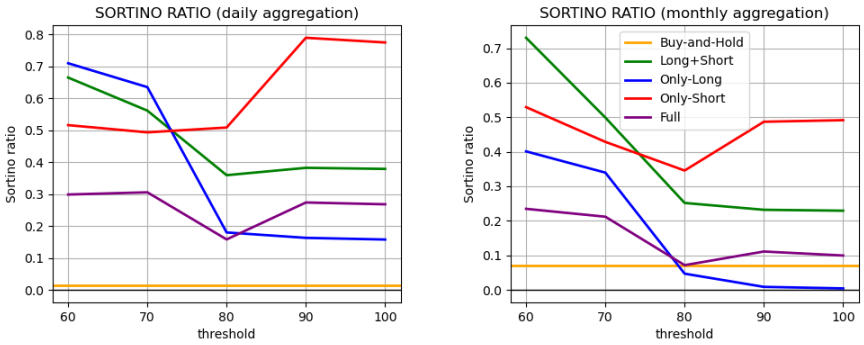
Figure 8 shows other performance metrics for that dataset. In Figure 8(a), we can observe that `Buy-and-Hold` has the better final Return in this trading scenario (which, by the way, shows a profit that is about 5,000 USD more than our best approach). However, going deeper in Figure 8(b) results, we can notice that, by quantifying the risk of investing through the Maximum Drawdown metric, most of our agents, with the only exception of the `Only-Short` agent at 60% agreement, perform significantly less risky operations than the `Buy-and-Hold` strategy. This aspect is fundamental, since market operators tend to favour risk control rather than exclusively monitoring the profit.

The coverage results shown in Figure 8(c) confirms the positive performance of the `Full` agent with 80% agreement, which performs decisions in only 30% of trading days. It confirms our approach to be flexible and robust, especially because, in the considered period, markets were affected by the aforementioned several unstable times. Indeed, as previously said, the `Full` agent learned to be aware of times of uncertainty and, therefore, performed much more opt-out*s*, resulting in a safer strategy with good returns and low risk. The outcome results from the equity curves in Figure 8(d) reinforce our hypothesis, as all of our agents beat the `Buy-and-Hold` baseline during the difficult times of 2007-2013 years range, where the latter suffered from the detrimental effects of the aforementioned crises. Most of the good returns of the `Buy-and-Hold` approach happened after that period, where our `Only-Long` agent showed similar behav-

ior. In general, we notice that the `Only-Long` agent remains reliable even in this scenario (albeit with lower agreement thresholds), in which, however, the higher alternation between positive and negative periods also encourages the adoption of a more polarized strategy, such as the one proposed by the `Full` agent.

5.3. Results on DAX market

We now illustrate the results of the last bunch of experiments, carried out in the German *DAX* market. As for the *S&P*500-L experiment, we also consider a period which spans the 2007-2017 years range. Figure 9 shows the sharpe ratio for each different configuration of our approach against the `Buy-and-Hold` baseline.



(a) Daily aggregated SR; (b) Monthly aggregated SR.

Figure 9: Results of the *DAX* experiment in terms of (a) daily aggregated Sortino Ratio; and (b) monthly aggregated Sortino Ratio. All of our configurations overcome the `Buy-and-Hold` strategy for at least two different agreement thresholds. On a monthly basis, the `Long+Short` agent provides the best result, at a 60% threshold.

The Sortino Ratio results in Figure 9 show that all our approaches, regardless of the agreement threshold (with the exception of the `Only-Long` agent at $80 - 100\%$), are promising agents to trade in this market. Nonetheless, all of them outperform `Buy-and-Hold` when using at least two different agreement thresholds. The best approach in this market is the `Long+Short` agent, which mainly benefits from the excellent performance of the `Only-Short` agent. Such a market has similar characteristics with the *S&*500-L market discussed before, especially due to the fact that it suffered from the same crisis periods; however, it is affected by a stronger impact of the market crashes, and a weaker recovery compared to the U.S. market. We observe also that the `Full` agent, in this context, has a similar behaviour compared to the *S&P*500-L experiment, confirming the intuition that longer observation periods are more effective to better balance this approach.

Figure 10(a) shows that all approaches are more profitable than the baseline for the 60% agreement threshold. It means that, in this unstable period of this

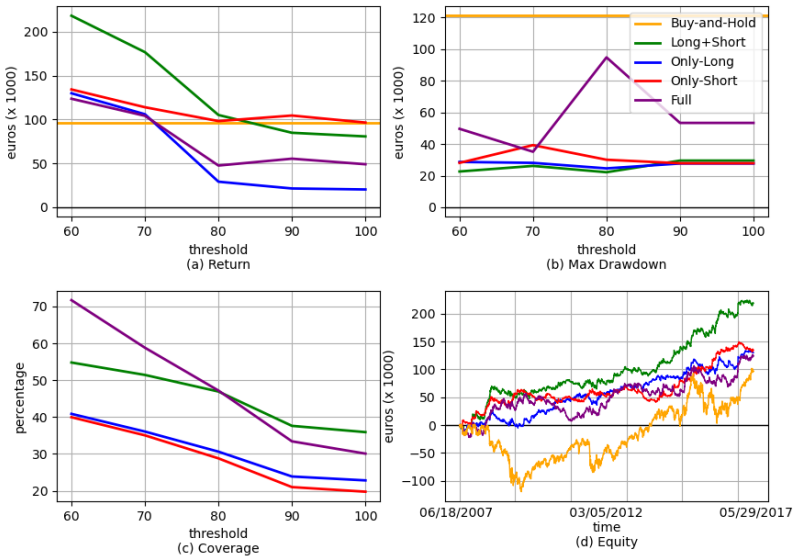


Figure 10: Results of the *DAX* experiment in terms of Return, Maximum Drawdown, Coverage and Equity.

specific market, an agreement between multiple agents must not be that strong if return is the key metric for performance. However, a key indicator in this experiment is given by the Maximum Drawdown. As previously discussed, such a metric is very significant for domain experts as it allows to estimate the risk of the adopted strategy. Figure 10(b) highlights how all our different configurations are considerably better than the `Buy-and-Hold` strategy in terms of risk management. Moreover, the coverage values denoted in Figure 10(c) let us observe that, similarly to previous scenarios, our `Only-Long` and `Only-Short` agents took decisions in about 40% of the trading days in their best configurations, while their `Long+Short` combination manages to keep the coverage sufficiently low (55%), but still maintaining a high degree of precision in the choices taken. Summarizing, all these indicators are positively reflected on the Equity curves (Figure 10(d)), where our proposed methods confirm to be clearly preferable to the passive `Buy-and-Hold` strategy in this market.

Therefore, from all the experiments conducted in this paper until now, we can conclude that our proposed approach, in general, provides convincing results when adopting one of the following configurations: (i) the `Only-Long` agent, on markets with a mainly upward trend; and (ii) the `Full` and the `Long+Short` agents, on more unstable markets, if the main objective is to contain the risk (in contrast to a passive investment), or to take better advantage of negative market phases. Finally, we point out that the performance of our method, compared to

the `Buy-and-Hold` baseline, has been statistically tested through the Student's t-test, through the process described in previous Section 4.3.6, by considering the best configurations mentioned above and the Sortino metric. The resulting *p-values* of 3.68×10^{-5}, 0.0219 and 0.0034, for the SP500-S, SP500-L and DAX market scenarios, respectively, make us reject the null hypothesis and let us say that the superior performance of our method is statistically significant. In the light of the results discussed above, we strongly believe that further refinements of such agents, possibly supported by accurate news data, can put the basis of reliable and robust real-world trading agents based on RL techniques.

6. Conclusion

The development of automated systems to perform intraday stock trading has gained a lot of popularity in the recent years, as it allows non-expert users to become wealthy in the stock market without actively acting on it. Several machine learning algorithms have been proposed to do such a task, but a deeper study on reward-based classifiers performing a final decision of several reinforcement learning classifiers, according to *ad-hoc* combinations of decisions for specific markets have not been extensively explored so far.

In this paper, we present a step toward such a task by proposing an ensemble of the same Deep Q-learning classifiers with different experiences with the environment. Our experiments results support our hypothesis that our approach can tackle uncertain and chaotic behavior of different stock markets thorough a flexible ensemble strategy, composed by: (i) several agents with different experiences in the market; (ii) diverse combinations of actions to be done by the trader; (iii) an ensemble strategy, done with different agreement thresholds; and (iv) multiple resolution data, representing past prices behavior in a complementary fashion. Such properties could help designing *ad-hoc* strategies to different markets and trading periods, with a high potential to be a recommended strategy to several different markets.

The performance experiments done in different markets let us conclude that different configurations of our method yielded promising results, especially in terms of Sortino Ratio, Maximum Drawdown and Equity curve when compared to the `Buy-and-Hold` passive investment strategy. Firstly, the `Only-Long` agent, which combines the decisions of several individual agents performing just *long* or *opt-out* actions in the market, showed to be more suitable to markets with a typical upward behavior (*i.e.* markets where *long*s are highly recommended) since, thanks to a controlled coverage, allows investors to significantly reduce the strategy risk without drastically compromising the expected returns. The other promising solution is represented by the `Long+Short` and `Full` agents, which behave in a similar (alternative) way: the first one is an ensemble of the `Only-Long` and `Only-Short` configurations, whereas the second one combines the decisions of individual agents performing not only *long* and *opt-out* actions, but also *short* operations. These approaches, rather than the `Only-Long` agent, are more appropriate to better mitigate the effects of negative market periods,

thus making them preferable to more unstable long-term trading strategies. Although the limitation of our approach lies in the fact that the data processed contain past prices data only, and they would need a second step of analysis based on news data for even better trading results, we think that these results can inspire several future work to be done as an extension of the current research. Thereby, a first research target would require to couple the stock trading systems with news based classification systems in order to better take into account the impact of external factors on these markets. Additionally, the validation of such proposed solutions to other types of markets, *e.g.,* bonds and commodities markets, is a natural continuation of this work. Then, the experimentation of several other kinds of agents, rather than the deep Q-Learning agents we proposed, is also a promising path to take. Finally, different network architectures, modules such as LSTM layers and other hyperparameters should also be explored in order to further increase the performance of the proposed solution.

Acknowledgments

The research performed in this paper has been supported by the "Bando Aiuti per progetti di Ricerca e Sviluppo - POR FESR 2014-2020 - Asse 1, Azione 1.1.3. Project IntelliCredit: AI-powered digital lending platform". Furthermore, we gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

References

Asad, M. (2015). Optimized stock market prediction using ensemble learning. In *International Conference on Application of Information and Communication Technologies* (pp. 263–268). IEEE.

Azhikodan, A., Bhat, A., & Jadhav, M. (2019). Stock trading bot using deep reinforcement learning. In *Lecture Notes in Networks and Systems* (pp. 41–49).

Barra, S., Carta, S., Corriga, A., Podda, A. S., & Recupero, D. R. (2020). Deep learning and time series-to-image encoding for financial forecasting. *IEEE/CAA Journal of Automatica Sinica.*, . doi:10.1109/JAS.2020.1003132.

Chaudhry, A., & Johnson, H. L. (2008). The efficacy of the sortino ratio and other benchmarked performance measures under skewed return distributions. *Australian Journal of Management*, *32*, 485–502. doi:10.1177/031289620803200306.

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

Coates, A., Abbeel, P., & Ng, A. Y. (2017). Autonomous helicopter flight using reinforcement learning. In *Encyclopedia of Machine Learning and Data Mining* (pp. 75 – 85). Springer.

Dase, R., & Pawar, D. (2010). Application of artificial neural network for stock market predictions: A review of literature. *International Journal of Machine Intelligence*, *2*, 14–17.

Deng, Y., Bao, F., Kong, Y., Ren, Z., & Dai, Q. (2017). Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, *28*, 653–664.

Fischer, T. G. (2018). *Reinforcement learning in financial markets - a survey*. FAU Discussion Papers in Economics 12/2018 Friedrich-Alexander-Universität Erlangen-Nürnberg, Institute for Economics.

Gao, X., Hongkong, S., & Chan, L. (2000). An algorithm for trading and portfolio management using q-learning and sharpe ratio maximization. In *International Conference on Neural Information Processing* (pp. 832–837).

Gosset, W. S. (1908). The probable error of a mean. *Biometrika*, *6*, 1–25. URL: `http://dx.doi.org/10.2307/2331554`. Originally published under the pseudonym "Student".

Gu, D., & Hu, H. (2002). Reinforcement learning of fuzzy logic controllers for quadruped walking robots. *IFAC Proceedings Volumes*, *35*, 91 – 96.

Gudelek, M. U., Boluk, S. A., & Ozbayoglu, A. M. (2017). A deep learning based stock trading model with 2-d cnn trend detection. In *Symposium Series on Computational Intelligence* (pp. 1–8). IEEE.

Hasselt, H. (2010). Double q-learning. In *Advances in Neural Information Processing Systems* (pp. 2613–2621). Curran Associates.

Henrique, B. M., Sobreiro, V. A., & Kimura, H. (2019). Literature review: Machine learning techniques applied to financial market prediction. *Expert Systems with Applications*, *124*, 226–251.

Hiransha, M., Gopalakrishnan, E. A., Menon, V. K., & Soman, K. (2018). Nse stock market prediction using deep-learning models. In *International Conference on Computational Intelligence and Data Science* (pp. 1351 – 1362). Elsevier volume 132.

Huang, W., Nakamori, Y., & Wang, S.-Y. (2005). Forecasting stock market movement direction with support vector machine. *Computers & Operations Research*, *32*, 2513–2522.

Jagric, T., Bojnec, S., & Jagric, V. (2015). Optimized spiral spherical self-organizing map approach to sector analysis the case of banking. *Expert Systems with Applications*, *42*, 5531 – 5540.

Jagric, T., Bojnec, S., & Jagric, V. (2018). A map of the european insurance sector are there any borders. *Economic Computation and Economic Cybernetics Studies and Research*, *52*, 283–298.

Kamijo, K., & Tanigawa, T. (1990). Stock price pattern recognition–a recurrent neural network approach. In *International Joint Conference on Neural Networks* (pp. 215–221). IEEE volume 1.

Kim, K.-J. (2003). Financial time series forecasting using support vector machines. *Neurocomputing*, *55*, 307–319.

Kimoto, T., Asakawa, K., Yoda, M., & Takeoka, M. (1990). Stock market prediction system with modular neural networks. In *International Joint Conference on Neural Networks* (pp. 1–6). IEEE volume 1.

Kingma, D. D., & Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*, .

Ładyżyński, P., Żbikowski, K., & Grzegorzewski, P. (2013). Stock trading with random forests, trend detection tests and force index volume indicators. In *International Conference on Artificial Intelligence and Soft Computing* (pp. 441–452). Springer.

Lee, J. W., Park, J., O, J., Lee, J., & Hong, E. (2007). A multiagent approach to q-learning for daily stock trading. *IEEE Transactions on Systems, Man, and Cybernetics*, *37*, 864–877.

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*. Atlanta, Georgia.

Magdon-Ismail, M., & Atiya, A. (2004). Maximum drawdown. *Risk Magazine*, *17*, 99–102.

Manojlovi, T., & Stajduhar, I. (2015). Predicting stock market trends using random forests: A sample of the zagreb stock exchange. In *International Convention on Information and Communication Technology, Electronics and Microelectronics* (pp. 1189–1193). IEEE.

Mihatsch, O., & Neuneier, R. (1999). Risk-sensitive reinforcement learning. In *Advances in Neural Information Processing Systems* (pp. 1031–1037). MIT Press.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, *abs/1312.5602*. URL: `http://arxiv.org/abs/1312.5602`.

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*, 529–533.

Moody, J., & Saffell, M. (1999). Reinforcement learning for trading. In *Advances in Neural Information Processing Systems* (pp. 917–923).

Moody, J., Wu, L., Liao, Y., & Saffell, M. (1998). Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting*, *17*, 441–470.

Murphy, J. (1999). *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*. New York Institute of Finance.

Neuneier, R. (1998). Enhancing q-learning for optimal asset allocation. In *Advances in Neural Information Processing Systems* (pp. 936–942). MIT Press.

Pendharkar, P., & Cusatis, P. (2018). Trading financial indices with reinforcement learning agents. *Expert Systems with Applications*, *103*, 1 – 13.

Plappert, M. (2016). keras-rl. `https://github.com/keras-rl/keras-rl`.

Rapach, D., Strauss, J., & Zhou, G. (2012). International stock return predictability: What is the role of the united states? *The Journal of Finance*, *68*. doi:`10.2139/ssrn.1508484`.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized experience replay. In *International Conference on Learning Representations*. Puerto Rico volume abs/1511.05952.

Schipper, K., & Smith, A. (1986). A comparison of equity carve-outs and seasoned equity offerings: Share price effects and corporate restructuring. *Journal of Financial Economics*, *15*, 153–186.

Schultz, W., Dayan, P., & Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, *275*, 1593–1599.

Shen, Y., Huang, R., Yan, C., & Obermayer, K. (2014). Risk-averse reinforcement learning for algorithmic trading. In *Conference on Computational Intelligence for Financial Engineering Economics* (pp. 391–398). IEEE.

Si, W., Li, J., Ding, P., & Rao, R. (2017). A multi-objective deep reinforcement learning approach for stock index futures intraday trading. In *International Symposium on Computational Intelligence and Design* (pp. 431–436). IEEE volume 2.

Singh, S., Jaakkola, T., Littman, M., & Szepesvari, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, *38*, 287–308.

Sutton, R., & Barto, A. (1998). *Reinforcement Learning: an introduction* volume 1. MIT press Cambridge.

Tan, T. Z., Quek, C., & Ng, G. S. (2005). Brain-inspired genetic complementary learning for stock market prediction. In *Congress on Evolutionary Computation* (pp. 2653–2660). IEEE volume 3.

Tesauro, G. (1995). *TD-Gammon: A Self-Teaching Backgammon Program*. Boston, MA: Springer US.

Tsurumine, Y., Cui, Y., Uchibe, E., & Matsubara, T. (2019). Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation. *Robotics and Autonomous Systems*, *112*, 72 – 83.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning* (pp. 1995–2003). New York, New York, USA: PMLR volume 48 of *Proceedings of Machine Learning Research*.

Watkins, C. (1989). *Learning from Delayed Rewards*. Ph.D. thesis Cambridge University.

Weng, B., Lu, L., Wang, X., Megahed, F., & Martinez, W. (2018). Predicting short-term stock prices using ensemble methods and online data sources. *Expert Systems with Applications*, *112*, 258–273.

Zhang, Y., & Wu, L. (2009). Stock market prediction of s&p 500 via combination of improved bco approach and bp neural network. *Expert Systems with Applications*, *36*, 8849–8854.