Noname manuscript No.
(will be inserted by the editor)

A Multi-Layer and Multi-Ensemble Stock Trader Using Deep Learning and Deep Reinforcement Learning

Salvatore Carta · Andrea Corriga · Anselmo Ferreira · Alessandro Sebastian Podda · Diego Reforgiato Recupero

Received: date / Accepted: date

Abstract The adoption of computer-aided stock trading methods is gaining popularity in recent years, mainly because of their ability to process efficiently past information through machine learning to predict future market behavior. Several approaches have been proposed to this task, with the most effective ones using fusion of a pile of classifiers decisions to predict future stock values. However, using prices information in single supervised classifiers has proven to lead to poor results, mainly because market history is not enough to be an indicative of future market behavior. In this paper, we propose to tackle this issue by proposing a multi-layer and multi-ensemble stock trader. Our method starts by pre-processing data with hundreds of deep neural networks. Then, a reward-based classifier acts as a meta-learner to maximize profit and generate stock signals through different iterations. Finally, several metalearner trading decisions are fused in order to get a more robust trading strategy, using several trading agents to take a final decision. We validate the effectiveness of the approach in a real-world trading scenario, by extensively testing it on the *Standard & Poor's 500* future market and the *J.P. Morgan* and *Microsoft* stocks. Experimental results show that the proposed method clearly outperforms all the considered baselines (which still performs very well in the analysed period), and even the conventional *Buy-and-Hold* strategy, which replicates the market behaviour.

Keywords Deep Learning · Deep Reinforcement Learning · Intraday Stock Trading.

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, Strategy 2- Program 3, Project AlmostAnOracle - AI and Big Data Algorithms for Financial Time Series Forecasting"

Department of Mathematics and Computer Science
University of Cagliari, Via Ospedale 72 - 09124 Cagliari, Italy E-mail: salvatore@unica.it, andrea.corriga@unica.it, anselmo.ferreira@gmail.com, sebastianpodda@unica.it, diego.reforgiato@unica.it

1 Introduction

The growing popularization of stock market trading has become a rich field to be explored by powerful and fast computing techniques. These tools have been widely explored, to provide better trading strategies and to maximize performance metrics such as, among others, profit, economic utility, or risk-adjusted return. Indeed, machine learning solutions have been proposed to stock trading even before the popularization of computers [21,16,22] and have potential to distribute wealth among investors with a reduced need for human intervention.

Financial forecasting techniques are usually divided into the branches of Technical Analysis (TA), which uses data acquired from the past to indicate future behavior of the market, and the Fundamentalist Analysis (FA), which analyzes economic metrics that may influence market future trends. Machine-learning based TA approaches treat stock market forecasting as a *classification problem*, where classifiers use labeled time series data from historical prices in order to assign the following trading signals: a *long* operation (*i.e.*, betting that the price will rise in the future) or a *short* operation (*i.e.*, to bet the opposite) should be done in order to maximize the profit. Some solutions in machine learning-based TA have been widely proposed in the financial forecasting literature using, for example, deep learning [35,25], Support Vector Machines [7,46], Random Forests [38,19], reinforcement learning [17,24], among others. Although not the focus of this paper, both TA and FA can be used together to perform financial trading [8,18].

The computer-aided stock trading is basically composed of two steps: (i) analysis of past market behavior, and (ii) taking the optimal stock trading decision. To perform such tasks, time-series data from past prices are usually considered as input. These data are usually given by the market under different resolutions (minutes, hours, days, etc) and contain information such as open prices, close prices, among others. However, such data contain a lot of uncertainty, specially because stock prices are often influenced by other factors such as political events, epidemics, performance of other markets and even the behavior of other investors [45]. These aspects model stock markets as dynamic, non-linear, non-parametric and chaotic environments [37].

In order to minimize such issues in machine learning based trading, several solutions have proposed the fusion of dozens of machine learning classifiers with the aim to get more computer-aided stock traders involved in the trading decision, which in turn minimizes the error of single classifiers trained on insufficient data. These algorithms tackle the no-free lunch theorem [42] by fusing several weak classifiers decisions with low correlation in order to yield a strong final classifier. In the stock market forecasting scenario, several literature approaches have been explored in that sense, such as using boosting [43], late fusion [3], dynamic ensembling [6], stacking [10], heterogeneous classifiers [29] among many others. Notwithstanding, such supervised approaches still do not deal efficiently with raw past prices data and their very known limitations. Therefore, an efficient solution should be designed to better cope with the un-

certainty and, at the same time, maximize ensembling steps in such a way to perform profitable and safer trading.

In this paper, we move towards tackling these issues by maximizing the performance of an ensemble based on Reinforcement Learning. Our stock trader is composed of a multi-layer and multi-ensemble classifier that acts in three steps and use two ensembling techniques at the same time. Our approach works by firstly applying several neural networks in stock prices data in the first layer, and the output will be used in a stacking ensembling technique in the second layer. Then, the set of trading signals from the neural networks are used as features and a reinforcement learning agent acts as a meta learner, maximizing rewards (profits) and yielding new trading signals through different iterations. Finally, in the last layer of our trader, these new signals are fused through another ensembling technique, namely a majority voting (late fusion) approach, in order to generate more robust final decisions. Simulations of intraday trading on the *Standard & Poor's 500* future market and the *J.P. Morgan* and *Microsoft* stock markets show that the proposed approach clearly outperforms not only the conventional *Buy-and-Hold* strategy (and, consequently, the market behaviour), but also several non-RL ensembling techniques and literature methods, which provide strong performance in the considered period.

In summary, the contributions of this paper are:

1. We propose a machine-learning based stock trading system composed by three different layers, that leverages both on deep learning and deep reinforcement learning benefits, which were proven to be effective in several other machine learning fields [44, 2];
2. We exploit a novel pre-processing step based on generating meta-features after converting price time-series into Gramian Angular Field images [39], in order to generate input data to be further processed in our stacking ensembling approach;
3. We propose the use of a deep reinforcement learning meta-learner, which processes the trading signals from Convolutional Neural Networks (CNNs) to generate the final trading decisions;
4. We combine two different ensembling steps (stacking in the second layer of our agent, and majority voting in the last layer) to maximize the robustness and stability of our trading strategy;
5. We evaluate our multi-layer and multi-ensemble approach in a real-world trading scenario, comparing our experimental results with several baselines and literature competitors, showing that it clearly outperforms their performance.

The rest of this paper is organized as follows: in Section 2, we discuss related work in stock trading using machine learning solutions. Section 3 motivates and present our algorithm in details. Section 4 describes the experimental setup considered to validate the proposed approach. Section 5 shows experimental results and, finally, Section 6 concludes this work and discusses possible future research directions motivated by the proposed method.

2 Background and Related Works

Several works in literature have been vastly proposed in order to perform automated stock trading. One of the pioneer works in this aspect is the work of Kimoto *et al.* [21], who used modular neural networks trained on various technical and economical information in order to determine when to buy and sell stocks. This work was extended later in [16] using recurrent neural networks that are more suitable to time series. The work of Lee and Park [22] used a similar recurrent neural network, but trained on eleven economic indicators. Other subsequent works have used more advanced machine learning classifiers to perform stock trading. Sun *et al.* [35] used Long Short Term Memories deep neural networks, an evolution of recurrent networks, on augmented market trading data. The work of Lin *et al.* [25] used a more evolved recurrent network to predict the stock trading signal by forecasting the opening, closing, and difference between these prices.

Other works have explored the use of non-neural network based classifiers. Fenghua *et al.* [7] used Support Vector Machine on features originated from the prices, such as trend, market fluctuation, and noise on multiple time resolutions to predict stock trends. Zhou *et al.* [46] employed the same classifier on a very heterogeneous dataset with sources from historical transaction data, technical indicators, stock posts and news to predict the directions of stock price movements. Another kind of classifier usually considered in previous works is the Random Forest (RF), as it consists of an ensemble of individual decision trees and, therefore, can be a powerful tool for trading. The work of Tan *et al.* [38] evaluated the robustness of RF to stock selection through fundamental/technical and pure momentum feature spaces. Finally, the work of Khan *et al.* [19] has assessed the effectiveness of RF on features from social media and financial news data.

The use of Reinforcement Learning (RL) in stock market prediction has shown state-of-the performance in several works in the literature and is considered a trending topic in stock market prediction. By considering the market as an environment that returns maximized rewards when the right trading signals are emitted, the stock trader agents are trained as much as possible in order to follow the market behavior by optimizing financial metrics. These metrics can be returns, sharpe ratio, among others. One of the pioneer works in this regard comes from the work of Neunier [39], that used a Q-Learning value-based RL approach to optimize asset allocation decision. Later, Mihatsch and Neunier [27] added in the Q-learning function the notion of risk. Gao and Chan [9] used as performance functions of Q-learning training the absolute profit and relative risk adjusted profit. Lee et *al.* [23] used four Q-learning agents that act serially in different steps of the trading procedure. Moody *et al.* [28] used a recurrent neural network in the RL pipeline for trading, an approach known as Recurrent Reinforcement Learning (RRL). Recent solutions proposed in this aspect are the work of Kang *et al.* [17], which modified and adapted the A3C RL algorithm and joined it with deep learning, and also the work of Lei

et al.[24], which proposed deep learning and deep reinforcement learning to adaptively select and reweight several features of financial signals.

3 Proposed Approach

As it can be noticed from the literature on RL research, there are a few explorations on maximizing ensembling steps on the reward-based training pipeline, with most of the works in this regard relying solely on one individual agent that receives one or multiple sources of information as input, and the output is done by one single agent that is trained on a customized number of iterations.

In our work, we propose to do exactly the opposite by maximizing ensembling steps in a three layer stock trader. Our approach receives stacked inputs from other classifiers and fuses its set of classifications through several training iterations, integrating these steps on an RL trader agent. Our three layer-stock trader, whose pipeline is better illustrated in Figure 1, is composed of the following layers:

1. Layer #1: **Stacking/pre-processing layer:** this layer acts on stock market data at different time resolutions, converting these time series data to images and using intra-day trading signals generated by 1000 CNNs as meta-features to be used by the next layer.
2. Layer #2: **Reinforcement Meta Learner:** we consider a reward based classifier to process the outputs from the previous layer, an ensemble process that is usually called stacking [41]. The stacking is performed by a meta-learner, which is, in our case, based on Deep Double Q-Learning [12].
3. Layer #3: **Ensembling Layer:** this layer fuses different signals of different training iterations of the meta learner in order to get a final decision.

We discuss the details of each layer in the next subsections.

3.1 Layer #1: Stacking trading signals with Convolutional Neural Networks

Several approaches in the literature have used CNNs to perform stock trading decision [4, 34, 20], treating the stock trading problem as a Computer Vision application by using time-series representative images as input. Following this approach, CNNs can highlight in their deeper layers details of these images that are difficult to find in their numerical format, helping in performing accurate classification and, therefore, better trading.

In the first layer of our trading algorithm, we leverage the benefits of these networks by generating a series of meta-features using hundreds of CNNs. This approach, inspired by the work of Barra *et al.* [4], is based on three steps: (i) time-series-to-image conversion; (ii) different CNNs processing; and (iii) fusing of trading results.

For converting time series to images, the authors consider the Gramian Angular Field (GAF) imaging inspired from Wang and Oates [39]. To build

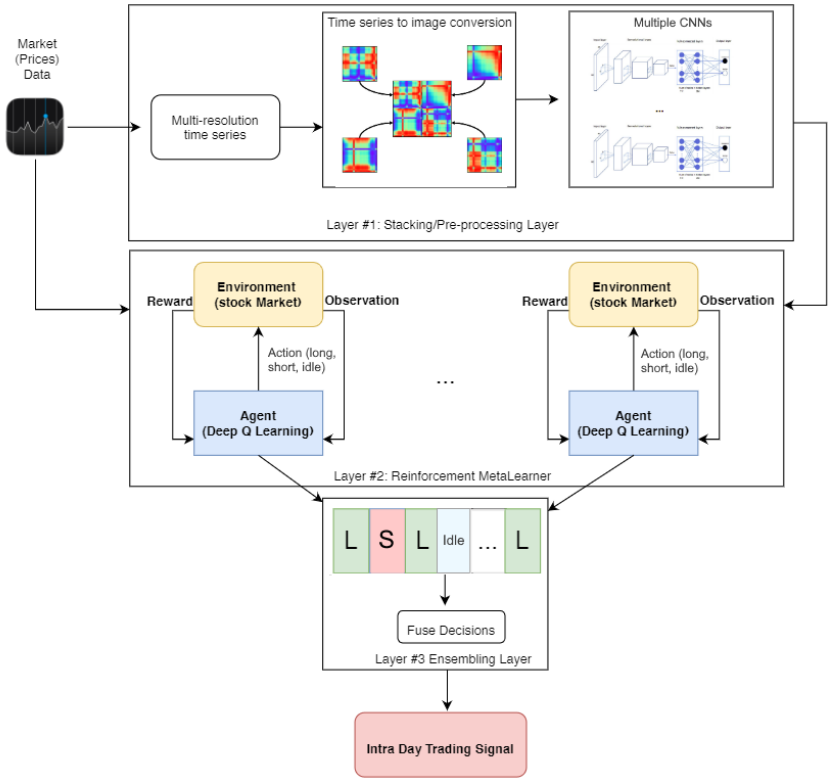

Fig. 1 The proposed three layered multi-ensemble approach. The first layer stacks decisions from CNNs, which are then used as observations (or states) in a reward-based meta-learner (second layer) in different training iterations. Finally, a last layer fuses the final decisions in a final trading signal.

such images, re-scaling of time-series are needed. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a time series with n components, the following normalization, usually called the min-max normalization [11] is done as follows:

$$\tilde{x}^i = \frac{(x_i - max(X)) + (x_i - min(X))}{max(X) - min(X)} \tag{1}$$

The scaled series, represented by $\tilde{X} = \{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n\}$ are then transformed to polar coordinates system by the following equation:

$$\begin{bmatrix} \theta_i = arccos(\tilde{x}_i), & \tilde{x}_i \in \tilde{X} \\ r_i = \frac{i}{N}, & \text{with } t_i \leq N \end{bmatrix} \tag{2}$$

Finally, Gramian Summation Angular Field (GSAF) and Gramian Difference Angular Field (GDAF) matrices can be easily obtained by computing the sum/difference between the points of the time series:

$$GSAF(i,j) = [cos(\theta_i + \theta_j)] = \tilde{X}' \cdot \tilde{X} - \sqrt{I - \tilde{X}^2}' \cdot \sqrt{I - \tilde{X}^2}$$
$$GDAF(i,j) = [sin(\theta_i + \theta_j)] = \sqrt{I - \tilde{X}^2}' \cdot \tilde{X} - \tilde{X}' \cdot \sqrt{I - \tilde{X}^2} \quad (3)$$

The result of such transformations are 1-channel 2D matrices that represent a heatmap, whose values range from 0 (*black*) to 1 (*red*). In a second step, a colormap is applied to the input matrix, resulting in a three channel output matrix. In the work of Barra *et al.* [4], best results were reported using GDAF.

Converting stock market time series to images in this approach has the limitation of not generating too big images, making such inputs unfeasible to deep convolutional neural networks. Time-series data show an important feature that is the variation of data in the time. They way the data change can, therefore, provide important interpretation about how an event evolves. Thus, the authors propose to aggregate the data under K different intervals of time, using these K time series as sub-regions of the final converted image. This image has a final resolution of 40×40 pixels, and is processed by an ensemble of VGG-based CNNs, with the architecture better detailed in Figure 2. The final trading signal is the majority voting of the outputs.

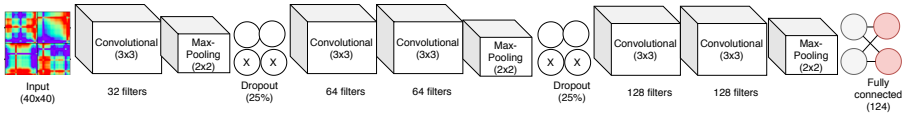

Fig. 2 The VGG-based CNN used to pre-process stock market time series to image data.

Our extensions of the work in [4] for this paper are two-fold: (i) in [4], twenty different CNNs (each trained with a different weight initialization) are fused to provide the final (daily) decisions. We noticed that the adoption of a naive consensus mechanism to fuse these CNN outputs did not allow the approach to be sufficiently robust in markets with different characteristics. Hence, in this new work, we improved the method by exploiting Reinforcement Learning to find the optimal policy to fuse these CNN outputs, with the aim of favouring the most reliable networks, and penalising the less precise ones; (ii) to do so, we first extend the numbers of CNN used, by considering 10 best training epochs over 100 CNNs, with different initialization parameters (not previously used in [4]), to generate daily trading decisions. Then, these 1000 array of decisions are used, in a novel way, to generate meta-features to be employed as inputs of several RL agents. Therefore, the 100 CNNs act as a first layer (or feature generation/pre-processing) of our new approach, not used to trade as the 20 CNNs from [4] are used for; and (iii), finally, we train these RL agents (with different experience of the environment) based on Neural Networks (Deep Q-Learning) with these features to automatically find the possible best ensembling policies, to perform a majority voting over these agents (as described in next Sections 3.2 and 3.3), in order to get final decisions to be tested in real-world trading scenarios.

3.2 Layer #2: Reinforcement MetaLearner

Reinforcement Learning (RL) [36] is a self-taught learning process which can be represented by a Markov Decision Process (MDP) [31]. 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 the agent is allowed to do, thus in our case this set is finite);
- $P_a(s, s')$ is the probability that an action a in state s will lead to the state s';
- $R_a(s, s')$ is the immediate expected reward received in the transition from a state s to a state s' doing an action a.

The main goal of the reinforcement learning process is, therefore, to find a policy π that returns the highest reward. To perform such a task, the agent can try different policies in a training stage, but only one of those has the highest reward possible, which is called the *optimal policy*. Several approaches can be used to train RL agents, among them, a promising strategy is using evolutionary algorithms [15].

When applied in machine learning problems, the following analogies are usually made: (i) the state is a feature vector which represents the input of the agent (*e.g.*, in our case, a vector containing input data); (ii) the actions are the outputs of the machine learning agent (*e.g.*, the trading decision to do); and (iii) the reward function is built at a training stage and returns higher rewards to correct actions (*e.g.*, the market profit generated by the local trading decision). Thus, differently from supervised learning (when agents learn from annotations) and unsupervised learning (where agent searches for hidden structures in the data, to identify clusters), in reinforcement learning the agent learns how to maximize a reward function in a given environment.

In our proposed Reinforcement Trading (RT), the trading signals vector from the previous layer is described as $\{p_1, ..., p_n\}$, and the return at time t is defined as $r_t = p_t - p_{t-1}$. Trading decisions such as long, idle and short are represented as $A_t \in \{1, 0, -1\}$. Therefore, in RT, the profit R_t is calculated as [33]:

$$R_t = A_{t-1} r_t - c|A_t - A_{t-1}| \tag{4}$$

where c is the transaction cost paid to the brokerage company when two 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

$$\max_{\Theta} \sum_{t=1}^{T} R_t|\Theta, \tag{5}$$

where Θ is the family of parameters of the model (agent). 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.

In our approach, we use the benefits of reinforcement learning classifiers to act in a stacking scenario. Here, the RL agent acts like a *meta-learner*, using the outputs from previous layer as states. In its training, the agent learns to maximize rewards, which in our paper is calculated as:

$$Reward = \begin{cases} close - open & \text{if action is } long \\ -(close - open) & \text{if action is } short \\ 0 & \text{if action is } idle \end{cases} \quad (6)$$

where *open* and *close* are, respectively, the opening and closing prices of the mar

Our meta-lket in the considered day.earner is based on Double Q-Learning (DQL) agents [13]. In such an algorithm for Deep Reinforcement Learning (DRL), Deep Q Networks (DQN) are used in order to learn a reward-inspired function called Q-function $Q(x, a)$, which is disposed in a table. This function is calculated as:

$$Q(x, a) = D(x, a) + \gamma \max_a Q(y, a), \quad (7)$$

where x and y are a states, a is an action, $D(x, a)$ is the reward from taking action a and γ is the discount factor, which controls the contribution of rewards in the future. Therefore, in Equation 7, the Q-value yielded from being at state x and performing action a is the reward $D(x, a)$ plus the highest Q-value possible from the next state y also performing action a, discounted by the discount factor.

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

$$Q(x, a)_{new} = Q(x, a)_{old} + \alpha[(D(x, y, a) + \gamma \max_b Q(y, b) - Q(x, a)_{old})], \quad (8)$$

where α is the learning rate, $D(x, y, a)$ is the immediate reward from taking action a and moving from state x to y and $\max_b Q(y, b)$ is the maximum future reward considering all possible actions b in the new state y.

Such a parameterized table is learned through network weights in deep q-learning, found through backpropagation of errors in a training stage. DQN loss functions are calculated as:

$$Loss = [Q(x, a, \Theta) - (D(x, a) + \gamma \max_a Q(y, a, \Theta')]^2 \quad (9)$$

Therefore, the goal of the network is to find parameters Θ that minimize Equation 9.

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 our considered DQL are the target networks [13] and dueling networks [40]. The first one is used just to calculate the target from Equation 10 below. 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. When using target networks, the weights update of DQNs happens as it follows:

$$\triangle w = \alpha[(D(x,y,a) + \gamma \max_a Q(y,a,w)) - Q(x,a,w)] \bigtriangledown_w Q(x,a,w), \quad (10)$$

where $\triangle w$ is the change in weights to perform in the training stage, $Q(x,a,w)$ is the current predicted Q-value and $\bigtriangledown_w Q(x,a,w')$ is the gradient of the current predicted Q-value. The part of the equation $D(x,y,a) + \gamma \max_a Q(y,a,w) - Q(x,a,w)$ is commonly known as Temporal Difference (TD) error. Finally, the part $D(x,y,a) + \gamma \max_a Q(y,a,w)$ is also called *target*.

The use of target networks is justified since the same parameters (weights) are used for estimating the target $D(x,y,a) + \gamma \max_a Q(y,a,w)$ and the predicted Q-value $Q(x,a,w)$ in Equation 10. 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 the highest Q-value); and (ii) use the target network to calculate the target Q value of taking that action at the next state.

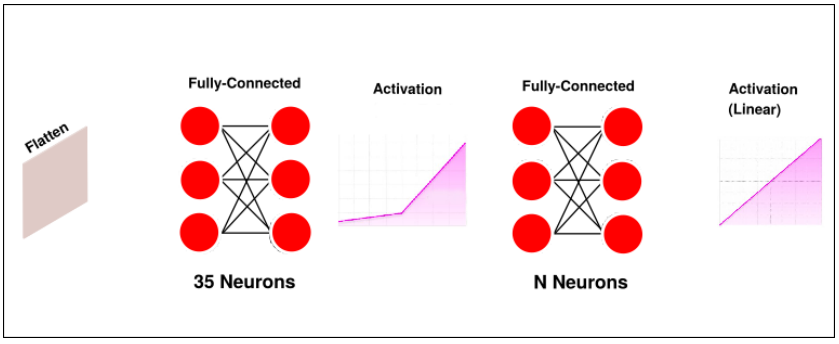

Fig. 3 The neural network architecture of our proposed multi-DQN agents for stock trading. The activation of the first layer is chosen through optimization experiments.

Additionally, dueling networks [40] help the network to calculate each of the values ($D(x, y, a)$ and $\max_b Q(y, b)$) in Equation 7 separately. 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 $Q(x, a)$. To avoid backpropagation issues created by calculating Equation 7 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 3 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 fully connected layer with 35 neurons. The activation of such neurons are activated after optimization experiments, which depend on the market considered (we discuss how it is done in Section 5.1 for one dataset), and N neurons fully connected layer with linear activation, where N is the number of decisions to take in the stock market (we chose $N = 3$, which are long, short and long operations).

3.3 Layer #3: Ensembling Multiple Learners

In this last layer, we propose the fusion of several multiple DQN agents trading signals, in order to take more decisions into account for trading. After multiple different training iterations (or epochs) of the above meta-learning agent with the environment, different actions in the market are done, as Figure 4 illustrates.

Given such outputs from the previous meta-learners, the final agent works in an ensemble (or *late fusion*) fashion, which considers the majority voting of decisions to generate the final trading signal. By taking a vector $L_t = \{l_1, l_2, ..., l_n\}$ of n iterations (epochs) decisions for a day t, where $l_i \in \{L, H, S\}$, the ensembling layer fuses them in just one trading signal l_t as the following:

$$l_t = Mode(L_t) \tag{11}$$

where $Mode(L_t)$ is the most frequent value in L_t. In our approach we consider L_t as a 25-dimensional vector (empirically chosen), which means that we ensemble 25 meta-learner training iterations decisions.

The idea behind this approach is that the same agent trained at different iterations can be complementary, as the agents have different experiences with the environment thus making the approach more robust against uncertainties of the stock markets behavior. Indeed, by ensembling the individual agents decisions through the majority voting, we require the final agent to perform a *long*, *idle* or a *short* action only when the fusion has a good confidence about the choice (*i.e.*, when there is a wide agreement among the individual agents).

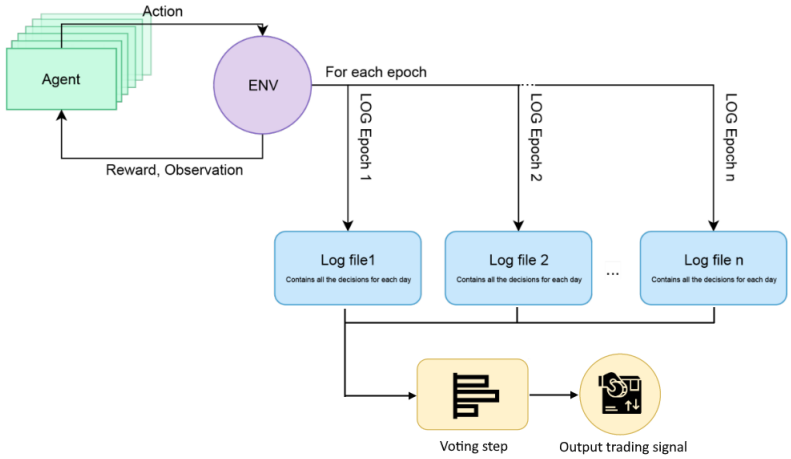

Fig. 4 Fusion Layer pipeline of our proposed multi-layer agent. Multiple agents trained after multiple series of different iterations with the environment perform intra-day stock trading, done by choosing among different combinations of actions. The final action to take is decided by majority voting of decisions.

4 Experimental Setup

This section describes the experimental setup adopted in this paper. It reports the markets, metrics, state-of-the-art approaches considered for comparison and implementation details of the proposed approach.

4.1 Datasets

We perform the experimental validation of our approach by using historical data from the *Standard & Poor's 500* future index (S&P 500), as well as the *JP Morgan* (JPM) and *Microsoft* (MSFT) single stock indexes. These data can be usually found at different time resolutions, or *granularity* (*e.g.*, *5-minutes*, *10-minutes*, *1-hour*, *etc.*). The dataset structure is summarized in Table 1: each entry specifies the timestamp of the considered time slot (technically known as *candle*), among with its open, close, high and low prices. These prices are usually denoted by market points (1 point=$50,00 for the S&P 500 future index, while 1 point=$1,00 for single stocks). We run our first layer of CNNs over these datasets, thus extracting 1000 different predictions (long, short or idle) for each day of the chosen period, to be used as input for the second layer (the meta-learner).

Table 1 Dataset structure (1-hour time resolution); prices are expressed in market points.

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
...	...	...	...	...	...

4.2 Metrics

To compare the proposed method against the baselines (described in the next paragraph), we adopt a set of state-of-the-art trading metrics, that help to evaluate each algorithmic trader in terms of both classification and trading performance.

The first metric we evaluate is the coverage, which indicates how many times the trading agent enters the market. Given $|X'|$ the number of trading days the agent performs long or shorts, and $|X|$ the total number of trading days the agent was working, the coverage is calculated as in Equation 12.

$$COV = \frac{(|X'|)}{|X|}. \tag{12}$$

In our classification/trading algorithm, we may want to know the precision of long and shorts. The precision calculates the number of corrected classified samples of a given class, dividing it by all instances classified for that class (right or wrong). For example, the long precision can be calculated as:

$$Precision_L = \frac{TL}{TL + FL}, \tag{13}$$

where TL is the number of longs correctly classified longs and $TL + FL$ is the total number of correctly and incorrectly classified longs.

The Maximum Drawdown (MDD) estimates the risk in the time period taken into account [26]. Low values of maximum drawdown are usually preferred. Formally, if P is the peak value before the largest drop and V is the valley value before a new high value is established, its formal notation can be established as

$$MDD = \frac{(P - V)}{P}. \tag{14}$$

We also use a metric that measures how worth the investment is according to the risk: the Return Over Maximum Drawdown (ROMAD). This metric is largely used to evaluate the gain or loss in a certain period of time. More formally, it measures the return of a portfolio as a proportion of the *Maximum Drawdown* level, as shown in Equation 15, where *Portfolio Return* denotes the difference between the final capital and the initial one.

$$RoMaD = \frac{Portfolio\ Return}{MDD}. \tag{15}$$

In practice, investors want to see high RoMaDs, which means that the portfolio returns are high and MDDs are low. High RoMaDs mean that the risk is low and the return is high enough to worth the trading. We consider this metric the most important one and will use it to rank the results, as it measures how many times the return is higher than the risk assumed.

4.3 Baselines

In order to make a complete comparison between the proposed approach, some other well-performing ensembling techniques, and literature competitors, we selected several significant baselines.

More in details, the baselines we consider can be divided into two groups. The first group includes some variations of our approach, where the second and third layers of our stack are replaced with alternative and well-performing non-RL ensembling techniques. They are:

1. the first layer only of our approach, through a majority voting over the 1000 CNN trading signals, which shows to be the strongest baseline, with a very competitive behaviour in the considered periods;
2. the first layer only of our approach, but considering a threshold agreement of at least 50% of the same CNN trading signals;
3. the same first layer only of our approach, but now considering a threshold agreement of at least 70% of the same CNN trading signals.

On the other hand, the second group of baselines is represented by some literature works that propose trading techniques based on machine learning. As far as we know, however, the existing works that punctually report the used out-of-sample periods, as well as the results in terms of economic performance of their methods, are very few for intra-day stock trading. Among them, we have selected the following two:

1. the approach proposed by Sezer *et al.* in [32], called *CNN-TAr*, where the authors encode trading signals (*e.g*: EMA, SMA, *etc.*) into 15x15 images, and train a CNN to perform daily trading;
2. the technique presented by Calvi *et al.* in [5], where the authors propose a *Support Tensor Machine*, called *LS-STM*, *i.e.* a tensor extension of the best known support vector machine, to perform daily trading.

Finally, together with all the baselines mentioned above, for each period and market considered, we compare our proposed method against the Buy-and-Hold strategy, which consists of a passive investment where the investor buys the stocks at the first day of the considered period, and sells them at the end of the period. Since this strategy essentially replicates the behaviour of the market, it represents a valid benchmark to compare the performance of the proposed method against the market itself.

4.4 Implementation Details of the Proposed Approach

The approach proposed in this paper has been developed in *Python* using KERAS-RL library [30] and was run in an *Intel i7-8700k* with a *64-bit* Operating System (*Ubuntu Linux 18.04* for tge development and learning stages, *Microsoft Windows 10* for the simulation step) with *64 GBytes* of *RAM* and a Nvidia TITAN XP GPU. Our method includes several DQN agents, equipped with target networks and dueling architectures. In its second layer, it considers agents trained for 25 epochs/iterations (empirically chosen) and the final trading signal is given in the last layer of our method, by considering the majority voting of these decisions.

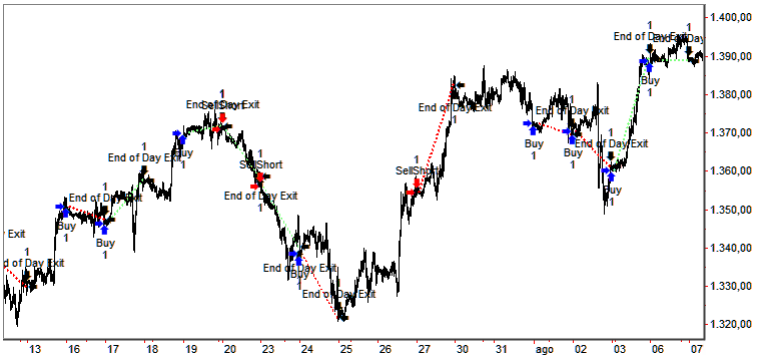

Fig. 5 Example of real-world trading simulation through MultiCharts.

The output of our last layer is hence encoded into a *(datetime, decision)* `.csv` file. Then, each real-world trading simulation is performed through MultiCharts[1], a Windows-based trading platform. Figure 5 shows how MultiCharts implements our trading decisions: when performing *long* actions, the tool executes a buy order at the begin of the day, immediately after the market opens (recall we perform daily trading), and a sell order immediately before the market closes; for *shorts*, the opposite (i.e. it performs a sell order when the market opens, and a buy order when it closes, by exploiting the *uncovered sales* mechanism); finally, *idle* actions are simply ignored. Notably, the source code of our solution is available for download at a public repository[2].

5 Experiments

In this section, we validate the quality and trading performances of our approach through several simulations, in which we perform intra-day trading in

[1] `https://www.multicharts.com/`

[2] `https://github.com/Artificial-Intelligence-Big-Data-Lab/A-Multi-Layer-and-Multi-Ensembled-Stock-Trader-Using-Deep-Learning-and-Deep-Reinforcement-Learning`

several stock markets, through a *backtesting* mode. Our experiments consist of two stages: in the first stage, we perform a preliminary study on the metalearner parameters to optimize the second layer of our approach; then, we compare our proposed strategy against the baselines introduced in previous Section 4.3, in a real-world trading scenario.

5.1 Metalearner Parameters Optimization

Before showing the comparison with baselines and other relevant literature works, we report the outcomes of the preliminary parameter optimization performed to tune the second layer of our approach. To do so, we consider a small out-of-sample subset, that we call *validation set*, to analyze the results. To this, we exploit our S&P 500 dataset, spanning the date interval between February 1st of 2012 to September 5th of 2018 for training, and from September 6th of 2018 to March 5th of 2019 for validation.

The first set of parameters are the optimizers of the deep-Q networks. Indeed, this is a very important parameter, because the goal of the reinforcement learning network is to minimize the difference between the different Q-functions in Equation 9. Therefore, the optimizers update the weight parameters in Θ to perform such an optimization. The Loss Function acts as a guide to the optimizer, telling if it is moving in the right direction to the global minimum.

The second parameter comes from the RL dilemma known as *exploration versus exploitation*, that requires to choose between keep doing the normal learning process (exploitation), or try something new (exploration). In our deep-q agent, exploitation means always taking the action with the highest Q-value, and explorations means taking random actions with a given percentage of probability. We vary the probability of explorations in the greedy policy used in our metalearner.

Finally, the final parameter we want to tune is the activation of the 35-neurons layer of our Q-network. Activation functions are one of the most important components of a neural network model. They determine the output of a model, its performance and efficiency of training.

In summary, the possible values of each parameter are:

1. Adagrad, Adam, Adadelta, Adamax and Rmsprop, as optimizers;
2. 30%, 50%, 70% and 90%, as percentage of explorations;
3. TanH, Selu, Relu, Linear and Sigmoid, as activation functions.

Table 2 shows the top-10 results considering the RoMaD metric out of 100 experiments performed, showing their classification and trading performance.

At first, we observe that the Adadelta optimizer presented the two best positioned results in terms of RoMaD, by using, respectively, 50% of explorations with TanH activation and 30% of explorations with Relu activation. The TanH activator presented a return that is 9.36 higher than the risk assumed in trading, but it trades only 20% of the time. Such a strategy showed

Table 2 Top-10 results, in terms of RoMaD, of the parameter optimization experiment performed over the validation set, by considering different metalearner optimizers, layer activations and different levels of explorations in the metalearner policy. Classification metrics are percentages in $[0, 1]$ range, while trading metrics are shown in market points (1 point = 50 USD).

Optimizer	**Explor.**	**Activator**	*Long Prec.*	*Short Prec.*	*Accuracy*	*Cover.*	*MDD*	*Return*	*RoMaD*
ADADELTA	0.5	TanH	0.58	0.38	0.48	0.20	38.75	362.75	9.36
ADADELTA	0.3	Relu	0.53	0.59	0.55	0.77	79.50	520.25	6.54
ADAM	0.9	Selu	0.58	0.57	0.57	1.00	135.75	833.50	6.14
RMSPROP	0.3	Selu	0.53	0.74	0.60	0.42	121.00	541.00	4.47
ADAGRAD	0.5	Selu	0.20	0.62	0.50	0.14	54.25	223.75	4.12
ADAM	0.5	Relu	0.55	0.52	0.53	0.80	151.25	598.25	3.96
ADAGRAD	0.7	Sigmoid	0.57	0.53	0.55	0.86	200.50	610.50	3.04
RMSPROP	0.7	Sigmoid	0.52	0.50	0.51	1.00	134.75	386.00	2.86
ADAGRAD	0.3	Linear	0.50	0.56	0.53	0.34	117.25	321.50	2.74
ADAM	0.3	Sigmoid	0.57	0.55	0.56	0.41	115.75	300.50	2.60

the lowest risk, illustrated by a maximum drawdown of 38.75. It also showed the best long precision (58%), which entails its very well position in terms of the RoMaD metric. The ADAM optimizer showed three configurations positioned in the top-10 approaches. In particular we noticed that, when using a very high probability of explorations (90%) and the Selu activation, it increased the profit (the highest of all, with 833 market points), but also the risk (an MDD of 135.75, the third highest of the experiments). Even in that scenario, it showed a RoMaD of 6.14, the third best of all. However, that comes with a 100% coverage, which means that the agent was more exposed at the market risk.

Conversely, the RMSPROP activation function had only one configuration in the RoMaD top-10 results, however, it showed the best accuracy of all (60%), and also the best precision of shorts (a surprising 74%). However, that high short precisions do not yield a very high return in the market period analyzed, specially considering that the agent trades only 42% of the time. However, that configurations still had a good risk-return trade-off, with a 4.47 RoMaD. Finally, ADAGRAD optimization approach had three configurations in the top-10 results. Its best configuration uses 50% probability of explorations in the RL agent training, with the Selu activation approach. However, it showed the lowest long precision and accuracy of all approaches, but its fifth best position in that table is mainly due to the fact that such an agent enters the market only 14% of the time (second lowest of all), but also providing the second lowest risk of all the approaches.

5.2 Experimental Results and Comparison With Baselines

We now focus on the comparison of our proposed technique against the baselines previously summarized in Section 4.3. As previously outlined, we start our analysis of the experimental results by comparing the performance of our multi-ensemble and multi-layer approach, against several well-performing variations of it. These variations only rely on the first layer of the stack, thus not providing the reinforcement learning components and ensembles. However, the

experiments show how our second and third levels based on RL both help to significantly improve the overall performances of the approach.

To obtain such a result, we considered, for all the strategies, a real-world trading scenario where we assumed to trade a single future on the S&P 500 market, with the same initial investment capital of 4,000 market points (200,000 USD). As a robust training period, we consider the same 7-years period of Section 5.1, starting from February 1st of 2012 to September 5 of 2018, while as test set, we exploited a time frame spanning September 6th of 2018 to August 30th of 2019 (the last trading year of our dataset). Note that it represents an interesting period to analyse, since it is characterized by a first phase in which the market suffered high volatility and a significant fall, mainly due to commercial war between the USA and China, opposed to a second phase, in which the market experienced a strong recovery, with very positive performance. Table 3 shows the results of our experiments.

Table 3 Summary of the comparison between our RL layers and the non-RL ensembling techniques outlined in Sec. 4.3, in an out-of-sample trading scenario (September 2018 to August 2019). Best results per trading metric are underlined. Trading metrics are expressed in market points (1 point = 50 USD) and classification metrics are in the interval [0,1].

	Classification metrics			*Trading metrics*		
Method	**Long prec.**	**Short prec.**	**Cover.**	**MDD**	**Return**	**RoMaD**
RL-ensemble *(proposed)*	0.59	0.50	1.00	221.74	**1265.50**	**5.70**
Majority ensemble	0.60	0.69	0.55	162.75	901.00	5.53
Thresh. voting ensemble (50%)	0.61	0.60	0.35	**130.75**	662.75	5.06
Thresh. voting ensemble (70%)	0.31	0.00	0.06	186.74	-89.24	-0.47
Buy-and-Hold baseline	-	-	-	598.00	45.50	0.08

First, let us analyse the Buy-and-Hold strategy, which represents a passive investment approach, commonly used as a trading benchmark since it essentially replicates the market behaviour. In our test period, B&H performances return a modest income, but with a high investment risk (due to the market fall at the end of 2018). Since S&P500 is a usually growing market, the Buy-and-Hold strategy is generally effective, but it is in such periods that it shows its limits. This is particularly relevant when operating with the *leverage*, in which sudden and sharp falls can lead to a loss of the invested capital, and therefore require to provide new investments to keep the position with the broker active (*margin calls*). Overall, it is therefore very important that advanced trading strategies should be able not to follow the market trend during negative phases, while, conversely, to detect and exploit the positive trends.

For this reason, when looking at the results in Table 3, we did not take into close consideration the results in terms of classification (*e.g., precisions*), and in particular we did not evaluate them in terms of accuracy. This is because, compared to a standard classification task, in trading it is not important the amount of correct choices, but rather to correctly select the operations with a higher specific weight (*i.e.,* days with higher volatility), since a single wrong choice can compromise the result of a high number of correct choices. On the other hand, we give particular emphasis to the most significant indicators in

the analysis of the financial strategy, i.e. the return, the MDD and – specifically – the RoMaD, which quantifies the ratio between the obtained profit and the assumed risk, thus making it possible to determine the safest and most robust solution to adopt.

Looking at Table 3, we can summarize the results of the competitors of the proposed approach as it follows. The strongest baseline is represented by an ensemble of independent neural networks, obtained as the majority voting of their decisions. This "naive" ensemble is therefore performed over the first layer of our approach. On the practical side, it shows to be a very effective strategy, with an extremely positive behaviour which overperforms the market throughout the considered test period. It provides the second highest return (901 market points, or $45,050.00$ USD), the second lowest MDD and also the second highest RoMaD of 5.53. However, this agent is very cautious about its decisions, not entering the market in almost half of the trading days (i.e. coverage is 0.55)

The rest of the approaches presented in the third and fourth row of Table 3 represent two variations of the aforementioned first-layer majority voting ensemble. These variations are obtained by imposing a minimum agreement threshold on the majority decision. However, from our experiments, this technique has shown to cause a regular decrease of the RoMaD as the threshold used increases. Therefore, we only reported results with agreement thresholds of 50 and 70%, for illustrative purposes. Going into detail, with a 50% decision agreement, the trader agent enters the market 35% of the days, providing the third highest return, which, according to its RoMaD, is 5.06 times higher than the risk. Notably, this solution achieves the lowest maximum drawdown value, thanks to a balanced combination of low coverage and high precision of operations, making it an effective and low-risk strategy for the considered period. Conversely, the ensemble with a 70% of majority voting agreement gives the overall worst performances, with a negative return and – consequently – a negative RoMaD. This approach is useful to only show how a reduction in coverage does not automatically translate into a decrease in risk, and also to denote the existence of a non-negligible variance in the predictions of individual CNNs.

At last, we analyse the results of our proposed approach, based on a metalearner that autonomously learns how to ensemble the CNN decisions. The metalearner is then executed in separate iterations, and its decisions are subjected to a second level of majority ensembling (in order to balance the different first-level ensembling strategies found by the metalearner). Such a multi-ensemble method provides the best return so far (1265 market points, equivalent to 63,250.00 USD, which means 5,270.83 USD per month in a very difficult period). Even though such a profit came with the second highest MDD of all (221.74 market points), it is still more than a half the value of the B&H strategy, and only the 0.055% of the initial investment (4,000 market points), which can still be considered a modest risk for a financial strategy.

Notwithstanding, even in this scenario, we increased our best competitor return by 28.78%, also providing the highest RoMaD of 5.70. In other words,

this means that the given return is almost six times the value of the risk assumed. To better understand how our approach boosts the best baseline (first-layer majority voting ensemble), in Figure 6 we show their equity curves, together with the Buy-and-Hold baseline that, as previously said, indicates the market behaviour.

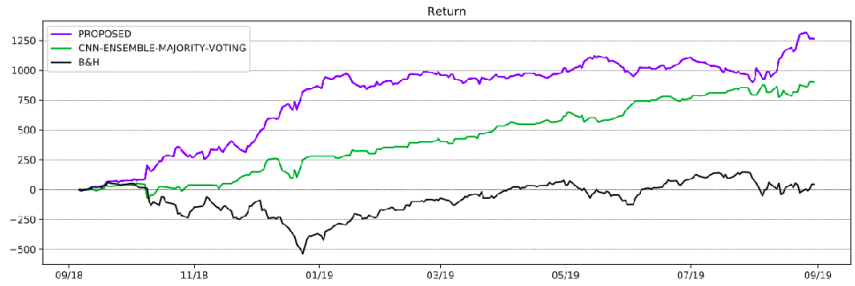

Fig. 6 Equity curve of our RL-ensemble approach, against the best non-RL baseline (majority voting ensemble) and the Buy-and-Hold strategy.

According to Figure 6, the proposed approach is not only consistently above, in terms of return, both to the market and to the competitor throughout the period considered, but also appears to be more effective in generating profit during negative market phases. Moreover, it is the only one of the three strategies to have a consistently positive return over the period in question. Following these results, we can therefore say that the adoption of the second layer based on the RL-ensemble technique is promising in terms of effectiveness of the final trading strategy provided, and seems to globally confirm the validity of the proposed approach.

Finally, to conclude the analysis of the performance of our approach, we start the comparison of our method against the last two baselines discussed in Section 4.3: the approach proposed by Sezer *et al.* in [32] (*CNN-TAr*), and the one by Calvi *et al.* [5] (*LS-STM*), which represents two different strong competitors in literature. To do that, we consider new symbols (MSFT and JPM) and periods, with respect to previous experiments, that has been used in the considered works.

Table 4 reports the results achieved in terms of annualized return, i.e. the percentage of annual return on the initial investment (which we assumed to be equal to the stock/future value at the begin of the period), together with the Buy-and-Hold performance, which allows us to estimate the behaviour of the market. The first period we considered spans the date interval between January 1st of 2007 and December 31st: in these six years, both JPM and MSFT recorded a slightly negative trend, mainly due to the sever economic crisis of 2008, associated with sub-prime mortgages. However, both our proposal and the CNN-TAr approach manage to generate a positive return in this phase: in particular, our approach has proved to be comparable and even mildly superior

Table 4 Comparison of the real-world trading performance, in terms of annualized returns, between our method and some literature competitors when considering different periods and markets. Our method clearly outperforms all the competitors in the considered settings.

	Annualized return		
	Period: 2007-2012		*Period: 2009-2015*
	JPM stock	MSFT stock	*S&P*500 future
OUR PROPOSAL	**11.3%**	**17.53%**	**23.20%**
CNN-TAr [32]	9.19%	4.44%	*n.a.*
LS-STM [5]	*n.a.*	*n.a.*	$\sim 10.15\%$ [1]
Buy-and-Hold	-1.67%	-1.93%	21.33%

[1] The annualized return value is approximate, since the authors in [5] reported the cumulative profit chart only.

when trading on the JPM symbol, while it showed to be extremely performing against this competitor for what concerns the MSFT index.

On the other hand, with regard to the second period, which covers dates between January 1st of 2009 and December 31st of 2015, we examine the performance of our approach when trading on the S&P500 future against the LS-STM competitor. We notice that the achieved annualized return of the latter is approximated, since in the original work from Calvi et al. [5], the authors plotted the cumulative profit curve without specifying the punctual values of returns. However, our method clearly outperforms this competitor, and also the Buy-and-Hold strategy, which nevertheless highlights a very strong market trend in this period, hard to outrun.

Overall, both in relation to the non-RL ensemble techniques we used as benchmarks and to the competitors present in literature, in the periods and markets they covered, our approach showed to be competitive in performance and robust in results. As outlined in next Section 6, further studies are needed to investigate the response of our method in markets with different behaviours and characteristics, as well as for the management of compound portfolios.

6 Conclusions and Future Work

Machine learning techniques cover a crucial role in many financial contexts, since they are used to evaluate the potential risks of investments, thus reducing the losses due to unreliable strategies. In the stock market context, such benefits are explored to the extreme by considering a novel fusion of several classifiers that usually exhibit low correlation, making such predictors complementary, to better classify the data when used together. However, in chaotic environments like the stock market, only using raw information from market prices and single ensemble techniques may lead to acceptable, but not optimal, trading performance.

This paper proposes a step forward in efficient stock trading with ensembles by presenting an approach that uses two well known and efficient machine learning approaches, namely deep learning and deep reinforcement learning, in a three layer fashion. The proposed method then exploits several ensembling

steps to provide its final intra-day trading strategy: firstly, we stack hundreds of deep learning decisions, provided by a large number of CNNs trained with historical market data encoded into GAF images, and use them as input for a reinforcement metalearner classifier; and, lastly, we ensemble the decisions of several training iterations of the metalearner.

From our experimental results, we observe that: (i) the metalearner leads to better trading results and less overfitting when we preliminary explore the training parameters and adopt the most promising ones; (ii) compared to well-performing non-RL ensemble baselines, our approach showed a final return improvement of 28.78% when compared to our best benchmark (which, notably, has still very good performances in the considered period), by yielding returns 5.70 higher than the risk assumed, and outperforming the market not only in the crisis scenario of the 2018, but also in the following phase of 2019 in which the market showed a very solid trend; and, finally, (iii) our approach provides the best performances, when tested against the strong methods proposed by Sezer et al. [32] and Calvi et al. [5], in two different real-world trading scenarios, and when considering several distinct periods and markets.

According to the previous considerations and findings, several future works can be inspired by the current research. First of all, we aim to extend the network structure in order to reduce the effects of overfitting: for instance, we think that adding recurrent layers – such as Long Short Term Memories networks – in the proposed pipeline can potentially help improving the results. Additionally, we also aim to evaluate the impact of optimization algorithms [1,14] when training both our first and second layers of our approach. And, finally, we would focus our future investigations on the fusion of different metalearner optimizers/parameters, which may increase the number of experts in the final ensembling methodology and, therefore, lead to a more robust and stable trading strategy.

References

1. Ahmadian, S., Khanteymoori, A.R.: Training back propagation neural networks using asexual reproduction optimization. In: Conference on Information and Knowledge Technology (IKT), pp. 1–6 (2015)
2. An, N., Ding, H., Yang, J., Au, R., Ang, T.F.: Deep ensemble learning for alzheimer's disease classification. Journal of Biomedical Informatics **105**, 103411 (2020). DOI https://doi.org/10.1016/j.jbi.2020.103411
3. Asad, M.: Optimized stock market prediction using ensemble learning. In: 2015 9th International Conference on Application of Information and Communication Technologies (AICT), pp. 263–268 (2015)
4. Barra, S., Carta, S.M., Corriga, A., Podda, A.S., Recupero, D.R.: Deep learning and time series-to-image encoding for financial forecasting. IEEE/CAA Journal of Automatica Sinica **7**(3), 683–692 (2020). DOI 10.1109/JAS.2020.1003132
5. Calvi, G.G., Lucic, V., Mandic, D.P.: Support tensor machine for financial forecasting. In: ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 8152–8156 (2019). DOI 10.1109/ICASSP.2019.8683383
6. Chun, S.H., Park, Y.J.: Dynamic adaptive ensemble case-based reasoning: application to stock market prediction. Expert Systems with Applications **28**(3), 435 – 443 (2005)

7. Fenghua, W., Jihong, X., Zhifang, H., Xu, G.: Stock price prediction based on ssa and svm. Procedia Computer Science **31**, 625 – 631 (2014). 2nd International Conference on Information Technology and Quantitative Management, ITQM 2014
8. Fu, T.c., Lee, K.k., Sze, D., Chung, F.l., Ng, C.m.: Discovering the correlation between stock time series and financial news. In: Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01, WI-IAT '08, p. 880–883. IEEE Computer Society, USA (2008). DOI 10.1109/WIIAT.2008.228. URL https://doi.org/10.1109/WIIAT.2008.228
9. Gao, X., Hongkong, S., Chan, L.: An algorithm for trading and portfolio management using q-learning and sharpe ratio maximization. In: International Conference on Neural Information Processing, pp. 832–837 (2000)
10. Gyamerah, S.A., Ngare, P., Ikpe, D.: On stock market movement prediction via stacking ensemble learning method. In: IEEE Conference on Computational Intelligence for Financial Engineering Economics (CIFEr), pp. 1–8 (2019)
11. Han, J., Kamber, M., Pei, J.: Data Transformation and Data Discretization, chap. 3, pp. 111–118. Elsevier (2011)
12. Hasselt, H.V.: Double q-learning. In: J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel, A. Culotta (eds.) Advances in Neural Information Processing Systems 23, pp. 2613–2621. Curran Associates, Inc. (2010)
13. Hasselt, H.V.: Double q-learning. In: J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel, A. Culotta (eds.) Advances in Neural Information Processing Systems 23, pp. 2613–2621. Curran Associates, Inc. (2010)
14. Jalali, S.M.J., Ahmadian, S., Kebria, P.M., Khosravi, A., Lim, C.P., Nahavandi, S.: Evolving artificial neural networks using butterfly optimization algorithm for data classification. In: Neural Information Processing, pp. 596–607. Springer International Publishing, Cham (2019)
15. Jalali, S.M.J., Ahmadian, S., Khosravi, A., Mirjalili, S., Mahmoudi, M.R., Nahavandi, S.: Neuroevolution-based autonomous robot navigation: A comparative study. Cognitive Systems Research **62**, 35 – 43 (2020)
16. Kamijo, K.i., Tanigawa, T.: Stock price pattern recognition-a recurrent neural network approach. In: 1990 IJCNN International Joint Conference on Neural Networks, pp. 215–221. IEEE (1990)
17. Kang, Q., Zhou, H., Kang, Y.: An asynchronous advantage actor-critic reinforcement learning method for stock selection and portfolio management. In: Proceedings of the 2nd International Conference on Big Data Research, ICBDR 2018, p. 141–145. Association for Computing Machinery, New York, NY, USA (2018). DOI 10.1145/3291801.3291831. URL https://doi.org/10.1145/3291801.3291831
18. Khairi, T.W.A., Zaki, R.M., Mahmood, W.A.: Stock price prediction using technical, fundamental and news based approach. In: 2019 2nd Scientific Conference of Computer Sciences (SCCS), pp. 177–181 (2019)
19. Khan, W., Ghazanfar, M.A., Azam, M.A., Karami, A., Alyoubi, K.H., Alfakeeh, A.S.: Stock market prediction using machine learning classifiers and social media, news. Journal of Ambient Intelligence and Humanized Computing (2020)
20. Kim, T., Kim, H.Y.: Forecasting stock prices with a feature fusion lstm-cnn model using different representations of the same data. PLOS ONE **14**(2), 1–23 (2019). DOI 10.1371/journal.pone.0212320
21. Kimoto, T., Asakawa, K., Yoda, M., Takeoka, M.: Stock market prediction system with modular neural networks. In: 1990 IJCNN international joint conference on neural networks, pp. 1–6. IEEE (1990)
22. Lee, C.H., Park, K.C.: Prediction of monthly transition of the composition stock price index using recurrent back-propagation. In: Artificial neural networks, pp. 1629–1632. Elsevier (1992)
23. Lee, J.W., Park, J., Jangmin, O., Lee, J., Hong, E.: A multiagent approach to q-learning for daily stock trading. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans **37**, 864–877 (2007)
24. Lei, K., Zhang, B., Li, Y., Yang, M., Shen, Y.: Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading. Expert Systems with Applications **140**, 112872 (2020). DOI https://doi.org/

10.1016/j.eswa.2019.112872. URL http://www.sciencedirect.com/science/article/pii/S0957417419305822
25. Lin, Y., Huang, T., Chung, W., Ueng, Y.: Forecasting fluctuations in the financial index using a recurrent neural network based on price features. IEEE Transactions on Emerging Topics in Computational Intelligence pp. 1–12 (2020)
26. Magdon-Ismail, M., Atiya, A.F.: Maximum drawdown. Risk Magazine **17**(10), 99–102 (2004)
27. Mihatsch, O., Neuneier, R.: Risk-sensitive reinforcement learning. In: Advances in Neural Information Processing Systems, pp. 1031–1037. MIT Press (1999)
28. Moody, J., Wu, L., Liao, Y., Saffell, M.: Performance functions and reinforcement learning for trading systems and portfolios. Journal of Forecasting **17**(5-6), 441–470 (1998)
29. Patil, P., Wu, C.S.M., Potika, K., Orang, M.: Stock market prediction using ensemble of graph theory, machine learning and deep learning models. In: Proceedings of the 3rd International Conference on Software Engineering and Information Management, ICSIM '20, p. 85–92. Association for Computing Machinery, New York, NY, USA (2020)
30. Plappert, M.: keras-rl. https://github.com/keras-rl/keras-rl (2016)
31. Puterman, M.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons (2014)
32. Sezer, O.B., Ozbayoglu, A.M.: Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach. Applied Soft Computing **70**, 525 – 538 (2018). DOI https://doi.org/10.1016/j.asoc.2018.04.024. URL http://www.sciencedirect.com/science/article/pii/S1568494618302151
33. Si, W., Li, J., Ding, P., Rao, R.: A multi-objective deep reinforcement learning approach for stock index future's intraday trading. In: International Symposium on Computational Intelligence and Design (ISCID), vol. 2, pp. 431–436 (2017)
34. Sim, H.S., Kim, H.I., Ahn, J.J.: Is deep learning for image recognition applicable to stock market prediction? Complexity (2019)
35. Sun, T., Wang, J., Ni, J., Cao, Y., Liu, B.: Predicting futures market movement using deep neural networks. In: 18th IEEE International Conference On Machine Learning And Applications (ICMLA), pp. 118–125 (2019)
36. Surton, R., Barto, A.: Reinforcement Learning: an introduction, vol. 1. MIT press Cambridge (1998)
37. Tan, T.Z., Quek, C., Ng, G.S.: Brain-inspired genetic complementary learning for stock market prediction. In: 2005 IEEE Congress on Evolutionary Computation, vol. 3, pp. 2653–2660. IEEE (2005)
38. Tan, Z., Yan, Z., Zhu, G.: Stock selection with random forest: An exploitation of excess return in the chinese stock market. Heliyon **5**(8), e02310 (2019). DOI https://doi.org/10.1016/j.heliyon.2019.e02310. URL http://www.sciencedirect.com/science/article/pii/S2405844019359705
39. Wang, Z., Oates, T.: Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. In: Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence (2015)
40. Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., Freitas, N.: Dueling network architectures for deep reinforcement learning. In: International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 48, pp. 1995–2003. PMLR, New York, New York, USA (2016)
41. Wolpert, D.H.: Stacked generalization. Neural Networks **5**, 241–259 (1992)
42. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. Trans. Evol. Comp **1**(1), 67–82 (1997). DOI 10.1109/4235.585893. URL https://doi.org/10.1109/4235.585893
43. Wu, Y., Mao, J., Li, W.: Predication of futures market by using boosting algorithm. In: International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), pp. 1–4 (2018)
44. Ye, C., Ma, H., Zhang, X., Zhang, K., You, S.: Survival-oriented reinforcement learning model: An effcient and robust deep reinforcement learning algorithm for autonomous driving problem. In: Y. Zhao, X. Kong, D. Taubman (eds.) Image and Graphics, pp. 417–429. Springer International Publishing, Cham (2017)

45. Zhang, Y., Wu, L.: Stock market prediction of s&p 500 via combination of improved bco approach and bp neural network. Expert systems with applications **36**(5), 8849–8854 (2009)
46. Zhou, Z., Gao, M., Liu, Q., Xiao, H.: Forecasting stock price movements with multiple data sources: Evidence from stock market in china. Physica A: Statistical Mechanics and its Applications **542**, 123389 (2020). DOI https://doi.org/10.1016/j.physa.2019.123389. URL http://www.sciencedirect.com/science/article/pii/S0378437119318941