Price Trailing for Financial Trading using Deep Reinforcement Learning

Avraam Tsantekidis, Nikolaos Passalis, Anastasia-Sotiria Toufa, Konstantinos Saitas-Zarkias, Stergios Chairistanidis, and Anastasios Tefas

***Abstract*—Machine learning methods have recently seen a growing number of applications in Financial Trading. Being able to automatically extract patterns from past price data and consistently apply them in the future has been the focus of many quantitative trading applications. However, developing machine learning-based methods for financial trading is not straightforward, requiring carefully designed targets/rewards, hyperparameter fine-tuning, etc. Furthermore, most of the existing methods are unable to effectively exploit the information available across various financial instruments. In this work, we propose a Deep Reinforcement Learning-based approach that ensures consistent rewards are provided to the trading agent, mitigating the noisy nature of Profit-and-Loss rewards that are usually used. To this end, we employ a novel price trailing-based reward shaping approach, significantly improving the performance of the agent in terms of profit, sharpe ratio and maximum drawdown. Furthermore, we carefully designed a data preprocessing method that allows for training the agent on different FOREX currency pairs, providing a way for developing market-wide RL agents and allowing, at the same time, to exploit more powerful recurrent Deep Learning models without the risk of overfitting. The ability of the proposed methods to improve various performance metrics is demonstrated using a challenging large scale dataset, containing 28 instruments, provided by Speedlab AG.**

***Index Terms*—Deep Reinforcement Learning, Market-wide trading, Price-trailing.**

I. Introduction

In recent years financial markets have been geared towards an ever increasing automation of trading by quantitative algorithms and smart agents. For a long time quantitative human traders have been getting "phased out" due to their inconsistent behaviour and, consequently, performance. A 2015 study reported that the financial products with the highest traded volume has garnered the biggest presence of automated trading agents [1]. Those products constitute the Foreign Exchange (FOREX) markets with 80% of the trading activity coming from automated trading.

One approach to automated trading are the rule-based algorithmic approaches, such as the technical analysis of the price time-series aiming to detect plausible signals that entail certain market movements, thus triggering trading actions that will yield profit if such movements occur. One step further are the machine learning techniques that automatically determine the patterns that lead to predictable market movement. Such techniques require the construction of supervised labels, from the price time-series, that describe the direction of the future price movement [2], [3]. Noise-free labels unfortunately can be difficult to construct, since the extreme and unpredictable nature of financial markets do not allow for calculating a single "hard" threshold to determine whether a price movement is significant or not.

The need for supervised labels can be alleviated by the use of Reinforcement Learning (RL). In RL an agent is allowed to interact with the environment and receives rewards or punishments. In the financial trading setting, the agent decides what trading action to take and is rewarded or punished according to its trading performance. In this work, trading performance is determined using an environment that simulates the financial markets and the profits or losses accumulated as a result of the actions taken by a trading agent. There is no need for supervised labels since RL can take into account the magnitude of the rewards instead of solely considering the direction of each price movement. This benefit over the supervised learning methods has lead to an increasing number of works that attempt to exploit RL for various financial trading tasks [4], [5], [6].

RL has seen great success in recent years with the introduction of various Deep Learning-based methods, such as Deep Policy Gradients [7], Deep Q-learning [8], [9], and Proximal Policy Optimization [10], that allow for developing powerful agents that are capable of directly learning how to interact with the environment. Although the benefits of such advances have been clearly shown, RL still exhibits inconsistent behaviour across many tasks [11]. This inconsistency can be exaggerated when RL is applied to the noisy task of trading, where the rewards are tightly connected to the obtained Profit and Loss (PnL) metric which is also noisy. The PnL of an agent can be evaluated by simulating the accumulated profits or losses from the chosen action of the agent along with the cost incurred from the trading commission. Although RL mitigates

Avraam Tsantekidis, Nikolaos Passalis, Anastasia-Sotiria Toufa, Konstantinos Saitas-Zarkias*, and Anastasios Tefas are (*were) with the School of Informatics, Aristotle University of Thessaloniki, Thessaloniki 54124, Greece. Stergios Chairistanidis is with the Research Department at Speedlab AG. Konstantinos Saitas-Zarkias is now with the KTH Royal Institute of Technology, Sweden (research conducted while at the Aristotle University of Thessaloniki). E-mail: avraamt@csd.auth.gr, passalis@csd.auth.gr, toufaanast@csd.auth.gr, kosz@kth.se, sc@speedlab.ag, tefas@csd.auth.gr. This research has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH - CREATE - INNOVATE (project code: T2EDK-02094). Avraam Tsantekidis was solely funded by a scholarship from the State Scholarship Foundation (IKY) according to the "Strengthening Human Research Resources through the Pursuit of Doctoral Research" act, with resources from the "Human Resources Development, Education and Lifelong Learning 2014-2020" program. We kindly thank Speedlab AG for providing their expertise on the matter of FOREX trading and the comprehensive dataset of 28 FOREX currency pairs.

the difficulties of selecting suitable thresholds for supervised labels it still suffers from the noisy PnL rewards.

A concept that has assisted in solving hard problems with RL is reward shaping. Reward shaping attempts to more smoothly distribute the rewards along each training epoch, lessening the training difficulties associated with sparse and delayed rewards. For example, rewarding a robotic arm's proximity to an object that it intends to grip can significantly accelerate the training process, providing intermediate rewards related to the final goal, compared to only rewarding the final successful grip. As tasks get more complex, reward shaping can become more complex itself, while recent application have shown that tailoring it to the specific domain of its application can significantly improve the agents' performance [12], [13]. Shaping the reward could possibly help the RL agent better determine the best actions to take, but such shaping should not attempt to clone the behavior of supervising learning algorithms such as reintroducing the hard thresholds used for determining the direction of the price in trading application.

The current research corpus on Deep RL applied on financial trading, such as [4], usually employ models, such as Multilayer Perceptrons (MLPs). However, it has been demonstrated that in the supervised learning setting more complex models, like the Long Short-Term Memory Recurrent Neural Networks (LSTM) [14], [15], consistently outperform the simpler models that are often used in RL. Using models, such as LSTMs, that are capable of modeling the temporal behavior of the data can possibly allow RL to better exploit the temporal structure of financial time-series.

Also, a major difference between existing trading RL agents and actual traders, is that RL agents are trained to profitably trade single products in the financial markets, while human traders can adapt their trading methods to different products and conditions. One important problem encountered when training RL agents on multiple financial products simultaneously is that most of them have different distributions in terms of their price values. Thus the agent cannot extract useful patterns from one Deep RL policy and readily apply it on another when the need arises, without some careful normalization of the input. Being able to train across multiple pairs and track reoccurring patterns could possibly increase the performance and stability of the RL training procedure.

The main contribution of this work is a framework that allows for successfully training Deep RL that can overcome the limitations previously described. First, we developed and compare a Q-Learning based agent and a Policy Optimization based agent, both of which can better model the temporal behavior of financial prices by employing LSTM models, similar to those used for simpler classification-based problems [3]. However, as it will be demonstrated, directly using recurrent estimators was not straightforward, requiring the development of the appropriate techniques to avoid over-fitting the agent to the training data. In this work, this limitation was addressed by employing a *market-wide* training approach, that allowed us to mine useful information from various financial instruments. To this end, we also employed a stationary feature extraction approach that allows the Deep RL agent to effectively work using data that were generated from different distributions. Finally, a reward shaping method that provides more consistent rewards to the agent during its initial interactions with the environment was employed to mitigate the large variance of the rewards caused by the noisy nature of the PnL-based rewards, significantly improving the profitability of the learned trading policies. The developed methods were evaluated using a large-scale dataset that contains FOREX data from 28 different instruments collected by SpeedLab AG. This dataset contains data collected over a period of more than 8 years, providing reliable and exhaustive evaluations of Deep RL for trading.

The structure of this paper is as follows. In Section II we briefly present existing work on the subject of machine learning applied to trading and compare them with the proposed method. In Section III the proposed approach is introduced and explained in detail. In Section V the experimental results are presented and discussed. Finally, in Section VI, the conclusion of this work are drawn.

II. Related Work

In recent works for financial applications of Machine Learning, the most prevailing approach is the prediction of the price movement direction of various securities, commodities and assets. Works such as [16], [17], [18], [19], [20], [21] utilize Deep Learning models, such as Convolutional Neural Networks and LSTMs, to directly predict the attributes of the price movement in a supervised manner. The expectation of such techniques is that, by being able to predict where a price of an asset is headed (upwards or downwards), an investor can decide whether to buy or sell said asset, in order to profit. Although these approaches present useful results, the extraction of supervised labels from the market data requires exhaustive fine-tuning, which can lead to inconsistent behaviour. This is due to the unpredictable behaviour of the market that introduces noise to the extracted supervised labels. This can lead to worse prediction accuracy and thus worse or even negative performance to invested capital.

One way to remove the need of supervised labels is to follow a RL approach. In works such as [6], [22], [23], [24], the problem of profitable trading is defined in a RL framework, in which an agent is trained to make the most profitable decisions by intelligently placing trades. However, having the agent's reward wholly correlated with the actual profit of its executed trades, while at the same time employing powerful models, may end up overfitting the agent on noisy data.

In this work we improve upon existing RL for financial trading by developing a series of novel methods that can overcome many of the limitations described above, allowing for developing profitable Deep RL agents. First, we propose a Deep RL application for trading using an LSTM-based agent, in the FOREX markets by exploiting a *market-wide* training approach, i.e., training a single agent on multiple currencies. This is achieved by employing a stationary feature extraction scheme, allowing any currency pair to be used as the observable input of a singular agent. Furthermore, we propose a novel reward shaping method, which provides additional rewards that allows for reducing the variance and, as a result,

improving the stability of the learning process. One additional reward, called the "trailing reward", is obtained using an approach similar to those exploited by human traders, i.e., mentally trailing the price of an asset helping to estimate its momentum. To the best of our knowledge this is the first RL-based trading method that a) performs market-wide RL training in FOREX markets, b) employs stationary features in the context of Deep Learning to effectively extract the information contained in the price time-series generated from different distributions and c) uses an effective trailing-based reward shaping approach to improve the training process.

III. Methodology

In this section, the notation and prerequisites are introduced followed by the proposed preprocessing scheme applied to the financial data. Then, the proposed market-wide training method, the price trailing-based reward shaping, and the proposed recurrent agents are derived and discussed in detail.

A. Notation and Prerequisites

Reinforcement Learning (RL) consists of two basic components: a) an environment, and b) an agent interacting with said environment. In this case, the environment consists of a mechanism that when given past market data it can simulate a trading desk; expecting trading orders and presenting their resulting performance as time advances. The environment also provides observations of the market to the agent in the form of features which are presented in Section III-B. The observation of the environment along with the current position of the agent is the *state* of the environment and is denoted as s_t for the state of time-step t.

The agent in this context is given three choices on every time-step either to buy, sell or exit any position and submit no order. The position of the agent is denoted as δ_t and can take the values $\{-1, 0, 1\}$ for the positions of *short* (sell), *stay* out of the market and *long* (buy) respectively. Depending on the agents actions, a reward is received by the agent which is denoted by r_t.

RL has multiple avenues to train an agent to interact with the environment. In this work, we are using a Q-learning based approach, (i.e., our agent will estimate the Q-value of each action) and a policy gradient based approach (i.e. Proximal Policy Optimization). For the Q-learning approach an estimator $\mathbf{q}_t = f_\theta(s_t)$ is employed for estimating the Q-values $\mathbf{q}_t \in \mathbb{R}^3$, whereas for the policy gradient approach a policy $\pi_\theta(s_t)$ is used to calculate the probability of each action. In both cases θ is used to denote the trainable parameters of the estimator and $\mathbf{s}_t \in \mathbb{R}^{d \times T}$ denotes the state of the environment at time t. The dimensions of the state consist of the number of features d multiplied by the number of past time-steps T provided as observation to the agent by the environment. The predicted vector $\mathbf{q}_t$ consists of the three Q-values estimated by the agent for each of the three possible actions.

B. Financial Data Preprocessing

The financial data utilized in this work consist of the trading data between Foreign Exchange (FOREX) currencies, such as the EUR/USD trading pair. Since the raw trading data containing all the executed trades is exceptionally large, a subsampling method is utilized to create the so-called *Open-High-Low-Close* (*OHLC*) candlesticks or candles [25]. To construct these candles, all the available trade execution data is split into time windows of the desired length. Then for each batch of trades that fall into a window the following four values are extracted:

1) *Open* Price $p_o(t)$, i.e., the price of the first trade in the window,
2) *High* Price $p_h(t)$, i.e., the highest price a trade was executed in the window,
3) *Low* Price $p_l(t)$, i.e., the lowest price a trade was executed in the window, and
4) *Close* Price $p_c(t)$, i.e., the price of the last trade in the window.

An example of these candles with a subsampling window of 30 minutes is provided in the supplementary material for the EUR/USD trading pair.

The values of *OHLC* subsampling are execution prices of the traded assets and if they are observed independently of other time-steps, they do not provide actionable information. Using the sequence of *OHLC* values directly as input to a neural network model can also be problematic due to the stochastic drift of the price values [26]. To avoid such issues, a preprocessing step is applied to the *OHLC* values to produce more relevant features for the employed approach.

The features proposed in this work are inspired from technical analysis [27], such as the returns, the log returns and the distances of the current price to a moving average. These values are the components of complex quantitative strategies, which in their simplest form, utilize them in a rule-based setting. The following features are employed in this work:

1) $x_{t,1} = \dfrac{p_c(t) - p_c(t-1)}{p_c(t-1)}$,
2) $x_{t,2} = \dfrac{p_h(t) - p_h(t-1)}{p_h(t-1)}$,
3) $x_{t,3} = \dfrac{p_l(t) - p_l(t-1)}{p_l(t-1)}$,
4) $x_{t,4} = \dfrac{p_h(t) - p_c(t)}{p_c(t)}$,
5) $x_{t,5} = \dfrac{p_c(t) - p_l(t)}{p_c(t)}$.

The first feature, the percentage change of the close price, is hereby also referred to as the *return* $z_t = \frac{p_c(t) - p_c(t-1)}{p_c(t-1)}$. The rest of the constructed features also consist of relativity measures between prices through time. One of the major advantages of choosing these features is their normalizing nature. For every time-step t we define a feature vector $\mathbf{x}_t = [x_{t,1}, x_{t,2}, x_{t,3}, x_{t,4}, x_{t,5}]^T \in \mathbb{R}^5$ containing all the above mentioned features.

By not including the raw *OHLC* values in the observable feature, the learning process will have to emphasize on the temporal patterns exhibited by the price instead of the circumstantial correlation of a specific price value of an asset. One example of such a correlation observed in preliminary experiments was that whenever an agent observes the lowest prices of the dataset as the current price, the decision was always to buy, while when the highest available prices were

observed the agent always decided to sell. This is an unwanted correlation, since the top and bottom prices may not always be the currently known extrema of the price.

C. Profit Based Reward

We follow the classical RL approach for defining an environment which an agent will interact with and receive rewards from. The environment has a time parameter t defining the moment in time that is being simulated. The market events that happened before time t are available to the agent interacting with the environment. The environment moves forward in time with steps of size k and after each step, it rewards or punishes the agent depending on the results of the selected action, while also providing a new observation with the newly available market data.

In previous applications, such as [4], [28], a common approach for applying reinforcement learning for trading is to reward the agent depending on the profit of the positions taken. This approach is also tested independently in this work with the construction of a reward that is based on the trading *Profit and Loss* (*PnL*) of the agent's decisions.

In a trading environment, an agent's goal is to select a market position based on the available information. The market position may either be to buy the traded asset, referred to as "going long" or to sell the traded asset, referred to as "going short". An agent may also choose to not participate in the trading activity. We define the profit based reward as:

$$r_t^{(\text{PnL})} = \begin{cases} z_t, & \text{if agent is long} \\ -z_t, & \text{if agent is short} \\ 0, & \text{if agent has no position} \end{cases}, \tag{1}$$

where z_t is the percentage return defined in Section III-B.

We also define the agent's position in the market as $\delta_t \in \{long, neutral, short\} = \{1, 0, -1\}$. This can simplify the reward of Eq. 1 to $r_t^{(\text{PnL})} = \delta_t \cdot z_t$. Furthermore, in an actual trading environment the agent also pays a cost to change position. This cost is called *commission*. The commission is simulated as an extra reward component payed by a trading agent when a position is opened or reversed. The related term of the reward is defined as:

$$r_t^{(\text{Fee})} = -c \cdot |\delta_t - \delta_{t-1}|, \tag{2}$$

where c is the commission fee.

D. Reward Shaping using Price Trailing

As already discussed in Section III-C, PnL-based rewards alone are extremely noisy, which increases the variance of the returns, reducing the effectiveness of the learning process. To overcome this limitation, in this paper we propose using an appropriately designed reward shaping technique that can reduce said variance, increasing the stability of the learning process, as well as the profitability of the learned trading policies. The proposed reward shaping method is inspired by the way human traders often mentally visualize the process of predicting the price trend of an asset. That is, instead of trying to predict the most appropriate trading action, we propose training an agent that should learn how to position itself in the market in order to closely follow the price of an asset. This process is called *price trailing*, since it resembles the process of driving a vehicle (current price estimation) and trying to closely follow the trajectory of the price (center of the road). Therefore, the agent must appropriately control its position on the road defined by the price of time series in order to avoid crashing, i.e., driving out of the road.

It is worth noting that keeping the agent within the boundaries of a road at all times cannot be achieved by simply aiming to stay as close to the center of the road as possible. Therefore, to optimally navigate a route the agent must take into consideration the layout of the road ahead and consider the best trajectory to enter and exit sharp corners in the most efficient way. This may give rise to situations where the best trajectory guides the agent from parts of the roadway that is far from the center, which smooths the manoeuvres needed to steer around a corner. Indeed, the price exhibits trajectories that traders attempt "steer around" in the most efficient way in order to maximize their profit. The prices move sharply and acting on every market movement may prove costly in terms of commission. Therefore, a trading agent must learn how to navigate in a smooth manner through the "road" defined by the price, while incurring the least amount of commission cost as possible, and without "crashing" into the metaphorical barriers, which would be considered a loss for the agent.

Utilizing this connection to manoeuvring a vehicle, a novel approach is introduced for training a Deep RL agent to make financial decisions. The proposed trailing-based reward shaping scheme allows for training agents that handle the intrinsic noise of PnL-based rewards better, significantly improving their behavior, as it is experimentally demonstrated in Section V. The agent is assigned its own price value $p_a(t)$ for each time-step, which it can control by upward or downward increments. The agent's price is compared with a target price $p_\tau(t)$ which acts as the mid-point of the trajectory the agent is rewarded the most to follow. In its simplest form the target price can be the close price $p_\tau(t) = p_c(t)$. The agent can control its assigned price $p_a(t)$ using upward or downwards steps as its actions. We also define the "barriers", as an upper $p_{um}(t)$ and a lower margin $p_{lm}(t)$ to the target price that are calculated as:

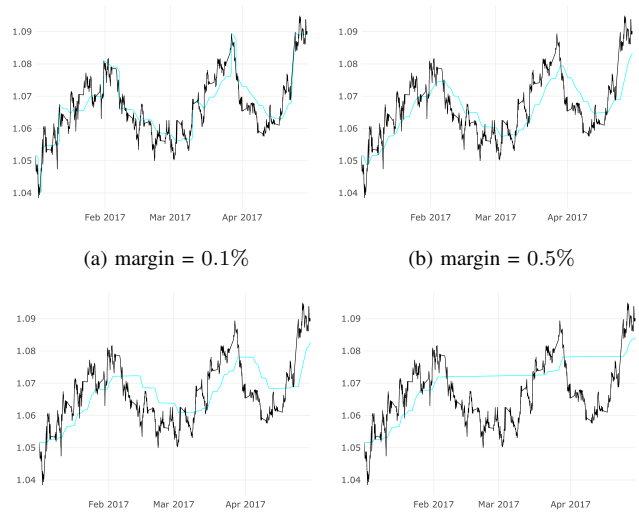
$$p_{um}(t) = p_\tau(t) \cdot (1 + m), \tag{3}$$

$$p_{lm}(t) = p_\tau(t) \cdot (1 - m), \tag{4}$$

where m is the margin fraction parameter used to determine the distance of the margin to the current target price. The agent's goal is to keep the *agent price* $p_a(t)$ close to the target price which in this case is the *close* price. The proposed trailing reward can be then defined as:

$$r_t^{(\text{Trail})} = 1 - \frac{|p_a(t) - p_c(t)|}{m p_c(t)}. \tag{5}$$

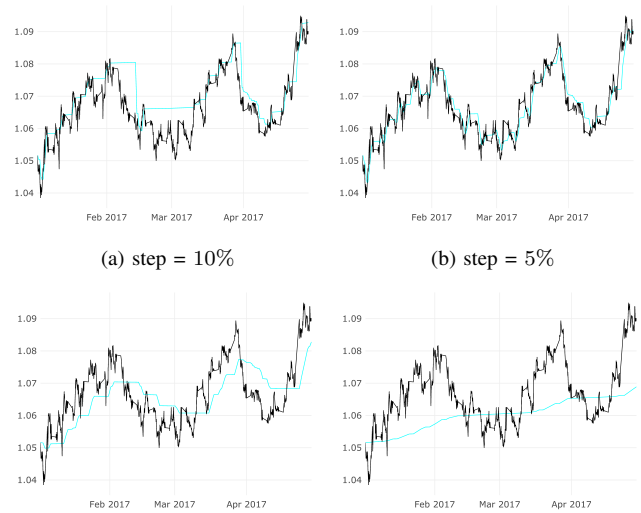
The reward is positive while the agent's position $p_a(t)$ is within the margins defined by m, obtaining a maximum value of 1, when $p_a(t) = p_c(t)$. If the agent price crosses the margin bounds either above or below, the reward becomes negative.



(a) margin = 0.1% (b) margin = 0.5%

(c) margin = 1% (d) margin = 3%

Fig. 1. Effect of different margin width values to the trailing path the agent adheres. Black line is the actual price and ciel line is the price trailing achieved by the agent.



(a) step = 10% (b) step = 5%

(c) step = 1% (d) step = 0.1%

Fig. 2. Effects of different step size values to the trailing path the agent adheres. Black line is the actual price and ciel line is the price trailing achieved by the agent.

The agent controls its position between the margins either when selecting the action δ_t that was described in Section III-A or through a separate action. The agent can chose to move upwards towards the upper margin $p_{um}(t)$, or downwards towards the lower margin $p_{lm}(t)$. To control its momentum we introduce a *step value* υ that controls the stride size of the agent's price and it is defined as the percentage of the way to the upper or lower margin the agent is moved. The agent price changes according to the equation:

$$p_a(t+1) \leftarrow \begin{cases} p_a(t) + \upsilon * |p_a(t) - p_{um}(t)|, \\ \text{if agent selects up action} \\ \\ p_a(t) - \upsilon * |p_a(t) - p_{lm}(t)|, \\ \text{if agent selects down action} \\ \\ p_a(t), \\ \text{if agent selects stay action} \end{cases} \tag{6}$$

Different variations of the margin m and step value υ are observed in the results of Figures 1, 2. As the margin percentage increases, the agent is less compelled to stay close to the price, even going against its current direction. The larger margins allow the agent to receive some reward, while in the smaller margin yields negative rewards when the agent price strays away from the margin boundaries. Changing the step percentage on the other hand leads to an inability to keep up with the price changes for the small step values, while larger values yield a less frequent movements behaviour.

Reward Normalization: The trailing and PnL rewards that have been described until now have vastly different scales and if left without normalization, the trailing reward will overpower the PnL reward, rendering the agent indifferent about it. Another problem is the PnL reward's statistic lie on a very small scale, which would slow down the training of the employed RL estimators. To remedy this we propose a normalization scheme to bring all the aforementioned rewards to a comparable scale. The PnL reward depends on the percentage returns between each consecutive bar, so the mean $\mu_{\mathbf{z}}$, mean of absolute values $\mu_{|\mathbf{z}|}$ and standard deviation $\sigma_{\mathbf{z}}$ of all the percentage returns are calculated. Then the normalized rewards $r^{(\text{PnL})}$, $r^{(\text{Fee})}$ and $r^{(\text{Trail})}$ are redefined as follow:

$$r^{(\text{PnL})} \leftarrow \frac{r^{(\text{PnL})}}{\sigma_{\mathbf{z}}} \tag{7}$$

$$r^{(\text{Fee})} \leftarrow \frac{r^{(\text{Fee})}}{\sigma_{\mathbf{z}}} \tag{8}$$

$$r^{(\text{Trail})} \leftarrow \frac{r^{(\text{Trail})} \cdot \mu_{|\mathbf{z}|}}{\sigma_{\mathbf{z}}} \tag{9}$$

The $r^{(\text{PnL})}$ and $r^{(\text{Fee})}$ are simply divided by the standard deviation of returns. We do not shift them by their mean as is usual with standardization, since the mean return is already very close to zero, and shifting it could potential introduce noise. The $r^{(\text{Trail})}$ up to this point, as defined in Eq. 5, could receive values up to a maximum of 1. By multiplying $r^{(\text{Trail})}$ with $\frac{\mu_{|\mathbf{z}|}}{\sigma_{\mathbf{z}}}$ we bind the maximum trailing reward to be reasonably close to the $r^{(\text{PnL})}$ and $r^{(\text{Fee})}$ rewards.

Combining Rewards: Since the aforementioned normalized rewards have compatible scales with each other, they can be combined to train a single agent, assuming the possible actions of said agent are defined by δ_t. In this case, the total combined reward is defined as:

$$r_t^{(\text{Total})} = \alpha_{\text{trail}} \cdot r_t^{(\text{Trail})} + \alpha_{\text{pnl}} \cdot r_t^{(\text{PnL})} + \alpha_{\text{fee}} \cdot r_t^{(\text{Fee})}, \tag{10}$$

where α_{trail}, α_{pnl} and α_{fee} are the parameters that can be used to adjust the reward composition and change the agents behaviour. For example, when increasing the parameter α_{fee}, one can expect the agent to change position less frequently to avoid accumulating commission punishments. To this end we formulated a set of experiments, as demonstrated in Section V, to test our hypothesis that combing the aforementioned trailing-based reward with direct PnL reward can act as a strong regularizer, improving performance when compared with the plain PnL reward.

IV. Reinforcement Learning Methods

The proposed reward shaping method was evaluated using two different Reinforcement Learning approaches. First a value based approach was employed, namely Double Deep Q-learning [9], and second a policy based approach, namely Proximal Policy Optimization [10]. In both cases the proposed approach improved the resulting profit generated by the trained agent.

A. Double Deep Q-learning

In Q-learning a value function calculates the potential return of each action available to the agent. This is known as the Q-value. During the training an epsilon-greedy [29] policy is followed allowing random actions to be taken in order to explore the environment. After the policy is optimized, the epsilon greedy exploration is deactivated and the agent always follows the action offering the highest potential return.

In its simplest form Q-learning stores and updates a matrix of Q-values for all the combinations of states $\mathbf{S}$ and actions $\mathbf{A}$. This matrix is called the Q-table and is defined as:

$$\mathbf{Q} : \mathbf{S} \times \mathbf{A} \rightarrow R^{|\mathbf{S}| \times |\mathbf{A}|} \tag{11}$$

In the modern approaches of Q-learning a neural network is trained to predict the value of each potential state-action combination removing the need of computing and storing a excessively large matrix, while also allowing for exploiting information about the environment from the states and actions. This leads to the Q-value network predictions being accurate in action-state combinations that might have never been observed before. The training targets for the approximator are iteratively derived as :

$$Y(s_t, a_t) \leftarrow r_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta); \theta^-) \tag{12}$$

where $Y(s_t, a_t)$ is the target value, that the used approximator must correctly estimate. The notation θ and θ^- refers to the parameter sets for the Q network and the target network, respectively. The target network is part of the Double DQN methodology [9] where the selection of the action strategy is done by a separate model than the estimation of the value of current state. Let

$$\Delta_{Y,Q} \equiv Y(s_t, a_t) - Q(s_t, a_t)$$

for posterity. To train the network we employ the differentiable Huber Loss:

$$L(s_t, a_t) = \begin{cases} \frac{1}{2}(\Delta_{Y,Q})^2, \text{ for } |\Delta_{Y,Q}| < 1 \\ |\Delta_{Y,Q}| - \frac{1}{2}, \text{ otherwise} \end{cases} \tag{13}$$

which is frequently used when training deep reinforcement learning agents [30]. Experience replay can be utilized along with batching so that experiences can be stored and utilized over multiple optimization steps. The framework followed in this work is the Double Deep Q-learning (DDQN) approach [9], which improves upon the simpler DQN by avoiding the overestimation of action values in noisy environments.

For this work, since we are dealing with time-series price data, we chose to apply an LSTM as the Q-value approximator. The proposed model architecture is presented in Figure 4. The network receives two separate inputs: one from the state s_t, which are the features described in Section III-B for the past n time-steps and the market position of the agent from the previous time-step. The LSTM outputs of its last time-step are fed through a fully connected layer with 32 neurons, which in turn has its activations concatenated with a one-hot vector containing the position of the agent on the previous time-step. The model from that point on has two fully connected layers with 64 neurons each that output the three Q-values.

B. Proximal Policy Optimization

Another approach to optimising reinforcement learning agents is via the use of Policy Gradients. Policy Gradient methods attempt to directly train the policy of an agent (i.e. the action an agent selects directly) rather that the estimation of the action-value like in the Q-learning approach. The objective used for policy gradient methods usually take the form of:

$$J(\theta) = \hat{\mathbb{E}}_t \Big[\pi_\theta(a|s) \hat{A}^\pi(s, a) \Big] \tag{14}$$

where $\pi_\theta(s|a)$ is the probability distribution of policy π parametrized by θ to select action a when in state s. The advantage estimation is denoted as $\hat{A}$ measuring the improvement in rewards received when selecting action a in state s compared to some baseline reward.

Proximal Policy Optimization (PPO) approach is one of the most promising approaches in this category. By using a clipping mechanism on its objective, PPO attempts to have a more stable approach to exploration of its environment, limiting parameter updates for the unexplored parts of its environment. The policy gradient objective from Eq. 14 is modified as to include the ratio of action probabilities:

$$J(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(\alpha_t|s_t)}{\pi_{\theta_{\text{old}}}(\alpha_t|s_t)} \hat{A}_t \right] \tag{15}$$

where $\pi_{\theta_{\text{old}}}$ is the probabilities over the actions with the old parameterization θ of the policy π. Then to ensure smaller

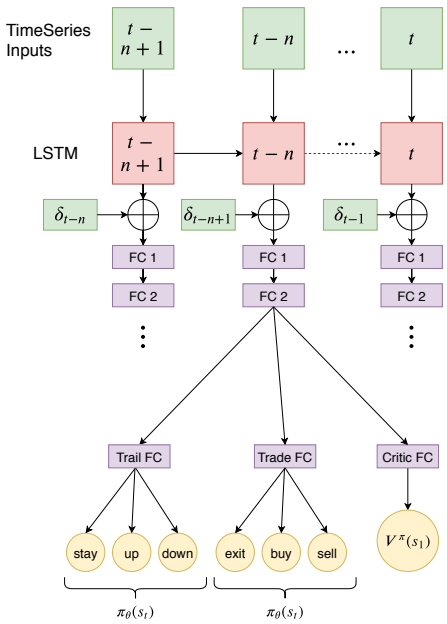


Fig. 3. Model architecture used with the PPO approach

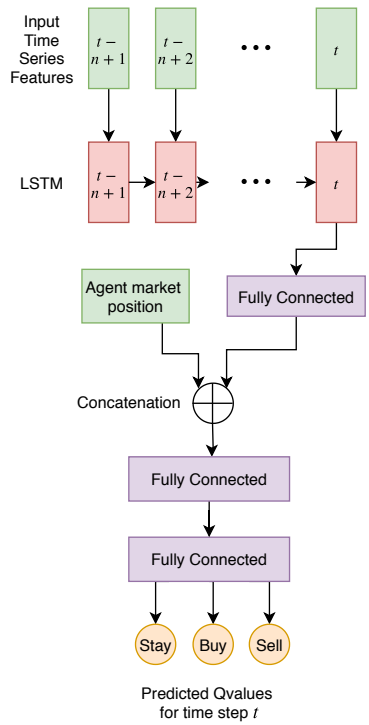


Fig. 4. Model architecture for predicting Q-values of each action

steps while exploring areas that produce rewards for the agent the objective is reformulated as:

$$J^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \Bigg[\min \Bigg(\frac{\pi_\theta(\alpha_t|s_t)}{\pi_{\theta_{\text{old}}}(\alpha_t|s_t)} \hat{A}_t, \\ \text{clip}(\frac{\pi_\theta(\alpha_t|s_t)}{\pi_{\theta_{\text{old}}}(\alpha_t|s_t)}, 1-\epsilon, 1+\epsilon)\hat{A}_t \Bigg) \Bigg] \tag{16}$$

where ϵ is a small value that dictates the maximum ratio of action probabilities change that can be rewarded with a positive advantage, while leaving the negative advantage values to affect the objective without clipping it. The advantage is calculated based on an estimation of the state value predicted by the model. The model is trained to predict a state value that is based on the rewards achieved throughout its trajectory propagating from the end of the trajectory to the beginning with a hyperbolic discount. The loss that is used to train the state value predictor is:

$$J^{\text{Value}} = L_H(r_t + \frac{1}{1+\gamma} V^\pi(s_{t+1}) - V^\pi(s_t)) \tag{17}$$

where $V^\pi(s_t)$ is the state value estimation given policy π on state s_t. This loss is proposed in [31] for estimating the advantage from the temporal difference residual. The final objective for the PPO method is obtained by summing the objectives in Equations 16 and 17 and using stochastic gradient descent to optimize the policy π_θ.

V. Experiments

The proposed methods are extensively evaluated in this section. First the employed dataset, evaluation setup and metrics are introduced. Then, the effects of the proposed reward shaping are evaluated on two different RL optimization methods, namely the Double Deep Q Network (DDQN) approach and the Proximal Policy Optimization (PPO) approach. In both approaches the reward shaping stays the same, while the agent

Fig. 5. Mean performance across 28 FOREX currency pairs of an agent trained with trailing reward vs. one without it. The y-axis represents Profit and Loss (PnL) in percentage to some variable investment, while the x-axis represents the date.

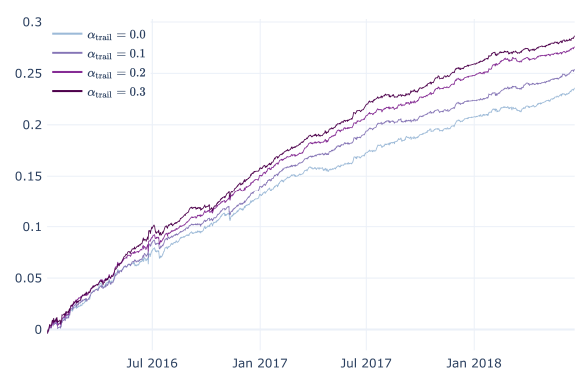


Fig. 6. Mean performance across 28 FOREX currency pairs of agents trained using Proximal Policy Optimization with different trailing values. The y-axis represents Profit and Loss (PnL) in percentage to some variable investment, while the x-axis represents the date.

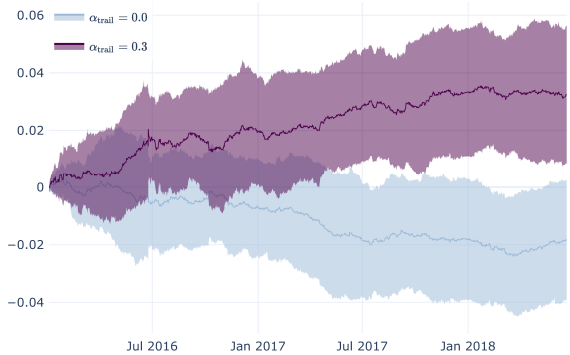


Fig. 7. Relative Mean PnL with one standard deviation continuous error bars. To allow for a more granular comparison between the relative performance of the presented experiments, we subtracted the mean PnL (calculated using all the PPO experiments) from the reported PnL for each experiment.

trailing controls are evaluated in a distinct way. This is a good indication that reward shaping approach has versatility and benefits the optimization of RL trading agents across different methods and implementations.

A. Dataset, Evaluation Setup and Metrics

The proposed method was evaluated using a financial dataset that contains 28 different instrument combinations with currencies such as Euro, Dollar, British pounds and Canadian dollar among others. The dataset contains minute price candles starting from 2009 to mid-2018 that were collected by SpeedLab AG. To the best of our knowledge, this is the largest financial dataset used for evaluating Deep RL algorithms and allows for more reliable comparisons between different trading algorithms. It is worth noting that an extended version of the same dataset is also internally used by SpeedLab AG to evaluate the effect of various trading algorithms, allowing us to provide one of the most exhaustive and reliable evaluations of Deep Learning-based trading systems that are provided in the literature.

To utilize the dataset, the minute price candles are resampled to hour candles, as explained in Section III-B. The dataset was split into a training set and a test set, with the training set ranging from the start of each instrument timeline (about 2009) up to 01/01/2017, and the test set continuing from there up to 01/06/2018.

The target price $p_\tau(t)$ is set to be the average of close prices of the future five time-steps, which allows the trailing target to be smoother. The trailing margin m is set to 2% while the training step υ is set to 1%.

The metrics used to evaluate our results consists of the Profit and Loss (PnL), which is calculated by simulating the effective profit or loss a trader would have accumulated had he executed the recommended actions of the agent. Another metric used is the annualized Sharpe ratio, which is the mean return divided by the standard deviation of returns thus penalizing strategies that have volatile behaviours. The final metric utilized is the max drawdown of profits, which is calculated as the maximum percentage difference of the highest peak in profits to the lowest following drop.

TABLE I
METRIC RESULTS COMPARING AGENTS TRAINED WITH AND WITHOUT TRAILING FACTOR α_{TRAIL} WHEN EVALUATED ON THE TEST DATA

	Without Trail	With Trail
Final PnL	3.3%	**6.2%**
Annualized Sharpe Ratio	0.753	**1.525**
Max Drawdown	2.6%	**1.6%**

B. DDQN Evaluation

For the evaluation of the DDQN approach we combine the agent's decision to buy or sell with the decision to move its price upwards or downwards.

The proposed LSTM, as described in Section IV-A, is used in order to evaluate its ability to reliably estimate the Q-values and achieve profitable trading policies. The agent is evaluated in a market-wide manner across all the available currency pairs. The episode length is set to be 600 steps. Note that limiting the number of steps per episode, instead of using the whole times-series at once, allows for more efficiently training the agent, as well gathering a more rich collection of experiences in less time, potentially accelerating the training process. Each agent during training runs for a total of 1 million steps. Episodes can abruptly end before reaching 600 steps when an agent gets stranded too far outside the margins we set, thus the total number of episodes is not consistent across all runs. The episodes are saved in a replay memory as described in [8]. For each episode a random point in time is chosen within the training period, ensuring it is at least 600 steps before the point where the train and test sets meet. A random currency pair is also chosen for every episode. The number LSTM hidden neurons is set to 1024 and L2 regularization is used for the LSTM weight matrix.

Given the features $\mathbf{x}_t$ of each time-step t described in Section III-B, we extract windows of size 16. For each time-step t in an episode we create the window $[\mathbf{x}_{t-16}, ..., \mathbf{x}_t]$. The LSTM processes the input window by sequentially observing each of the 16 time-steps and updating its internal hidden state on each step. A diagram of the window parsing process is presented in Figure 4. The discount γ used for the Q-value estimation based on future episode rewards is set to $\gamma = 0.99$.

Even though market-wide training leads to some performance improvements, the high variance of the PnL-based rewards constitute a significant obstacle when training Deep RL agents on such considerable amounts of the data. This problem can be addressed by using the proposed trailing-based reward shaping approach, as analytically described in Section III-D. Fig. 5 compares the average PnL obtained for the out-of-sample (test set) evaluation of two agents trained using the market-wide approach. For all the conducted experiments the reward functions (either with or without trailing) include both the PnL-based reward, as well as the commission fee, unless otherwise stated. Using the proposed reward shaping approach leads to significant improvements. Also, in Table I, the agent trained using trailing-based reward shaping also improves the draw down and Sharpe ratio over the agent based merely on PnL rewards.

To translate the models predictions to a specific position

TABLE II
METRIC RESULTS COMPARING AGENTS TRAINED USING PROXIMAL POLICY OPTIMIZATION

	Drawdown	Sharpe	PnL
Without trailing	$1.6\% \pm 0.6\%$	3.59 ± 0.37	$23.5\% \pm 2.0\%$
$\alpha_{\text{trail}} = 0.1$	$1.8\% \pm 0.5\%$	3.79 ± 0.59	$25.4\% \pm 3.2\%$
$\alpha_{\text{trail}} = 0.2$	$1.8\% \pm 0.4\%$	4.08 ± 0.72	$27.5\% \pm 4.1\%$
$\alpha_{\text{trail}} = 0.3$	$1.7\% \pm 0.5\%$	$\mathbf{4.20 \pm 0.43}$	$\mathbf{28.6\% \pm 2.4\%}$

which would specify the exact amount of an asset that would be bought or sold, the outputs representing the Q-values could be used. Dividing each Q-value by the sum of Q-values within a single output, an approximation of a confidence can be constructed and according to it a decision on the position allocation can be made.

C. PPO Evaluation

For the evaluation of the trailing reward application in a policy gradient method we separate the action of the agent to buy or sell from the action to move the agent's price p_a upwards or downwards into two separate action probability distributions as presented in Figure 3. The decoupling of the trade and trail actions while applying the reward shaping in the exact same manner is another valid approach to applying the proposed method. We carefully tuned the PPO baseline, including the trailing factor. We find that compared to the DDQN, lower values tend to work better. The optimal value of the trailing objective factor for the decoupled PPO application is 0.3 which leads to better test performance.

We conduct all experiments for this approach in the market-wide training mode, in the same manner as Section V-B. The agents used for this experiment consist of an LSTM with 128 hidden units which are connected to rest of the network components as shown in Figure 3. Each experiment is executed 10 times with different random seeds on each instance. The resulting PnLs presented for each trailing factor are averaged across its respective set of 10 experiments.

In Table II and in Figure 6 it is clearly demonstrated that the agents trained with a trailing reward factored by different values again perform better than the baseline. Optimizing RL agents using PPO has been shown to perform a better in many different tasks [10], which is also confirmed in the experiments conducted in this paper as well. Finally, to demonstrate the statistical significance of the obtained results, we also plotted the errors bars around the PnL for two agents trained with and without the proposed method. The results are plotted in Fig. 7 and confirm the significantly better behavior of proposed method.

VI. CONCLUSION

In this work, a deep reinforcement learning-based approach for training agents that are capable of trading profitably in the Foreign Exchange currency markets was presented. Several stationary features were combined in order to construct an environment observation that is compatible across multiple currencies, thus allowing an agent to be trained across the whole market of FOREX assets. Training in a market-wide manner allows for using powerful recurrent Deep Learning models — with reduced risk of overfitting, while significantly improving the results. The most remarkable contribution of this work is the introduction of a reward shaping scheme for mitigating the noisy nature of PnL-base rewards. The proposed approach uses an additional trailing reward, which encourages the agent to track the future price of a traded asset. Using extensive experiments on multiple currency pairs it was demonstrated that this can improve the performance significantly, increasing the final profit achieved by the agent, while also in some cases reduce the maximum drawdown.

A suggestion for future work in this area is to implement an attention mechanism, which has the potential to further increase the performance of RNNs. Another interesting idea is the design of more complex reward shaping methods that might include metrics extracted from the raw data itself such as the volatility of the price.

REFERENCES

[1] R. Haynes and J. S. Roberts, "Automated trading in futures markets," *CFTC White Paper*, 2015.

[2] M. Ballings, D. Van den Poel, N. Hespeels, and R. Gryp, "Evaluating multiple classifiers for stock price direction prediction," *Expert Systems with Applications*, vol. 42, no. 20, pp. 7046–7056, 2015.

[3] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Using deep learning to detect price change indications in financial markets," in *Proceedings of the European Signal Processing Conference*, 2017, pp. 2511–2515.

[4] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 3, pp. 653–664, 2017.

[5] M. A. Dempster and V. Leemans, "An automated fx trading system using adaptive reinforcement learning," *Expert Systems with Applications*, vol. 30, no. 3, pp. 543–552, 2006.

[6] J. Moody, L. Wu, Y. Liao, and M. Saffell, "Performance functions and reinforcement learning for trading systems and portfolios," *Journal of Forecasting*, vol. 17, no. 5-6, pp. 441–470, 1998.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[9] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning." in *AAAI*, vol. 2. Phoenix, AZ, 2016, p. 5.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[11] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," *arXiv preprint arXiv:1710.02298*, 2017.

[12] A. Hussein, E. Elyan, M. M. Gaber, and C. Jayne, "Deep reward shaping from demonstrations," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 510–517. [Online]. Available: https://academic.microsoft.com/paper/2596874484

[13] M. Grześ, "Reward shaping in episodic reinforcement learning," *adaptive agents and multi agents systems*, pp. 565–573, 2017. [Online]. Available: https://academic.microsoft.com/paper/2620974420

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[16] J. Patel, S. Shah, P. Thakkar, and K. Kotecha, "Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques," *Expert Systems with Applications*, vol. 42, no. 1, pp. 259–268, 2015.

[17] R. C. Cavalcante, R. C. Brasileiro, V. L. Souza, J. P. Nobrega, and A. L. Oliveira, "Computational intelligence and financial markets: A survey and future directions," *Expert Systems with Applications*, vol. 55, pp. 194–211, 2016.

[18] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Forecasting stock prices from the limit order book using convolutional neural networks," in *Proceedings of the IEEE Conference on Business Informatics (CBI)*, 2017, pp. 7–12.

[19] ——, "Using deep learning to detect price change indications in financial markets," in *Proceedings of the European Signal Processing Conference (EUSIPCO)*, 2017, pp. 2511–2515.

[20] A. Ntakaris, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Mid-price prediction based on machine learning methods with technical and quantitative indicators," *SSRN*, 2018.

[21] D. T. Tran, A. Iosifidis, J. Kanniainen, and M. Gabbouj, "Temporal attention-augmented bilinear network for financial time-series data analysis," *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

[22] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 3, pp. 653–664, 2017.

[23] J. Moody and M. Saffell, "Learning to trade via direct reinforcement," *IEEE Transactions on Neural Networks*, vol. 12, no. 4, pp. 875–889, 2001.

[24] J. E. Moody and M. Saffell, "Reinforcement learning for trading," in *Proceedings of the Advances in Neural Information Processing Systems*, 1999, pp. 917–923.

[25] S. Nison, *Japanese candlestick charting techniques: a contemporary guide to the ancient investment techniques of the Far East*. Penguin, 2001.

[26] D. Yang and Q. Zhang, "Drift-independent volatility estimation based on high, low, open, and close prices," *The Journal of Business*, vol. 73, no. 3, pp. 477–492, 2000.

[27] J. J. Murphy, *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin, 1999.

[28] C. Y. Huang, "Financial trading as a game: A deep reinforcement learning approach," *arXiv preprint arXiv:1807.02787*, 2018.

[29] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011.

[30] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proc. of the AAAI Conference on Artificial Intelligence*, 2018.

[31] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

Avraam Tsantekidis is currently a PhD candidate at the Department of Informatics in the Aristotle University of Thessaloniki, where he also received his B.Sc. in Informatics in 2016. His research interests include Machine Learning applications in Finance with most of his effort focused on Deep Learning methods. During his PhD he worked in a capital management firm where he applied methods Deep Learning methods for optimizing trading activities.

Nikolaos Passalis is a postdoctoral researcher at the Aristotle University of Thessaloniki, Greece. He received the B.Sc. in Informatics in 2013, the M.Sc. in Information Systems in 2015 and the Ph.D. degree in Informatics in 2018, from the Aristotle University of Thessaloniki, Greece. He has (co-)authored more than 50 journal and conference papers and contributed one chapter to one edited book. His research interests include Deep Learning, information retrieval and computational intelligence.

Anastasia-Sotiria Toufa obtained her B.Sc. in informatics in 2018 from Aristotle University of Thessaloniki. She worked as intern at Information Technology Institute of CERTH and she is currently studying for her M.Sc. in Digital Media - Computational Intelligence in the Department of Informatics at the Aristotle University of Thessaloniki. Her work includes machine learning, neural networks and data analysis.

Konstantinos Saitas-Zarkias is currently studying for his M.Sc in Machine Learning at KTH, Sweden. He obtained his B.Sc in Informatics in 2018 from Aristotle University of Thessaloniki, Greece. His current research interests include Deep Learning, Reinforcement Learning and Autonomous Systems, he also has an interest in AI ethics.

Stergios Chairistanidis is a Machine Learning Engineer / Quantitative Developer at Speedlab AG. He received the received the B.Sc. Electronics / Information Technology from the Hellenic Airforce Academy in 2008, the B.Sc. in Computer Science and Applied Economics from the University of Macedonia in 2013 and the M.Sc. in Management and Informatics from the Aristotle University of Thessaloniki in 2016.

Anastasios Tefas received the B.Sc. in informatics in 1997 and the Ph.D. degree in informatics in 2002, both from the Aristotle University of Thessaloniki, Greece. Since 2017 he has been an Associate Professor at the Department of Informatics, Aristotle University of Thessaloniki. From 2008 to 2017, he was a Lecturer, Assistant Professor at the same University. From 2006 to 2008, he was an Assistant Professor at the Department of Information Management, Technological Institute of Kavala. From 2003 to 2004, he was a temporary lecturer in the Department of Informatics, University of Thessaloniki. From 1997 to 2002, he was a researcher and teaching assistant in the Department of Informatics, University of Thessaloniki. Dr. Tefas participated in 12 research projects financed by national and European funds. He has co-authored 92 journal papers, 203 papers in international conferences and contributed 8 chapters to edited books in his area of expertise. Over 3730 citations have been recorded to his publications and his H-index is 32 according to Google scholar. His current research interests include computational intelligence, Deep Learning, pattern recognition, statistical machine learning, digital signal and image analysis and retrieval and computer vision.