AutoTrade: Automatic Stock Market Trading with a Deep Reinforcement Learning Agent

Yuval Heffetz
Ben-Gurion University
Beer Sheva, Israel
yuvalhef@post.bgu.ac.il

Eyal Arviv
Ben-Gurion University
Beer Sheva, Israel
eyalar@post.bgu.ac.il

Hila Delouya
Ben-Gurion University
Beer Sheva, Israel
hilade@post.bgu.ac.il

ABSTRACT

Recent advances in the field of deep reinforcement learning (DRL), and more specifically DRL algorithms that can operate on sequential time series data, make the prospects of applying these methods to real-world sequential decision making problems more promising than ever. In this work we suggest a new computational finance approach by combining time series analysis, text analysis and DRL to predict stocks' trends and suggest when to buy or sell these stocks. We extend the traditional task of stocks' price forecasting as a trading support system for human decision making, and propose an agent that automatically trades stocks. Our framework, called AutoTrade, incorporates time-series feature engineering methods along with a DRL agent that learns how to act optimally in a trading environment with the use of a recurrent neural network.

KEYWORDS

stocks, temporal data, clustering, neural networks, reinforcement learning, sentiment analysis

1 INTRODUCTION

The task of forecasting stock prices by artificial intelligence agents has been widely studied and discussed for many years [30]. The explosion of research and innovations in the fields of machine learning (ML) and deep learning (DL) in recent years, and more specifically in the domain of time-series data, makes the prospects of designing systems to aid human decision making for the stocks trading problem more appealing than ever.

However, the stock prices forecasting task is considered unconventional and more challenging than many other time-series forecasting tasks. The efficient-market hypothesis (EMH) is a hypothesis in financial economics that states that asset prices reflect all available information [14], and according to this hypothesis, stock prices cannot be forecasted. It is suggested that stock prices proceed in a stochastic manner - an idea known as Random Walk Hypothesis (RWH). On the other hand, there is another hypothesis that claims forecasting stock prices is possible using different time series forecasting models. Although it is not clear which approach is more reliable, we see many works that support the latter using different types of models [3, 19, 28].

While forecasting stock prices is a formidable task that can aid people when trading stocks, it does not provide full support for decision making while trading. In this work, we present, a first to our knowledge, technique that combines time series analysis and text analysis with reinforcement learning (RL) to generate an automatic trading agent, called AutoTrade. AutoTrade performs an online, automatic trading decisions to accumulate maximal profit over time.

RL is a sub-field of machine learning. It refers to the group of algorithms and methods in which an agent interacts with an environment and learns to act optimally in it to achieve maximal reward over time. It is a powerful tool for sequential decision making problems [34]. Relying on successful works of stock price prediction with artificial neural networks [27], and the recent advances in the field of deep reinforcement learning (DRL) which incorporates ANNs in RL agents, we propose a fully automated DRL trading agent that (implicitly) predicts the future price of a stock and the optimal action to take – either to sell, buy or hold – according to the price forecast.

In our work, we perform extensive time-series analysis of the data. We create clustering of similar stocks to identify trends in the cluster of a stock before it happens in the stock itself. As a preprocessing stage we perform temporal abstraction with symbolic aggregate approximation (SAX) to improve the similarity computation between stocks. In addition, we analyse text in news and tweets relating to stocks to obtain the sentiment towards them, with the hope it will help predict the future behaviour of the stocks.

2 BACKGROUND AND RELATED WORK

In this section we will present several works in fields related to our research.

2.1 Algorithmic Trading

Financial time series have some characteristics that makes them hard to forecast, especially when a traditional statistical method is used [28]. These characteristics are as follows: (1) Non-stationarity: because of the different business and economic cycles, the statistical properties of financial data change over time; (2) Nonlinearity: the relationship between the financial and economical independent variables and the desired dependent variable is not linear; (3) Noisiness: there are day-to-day variations in financial time series.

Recent studies which employ artificial (computational) intelligence methods such as artificial neural networks (ANN), support vector machines (SVM), genetic algorithms (GA) etc. suggest that significant levels of market inefficiency is present in a wide range of markets hence predictability of prices is viable. We can divide this prediction task into two main approaches of analyses, the first being univariate and the later multivariate, where ANNs can be used in both [6]. In univariate analyses, the features are limited to the variable being forecasted, while in multivariate case, any variable that is thought to be related to the output is considered a feature (an input of the model).

ANN is one of the most predominantly preferred and also in widespread use in the industry [18]. The following points justify the use of ANNs to forecast financial time series: (1) ANNs are non-linear. It means that they can capture nonlinear relations between feature (input or independent) and response (output or dependent) variables; (2) ANNs are data driven. It means that they do not need any explicit assumption on the model between inputs and outputs; (3) ANNs can generalize. It means that after training, they can produce good results even when they face new input patterns, and; (4) Unlike statistical techniques, ANNs do not need assumptions on the distribution of input data.

Few papers proposed the use of ANNs to forecast stock prices and financial trends. In a work by Taghi et al., they proposed a feed-forward ANN model to which they fed a 27 input variables to predict the stock trend the following day [28]. To reduce the combinations space they omitted features with correlation higher than 0.5 and created 6 groups of features that have similar nature within them. they kept only features that have significant influence on the output variable. The output layer consisted of only one node that determines the daily direction of the stock. They showed that using their model on the test period to decide each day whether to buy or sell the next day (by the model's prediction) is statistically significant better (at 5% significance level) than the buy-and-hold strategy (buying at the beginning of the test period with all the same initial amount and then sell at the end of the test period). In addition, the authors of the paper showed that the ANN's results outperform the results of a logistic regression model which validates the non-linearity of the relation between financial and economical variables. In our work we will take the methodology of this work to reduce redundant features, and we will apply time series analysis such as moving average on the features to gain more predictive features.

The study of Nair et al. [27] show two approaches to forecast one-day-ahead stock prices levels. The first approach views the stock price level forecasting problem as a regression problem using a Genetic Algorithm (GA) optimized symbolic aggregate approximation (SAX) - ANN based regression system. The second approach views the level forecasting problem as a classification problem and uses a similar GA-optimized-SAX-ANN based system. In our work we will apply SAX as well, which allows a huge dimensional reduction of the time series while maintaining the main characteristics of the financial data.

Another work of Canelas et al. [5] proposes a combination of SAX technique together with an optimization kernel based on genetic algorithms (GA), where the SAX representation is used to describe the financial time series, so that relevant patterns can be efficiently identified. The evolutionary optimization kernel is used to identify those most relevant patterns and generate investment rules. They showed their results outperforms some state-of-the-art solutions.

Many other works showed the importance of data mining in order to analyze large amounts of data and discover knowledge, patterns [5, 21] and future trends. In the work of marketos et al. [23] they used temporal data mining to build the Intelligent Stock Market Assistant that is capable of suggesting buying or sell stocks. It also takes into consideration the risk level, buying price and the need for cash.

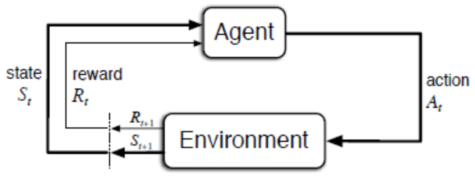

Figure 1: Interaction between an aget and the environment in a Markov Decision Process. Figure taken from [35].

K-means algorithm has been used as part of a process for stock trend prediction. For example, [38] proposed a model based on the Apriori All algorithm and K-means where they focused on detecting frequent patterns and predicting the trend corresponding to an input pattern in a financial time-series dataset such as stock history. This study could be applied in recommending both buying and selling actions. However, it is primarily focused on improving the accuracy of predictions without building the practical recommendation systems for investment. Another study [20] used a Hierarchical agglomerative and Recursive K-means clustering method to predict the short-term stock price movements after the release of the financial reports. The method consists of three phases. First, each financial report is converted into a feature vector and hierarchical agglomerative clustering method is used to divide the converted feature vectors into clusters. Second, the K-means clustering method is used to partition each cluster into sub-clusters so that most feature vectors in each sub cluster belong to the same class. Then, for each sub-cluster, its centroid is chosen as the representative feature vector. Finally, the representative feature vector is used to predict the stock price movements. In our work we will apply K-means algorithm as well to cluster together similar features and use the clustered features as the input features to our model.

2.2 Deep Reinforcement Learning (DRL)

While the previous work mentioned above can support human decision making with regard to trading stocks by predicting their future price, our work aspires to fully automate the end-to-end trading process. We propose a Reinforcement Learning (RL)-based decision making framework that automatically trades stocks.

In RL models, an agent interacts with the environment over time. When a RL problem satisfies the Markov property, i.e., that a state is dependent only on the previous state and action, it is formulated as a Markov Decision Process (MDP), a formalization of sequential decision making where the actions taken in a certain step can influence the future rewards of a trajectory [35]. In each time-step t and given the current state S_t, an agent picks an action A_t that will bring it to the next state S_{t+1} (i.e. S') with a reward R_t. The process is depicted in Figure 1. In RL, the underlying goal of the agent is to find a policy $\pi : S \longrightarrow A$ that maximizes the return, which is defined in Eq. 1:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k} \tag{1}$$

Where $\gamma \in (0, 1]$ is a discount factor.

Traditional RL methods, as well as DRL algorithms, can be categorized to *value-function*-based methods and *policy gradient* methods.

Deep Q-Networks (DQN) is perhaps the most used value-function-based DRL algorithm [26]. It incorporates a deep NN as a *Q-function* approximator and follows the traditional RL Q-learning algorithm [35]. A *Q-function* $Q^{\pi} : S \times A \rightarrow \mathbb{R}$, also known as the *action-value function*, is defined as the expected sum of future rewards used to measure how good an action is given the current state and following a policy π:

$$Q^{\pi}(s_0, a_0) = E_{\pi}[R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, s_2) + \ldots] \qquad (2)$$

DQN manages a Q-function NN, $\hat{Q}(s, a; \theta)$, to estimate (predict) the Q-value for each state-action pair. The weights of the Q-function NN are updated using gradient descent on the Temporal-Difference (TD) error which relies on the *Bellman equation* [34]:

$$R(s, a, s') + \gamma \max_{a} Q(s', a; \theta') - Q(s, a; \theta) \qquad (3)$$

When the Q-function NN is fully trained, the DQN agent follows a greedy policy $\pi = \arg\max_a Q(s, a; \theta)$.

The strength of DQN over the traditional RL methods, including Q-learning, is its ability to operate in high-dimensional problems with exponential number of states such as the problem defined in this work. It is possible due to the ability of the NN to generalize across states, so when a new unseen before state is given as an input it will be able to accurately predict its Q-values relying on past experiences.

More recent advances that follow DQN include the Dueling-DQN (D-DQN) algorithm [31] and DQN with Prioritized Experience Replay (PER) [37]. D-DQN decouple the Q-function's NN architecture to two sub-architectures, one for the state value function and another one for the actions, allowing the agent to evaluate how good a state is separately from the actions. PER organizes the past experiences of the agent in a prioritized order, sorted by the TD-error of each observation. Both methods enhance the convergence properties of the DQN model.

Some methods try to optimize the policy $\pi(a|s; \theta)$ directly with Policy Gradient (PG) algorithms, that work on the policy space itself. These methods are arguably more stable and suited to high-dimensional actions spaces but are also less sample efficient than DQN methods [35]. PG methods use a probability over actions parameterized by θ, expressing the probability to take action a in state s and defining the policy itself. The parameters (e.g., weights of a NN) are updated with gradient ascent to maximize the expected return $G_t = \mathrm{E}[\sum_t^{\infty} R(s_t)|\pi_{\theta}]$ [35].

Advantage actor critic (A2C) is an example of a state-of-the-art policy gradient algorithm, which utilises both a policy NN to pick actions (i.e., an actor) and an action-value NN, used as a critic [25]. The critic leads the direction in which the policy network is updated in each time-step, reducing the variance without introducing bias to the model. Another policy gradient method is the Trust Region Policy Optimization (TRPO) [32] and its more recent follow-up work, the Proximal Policy Optimization (PPO) algorithm [33]. Both TRPO and PPO are suitable for problems with large action spaces. They use a trust region of the current policy from which they sample actions and thus decreasing the variance of the model and make the training less noisy.

Some RL problems, as well as the problem in this work, involve environments that incorporate time-series data and thus can't use the Markov property assumption in a straight-forward manner. For example, in some computer games such as Atari there is importance to the time factor, since a single frame of the game cannot capture the evolution of the game. When a player has a *direction* and *speed*, a single frame of the game, i.e. the *state* of the environment, is only a partial observation since it does not provide any knowledge of these two factors.

One solution, suggested by Mnih et al. in [26], is to stack the recent k frames of the game so they will together form the *observation*, used as the modified *state* of the environment. The stacked frames capture the recent evolution of the game in a simple yet effective way and together they form the input to the action-value Q-network. This method can also be applied in our work, by stacking the k previous feature vectors that represent the stock, capturing the price change as well as other features evolution. However, the stacking method still cannot provide complete observation since it has limited memory of the environment.

A more advanced method to solve this problem was presented in [16], using deep recurrent q-learning. Hausknecht et al proposed to incorporate a Recurrent Neural Network (RNN) in the action-value network of a DQN (the q-network). RNNs are models that are designed specifically for sequential data that its variables (inputs and output) cannot be treated as independent [24]. RNNs perform the same task for every element of a sequence, with the output being depended on the previous computations and have a *memory* which captures information about what has been calculated in the past. Advanced architectures of the recurrent cell include the gated recurrent unit (GRU) and the long short-term memory (LSTM) network, both excel at remembering values for either long or short duration of time by solving the problem of vanishing gradients [10, 17]. A deep recurrent q-learning agent is capable of seeing only a single frame (or feature vector of a single event, in our case) at each time-step and successfully integrates information through time, therefor it can be of great use in our temporal model.

Being a relatively new development, DRL has not been yet tested on many real world problems. However, its success on virtual environments such as computer games and robot simulations along with a few preliminary works on trading suggest that it could be deployed relatively easily on the trading task, when modeled in the correct way.

In 2016, Deng et al. proposed a trading system for financial signal representation and self-taught RL trading [13]. The proposed framework includes two different components, integrated together to a unified model. For the first component they use a deep NN, used for feature learning and deep representation of the dynamic trading environment. The second component is a RL agent, used for sequential decision making. The agent's policy is parameterized and is optimized by learning a continuous representation of the actions directly with traditional policy gradient methods. More specifically, they use a RNN architecture to perform the learning and optimization of the policy.

There model had two main challenges. The first challenge was vanishing gradients in the backpropagation through time (BPTT) process which they solved with task-aware BPTT that incorporates virtual links from the objective function, directly connected with the deep layers of the NN. The second challenge was the dynamic nature of the NN's input, which they managed with fuzzy learning concepts.

While their work produced promising results, the resulting framework is highly complex, integrating many different parts, all require lots of tuning. Their framework is designed specifically to handle the task at hand and cannot be easily reproduced to other similar tasks. Furthermore, they did not use state-of-the-art DRL models that are prevalent today. We on the other hand, intend to use more advanced methods which were proven to perform well in similar settings, such as DQN and A2C, and to design a framework that simultaneously use DL and RL as a unified system.

Another work that incorporates the use of RL for automatic trading is presented in [12]. In their work, Cumming et al. proposed a more unified approach to their trading framework, with a policy iteration RL agent operating over an environment that represents the stocks' n most recent prices using candlestick data. They presented promising results, even-though they too used a RL method that predates the current state-of-the-art algorithms, and more specifically the Least-Squares Temporal Difference Learning (LSTD) method [34].

A more recent (preliminary) research by Gao et al. (in a recent preprint) proposed the use of a DQN agent integrated with a LSTM network architecture on an idealized trading game [15]. They propose a trading environment where the agent can *buy* and *sell* a stock depending on its current price, modeled as the *state* of the game. There results on a uni-variate representation of the stock's state (i.e., with only the price variable) are promising, suggesting a real potential of deploying a DRL agent for the automatic trading task. Similarly, another recent work used an Actor-Critic DRL agent that managed to learn a profitable policy [39]. In this work, the agent could choose to invest in multiple stocks at a time, rather than just one as in [15].

We intend to follow the same guidelines but with a multi-variate representation of the stock, incorporating many more variables including analysis of news and tweets, data discretization and more. Furthermore, we use real stock prices from recent years to model a real world trading environment.

3 PROBLEM FORMULATION

For our task, we assume a dataset of n stocks, traded for k consecutive time-steps in a stock-market. Each stock i consists of a sequence of prices, denoted by $P_i = \{p_0, p_1, ..., p_k\}$. First, we define a trading strategy as -

$$T_t(a_t|s_t)$$

Where a_t denotes the trading action selected by the strategy at time-step t, given the current state of the stock, s_t.

Each trading action a is defined as the pair $(sign(a), amount(a))$, where the *amount* operator represents the amount of capital movement and the *sign* operator defines its direction -

$$sign(a) = \begin{cases} -1, & Sell \\ +1, & Buy \\ 0, & Hold \end{cases}$$

We also denote a commission for all actions as C, so the income (or cost) of the action is calculated by -

$$I(a) = sign(a) * amount(a) * (1 - C)$$

Given an initial balance B_0 that a trader begins with, we calculate its balance in time-step t, following strategy T by -

$$B_t = B_0 + \sum_{j=1}^{t} I(T_t(a_j|s_j))$$

Let N_t denote the number of shares the trader currently owns, we define its total *Net worth* in time-step t as $W_T = B_t + N_t * p_t$. Our goal is to find a trading strategy -

$$\arg\max_T \mathrm{E}[W_T(k) - W_T(0)]_n$$

To achieve this goal, we further formulate the problem as a Partially Observable Markov Decision Process (PO-MDP). The PO-MDP is defined by the tuple $\mathcal{M}(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the set of possible states (i.e., the features representing the state of the stock), $\mathcal{A}$ is the actions space of trading operations, $\mathcal{P}$ is the transition function, $\mathcal{R}$ is the reward function which is derived from the obtained profits, and $\gamma \in (0, 1]$ is a discount factor. Given the expected sum of discounted rewards $R_t = E_\pi[R(S_0) + \gamma R(S_1) + \gamma^2 R(S_2) + ...]$, produced by a policy $\pi : \mathrm{S} \longrightarrow \mathrm{A}$, our goal is to obtain

$$\arg\max_\pi R_t$$

Our work attempts to achieve this goal through the application of DRL and extensive data preprocessing and feature engineering for the stocks' time-series data.

4 METHODS

In this section we define the environment and the methods for data preprocessing.

4.1 Trading Environment

We created an episodic environment to simulate the trading task. Each episode ends after K consecutive time-steps, when a new episode begins. In each episode, a single stock can be traded by buying or selling shares of this stock. Each episode starts with a given initial balance, B_0, which is granted to the agent that is operating over the environment. We now define the actions that an agent can take when interacting with the environment, the reward function and the state representation of the environment.

4.1.1 ***Actions Space.*** We simplify the action definition from the previous section and define a discrete action space of twelve actions, $\mathcal{A} \in \{0, ..., 12\}$. The actions are categorized into three types of actions, *{Buy, Sell, Hold}*, where each action $a \in \mathcal{A}$ is defined as follows:

- Action $a \in \{0, 1, 2, 3\}$: **Buy** $N_{shares}(a)$
- Action $a \in \{4, 5, 6, 7\}$: **Sell** $N_{shares}(a)$
- Action $a \in \{8, 9, 10, 11\}$: **Hold**

Where $N_{shares}(a_t)$ is the number of shares that are traded (bought or sold) given action a_t in time-step t:

$$N_{shares}(a_t) = [1/(a\%4 + 1)] * N_{total}$$

And $N_{total}(a_t)$ is the total number of shares that can be currently traded:

$$N_{total}(a_t) = \begin{cases} B_t/p_t, & a \in \{0, 1, 2, 3\} \\ N_t, & a \in \{4, 5, 6, 7\} \end{cases}$$

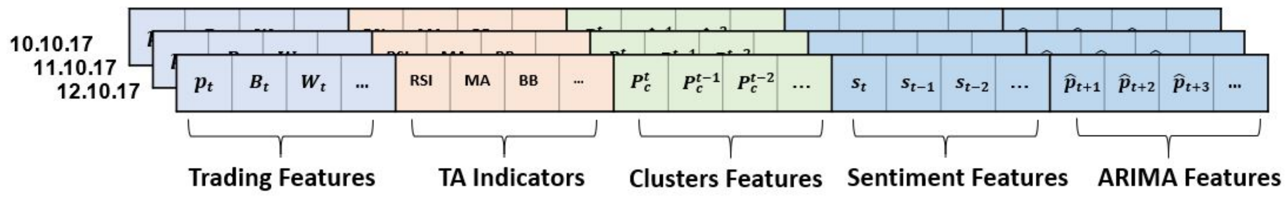

Figure 2: State vector.

Where N_t is the number of held shares (by the trading agent) in time t. To summarize, each action allows a trading operation in a certain amount of the stock's shares, relative to the total number of shares that can be traded at that time-step. This action space definition allows for relative flexibility while avoiding large or continuous action space with infinite number of choices of the amount of capital that can be traded.

4.1.2 ***Sortino Reward***. Choosing the appropriate reward function can have a significant influence on the behaviour of the agent's learned policy [35]. As described in the *Problem Formulation* section, our goal is to maximize the expectation of the trading agent's profit, over n stocks. To achieve this goal we shall design a reward function which is directly derived out of these profits. The simplest option is to define the reward as the immediate profit that was obtained from the most recent action, $R(t) = W_T(t) - W_T(t-1)$. However, this reward function may lead to a risky strategy, where the agent would learn to select actions that will cause short-term profits but with high risk of loosing money in the long run.

To avoid a risky strategy, we decided to utilize the *Sortino Ratio* as the reward function. The *Sortino Ratio* is a well known metric to evaluate the performance of trading strategies [7, 29]. This metric looks at the portfolio of returns (i.e., profits) obtained by our strategy over a long period of time, thus enabling to evaluate long-term behaviour. Furthermore, the *Sortino Ratio* compares the performance of our strategy, to the performance of a benchmark:

$$Sortino = \frac{R_p - R_b}{TDD}$$

Where R_p is the portfolio of profits obtained by our agent, R_b is the portfolio obtained by a benchmark strategy and TDD is the Target Downside Deviation, which is a normalization factor.

4.1.3 ***State Space***. In each time-step, the agent selects an action given the observation of the current state of the environment. In our case, the state of the environment represents the current conditions of the stock and of the trading process. Since we use DRL to solve the MDP, the observation of the state should be a vector (or matrix), used as an input to the DRL agent's Neural Network (NN). We define the *state vector* of our environment as a multivariate space, with multiple features representing the environment in a certain time-step.

A schematic representation of the vector is depicted in Figure 2. The vector consists of features of four types, the first being the *Trading Features*, which are the most basic features that describe the current state of the stock and trading process and do not require any feature engineering:

- Stock Price p_t
- Current Balance B_t
- Number of currently held shares N_t
- Number of shares that were bought or sold in the previous time-step
- Amount spent
- Amount earned

In the following sub-section we describe the extensive data preprocessing and feature engineering methods for time-series data that we undertook to complete the state vector and to create the features of the remaining three types that are shown in Figure 2.

4.2 Data Preprocessing

4.2.1 ***Technical Analysis Indicators***. Technical Analysis (TA) is a field of research that was developed through hundreds of years of trading. It is a general code name for a discipline that deploys statistical time-series analysis of stocks' price and volume data, in order to find specific patterns that indicate future behaviour of the stock [4, 36]. In TA, past prices movement and trading signals are analysed to create a series of a few dozen indicators of many sorts, some of which are mathematical transformations of price, often including up and down volume, advance/decline data and other inputs. These indicators are used to help assess whether an asset is trending, and if it is, the probability of its direction and of continuation.

TA is based on three assumptions: (1) all factors regarding a stock (e.g., company's success, market behaviour, etc.) are already reflected in past price movements and price patterns; (2) a stock price is more likely to continue a past trend than move erratically, and; (3) price movements have a repetitive nature and it is possible to use chart patterns to analyze these market movements and understand trends. The indicators that are based on these assumptions and are widely used to help traders in their decision making are grouped into three main measures:

- Price Trend: Indicators that are designed to look at past prices and produce a positive signal for an expected upwards trend and a negative signal for a downwards trend. An example of such indicator is the Moving Average Convergence Divergence (MACD) indicator, that uses two exponential moving average curves of the price data (with different parameters) to compute a third curve by the difference between them [11]. The indicator signals a negative trend when the third curve falls beneath the zero and a positive signal when it rises above the zero.
- Volatility: Indicators that are used analyze the volatility of the price of a stock in a specific period of time. For example, the Bollinger Bands (BB) indicator, that computes the price'

moving average curve and creates a band around it with an upper and lower borders in a distance of X standard deviations from the moving average curve [1]. When the price of the stocks meets one of the band's borders, a volatility signal is indicated.

- Volume: Indicators that try to find correlation between the volume of trade in the stock with its price, in order to indicate price movements. Includes the Money-Flow Indicator [8], Volume-price Trend [22] and more.

In this work we craft forty features out of the few dozen TA indicators, all added to the environment's state vector (see Figure 2). Our assumption is that adding these indicators as features may assist our agent in the decision making process, much like it assists human traders.

4.2.2 ***Distance matrix and Clustering.*** Realizing that each stock is not traded by itself in the world, but rather in a stock market in which many stocks are traded at the same time and influence each other, we set out to explore this effect to help predict future behavior of stocks. We intend to infer the future price of a single stock by exploring past data of stocks which are similar to the evaluated stock. We do so by creating clusters of similar stocks and extracting features regarding each cluster.

As a preliminary step, we normalize the stocks' price variable using standard normalization by subtracting the mean of the price data and scaling it to a unit variance. The purpose of this preprocessing step is to enable more simple and efficient comparison of the behavior through time of multiple stocks under a single similarity measure.

After normalizing the data, we created distance matrix using two methods. The first method is using Dynamic Time Warping (DTW) which is an algorithm for measuring similarity between two temporal sequences, which may vary in speed. using DTW we found an optimal alignment between each pair of stocks using Euclidean distance and achieved the distance between each of the stocks in our data. The second method was performed using temporal abstraction with SAX for each of the stocks' price data. For the SAX method we used 5 segments for the Piecewise Aggregate Approximation (PAA) part, as this resulted in the best performance of our DRL agent, and experimented with various number of SAX symbols, as we will show later in the *Results* section. Using the output of the SAX algorithm, we also created a distance matrix that keeps the distance between each of the stocks in our data.

Afterwards, using the distance matrix of our stocks we created clusters of stocks using two methods. The first clustering algorithm is Density-Based Spatial Clustering of Applications with Noise (DBSCAN), and the second is single-link hierarchical clustering for which we used a linkage matrix.

After obtaining the clusters of the stocks, we use them to create a set of features to each of the stocks in each cluster. Let i be the index of the stock that is currently traded by our agent and for which we create the current state vector. We look for the cluster, c, to which stock i belongs and use the price data of all the stocks in c to extract the set of features $\{P_c^t, P_c^{t-1}, ..., P_c^{t-b}\}$, where P_c^t is the average price of the stocks in cluster c at time-step t and b is the number of time-steps we want to look back. These features are added to the state vector (see Figure 2) to be used in our model. We hypothesise that these features would assist our agent to decide its actions regarding the trading of the current stock i.

Note that we considered the process of optimizing the clusters as supervised, where the target of the optimization was the overall performance of the DRL trading agent using the features obtained by each of the methods for creating the clusters (e.g., selecting the similarity metric, number of symbols, etc.).

4.2.3 ***Sentiment Analysis Features.*** We performed sentiment analysis on tweets from Twitter about the stocks to obtain additional features to the state vector (see Figure 2). For that purpose we collected tweets that are related to each one of the stocks in our data, and then calculated average sentiment analysis score on each of the tweet's text for each stock in every timestamp available. These averaged sentiment analysis scores were then added as additional features to the state vector of the currently traded stock.

Since our ability to gather tweets for all the stocks in our data and for all the time period was very limited, we performed a data enrichment method. By calculating the average sentiment in the cluster of each stock, we were able to use it instead of the actual sentiment towards the stock in case of missing data.

4.2.4 ***Data Stationarity and ARIMA.*** The final set of features we add to the state vector is a set of price forecasting features. In each time-step t of AutoTrade, we create a price forecasting model. The model is trained on past price data of the currently traded stock, from time 0 to time t and then used to forecast the stock's prices from time $t+1$ to time $t+f$ where f is a parameter determining how many time steps ahead we want to forecast. The set of predictions, $\{\hat{p}_{t+1}, \hat{p}_{t+2}, ..., \hat{p}_{t+f}\}$, is used by AutoTrade as features in the state vector (see Figure 2). If the price predictions are accurate enough, we believe that adding them as features may help our agent to decide which action is optimal given these predictions in a certain timestamp.

Since the price forecasting model is generated, trained and used for prediction in each and every time-step of AutoTrade, we want it to be a very light and efficient model. For that reason we decided to use the Autoregressive Integrated Moving Average (ARIMA) model in a univariate setting [2]. Autoregressive and Moving Average (ARMA) model is an important method to study time series and for time-series forecasting. The concept of autoregressive (AR) and moving average (MA) models was formulated in [9]. ARIMA is based on the ARMA model, where the "I" in the ARIMA model stands for *Integrated* and it is a measure of how many nonseasonal differences are needed to achieve stationarity[1]. What sets ARMA and ARIMA apart is differencing. If no differencing is involved in the model, then it becomes simply an ARMA model which is therefore a stationary model. Since our data isn't stationary over time and because we wanted to focus on forecasting only the closing price of each stock at each time stamp we used ARIMA model.

Before using the model, we transformed, as a preprocessing stage, the time-series data from non-stationary to stationary by taking a series of differences of our data. In addition to the ARIMA model, the transformed, stationary time-series was also used by the final DRL agent.

[1]A stationary time series is one whose statistical properties such as mean, variance, autocorrelation, etc. are all constant over time.

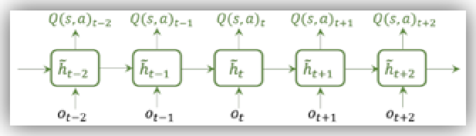

Figure 3: Model Architecture.

4.3 DQN Agent & LSTM NN

We use a DQN agent with a LSTM architecture of the Q-function – both methods are explained extensively in the Related Work Section. The choice of a value-based algorithm (i.e., DQN) was made because of the relatively small, discrete, action space of our environment, which value-based methods excel at. Furthermore, DQN is much more sample efficient than policy gradient methods and since the amount of data we hold is limited, DQN seems a more appropriate choice.

Since our agent operates on a time-series (temporal) data, we design a LSTM based NN architecture for the q-function. The goal of the NN is to learn patterns in the multivariate temporal data and to output a vector of q-values, with one entry per possible action (see Actions Space sub-section), given the observation of the current state. We rely on the approach that was suggested in [16], that used a RNN network to model the q-function and was discussed in the Related Work section. The LSTM models the transitions between the MDP states, and the input of each LSTM's time-step is the observation o_t of state s_t. The observation is the **state vector** (in a single time-step) that we defined in the previous section and presented in Figure 2.

The overall process is presented schematically in Figure 3. This kind of approach solves the problem of partial observability of the state vector since the LSTM preserves memory of past inputs and outputs and should process long sequences better than RNN.

5 EVALUATION

This section will address the following: first we introduce our research questions that we intend on answering in the evaluation section. Subsequently, we discuss the data we used in our work, and finally, we detail the evaluation metrics for validating our model's performance in comparison to other baseline methods and the experimental plan we used to evaluate the model.

5.1 Research Questions

Realising that the goals that we defined in the *Problem Formulation* section cannot be assessed directly due to the huge search space of possible trading strategies, we define the following, simplified research questions: (1) Is automatic trading possible using ML and Deep Learning methods? (2) Is automatic trading using DRL (i.e. with AutoTrade) better than using hand-crafted and rule-based trading strategies? (3) Does clustering and sentiment related features contribute in the tasks of predicting the trends of stocks prices and trading stocks?

Using the AutoTrade model we answer these questions empirically, through demonstration of the model's performance and with various evaluation methods.

5.2 Data

We used historical daily stock prices data of 457 stocks that are in the top 1000 traded US stocks between the years 2009 and 2017. For each stock we have the following information: date, open price (price of the stock at the beginning of the corresponding date), close price (price of the stock at the end of the corresponding date), high price (highest price of the stock during the corresponding date), low price (lowest price of the stock during the corresponding date) and volume (amount of traded shares for the specific stock during the corresponding date).

In addition, using Twitter's API, we collected tweets that discuss topics related to the stocks in our dataset and over the relevant years.

5.3 Evaluation Metrics

The main measure used to evaluate our model's performance is the *Net Worth* that is obtained by our agent at the end of the trading period. The Net Worth exact definition is presented in the *Problem Formulation* section. Note that as part of the default settings of the agent, it sells all the shares that it holds at the last day of the trading period.

During training, we evaluate our agent's ability to learn by the measure of accumulated rewards over time. As detailed extensively in the *Methods* section, we use the Sortino Ratio as the agent's reward, so positive rewards signify a profitable trading strategy over time and over multiple stocks.

We further validate our model's performance by measuring the percentage of stocks among which it outperforms other baseline models and the percentage of stocks in which AutoTrade managed to produce a profitable strategy.

5.4 Parameters Settings

The initial balance of all models in the beginning of each trading period is 10,000 dollars. We set K to be 400, meaning that the trading period and the length of each RL episode is 400 consecutive trading days. We use $f = 10$ for ARIMA's forecasting lookhead and $b = 5$ for the number of days we look back in the clusters' features. Our NN architecture and DQN agent use the following parameters:

- LSTM with 250 nodes
- Adam Optimizer
- Learning Rate is set to 0.0005
- Discount factor is 0.99

SAX and clustering parameters that are used in the final version of AutoTrade after extensive experimentation are:

- 5 SAX symbols
- 5 PAA segments
- single-linkage hierarchical clustering with euclidean distance, inconsistent criterion and a threshold of 12.

5.5 Experimental Plan

Our experiment consists of two main parts - an *offline* training phase, in which we train our agent over multiple stocks, and an *online* test phase that simulates the deployment of AutoTrade in a real-world trading environment.

We split our data to train and test sets leaving the 400 last consecutive days of the data to the test set for each stock and the rest to to the train set. This results in a ratio of 0.8-0.2 between the train and test sets. We then trained the RL agent over the train set while each RL episode would include 400 consecutive days of a stock data, starting in a random date, in order to enable the agent to learn how to trade optimally in a period of 400 days (which is the length of the test-set). We then apply the model on the test set for each one of the stocks that are in our data to simulate real-world online use of our model as a trading algorithm. Not that we did not conduct cross-validation (Time-Series CV) due to computation resources limitations, as each fold would take about two days of training and evaluation.

We created 2 different models. The first was built using only the first two sets of features that were presented in the *Methods* section - Trading features and TA-Indicators features. These are all the features that use only the price data, and we denote them as monitory features. We will refer to this model as *AutoTrade*. The second model was built using sentiment, cluster and ARIMA features in addition to the monitory features. This model will be referred to as *AutoTrade+*

We compare the results of AutoTrade to other baseline methods. The first method we compare to is Relative Strength Index (RSI) Divergence which takes the velocity and manitude (momentum) price movements and predicts the trend of the stock price in future time stamps. If there is a negative trend signal - the model suggests to sell the stock, and if the there is a positive trend signal - the model suggests to buy the stock. Another baseline method we compare our model to is Simple Moving Average (SMA) Crossover which suggests to buy when long-term SMA rises over short-term SMA, and to sell when the opposite case occurs.

6 RESULTS

In this section we will present the results on both the training and the test phases.

First, we examine the ability of our agent to learn during the *offline* phase. We can see in Figure 4 that the average reward of both versions of AutoTrade has an increasing trend as the number of episodes increases. This indicates that the model converges and is able to learn profitable trading strategies as the agent's learning progresses. In addition, we notice a better performance of *AutoTrade+* model compared to the simpler *AutoTrade* model in the last few episodes, and a more stable learning process, implying that the features we added in *AutoTrade+* improve the model's performance. This proves that using sentiment, clustering and ARIMA features assists in our stock price forecasting task and improve the model's ability to find important patterns in the data and to create a better trading strategy accordingly.

When observing the results of our models on the test set, in Figure 5 and table 1, we notice higher Net worth values and final profit among both *AutoTrade* and *AutoTrade+* models compared to the other two baseline methods, with slightly better performance of *AutoTrade+* over *AutoTrade*. This indicates once again that using the additional features in *AutoTrade+* model achieves better results. Figure 5 also suggests that AutoTrade is having more difficulty in short periods of trading where it does worse than the baselines, but

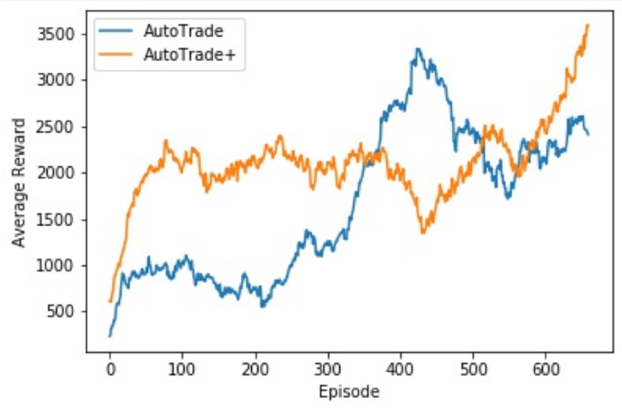

Figure 4: Average rewards of the DRL agent during training with respect to the number of training episodes. We compare the performance of *AutoTrade* with the *AutoTrade+* model.

in the long run it is more than compensating and accumulates a significant amount of money.

In addition, when we observe the actions our model takes during the test phase we see that, among the majority of the stocks, it learned to buy most of the shares of the stock in the beginning of the test period and sell it in the end. This resembles the Buy-and-Hold strategy where one buys all the shares at the beginning of the trading period and sells everything at the end of the period.

In table 2 we present the percentage of stocks each model was able to be profitable in. Both of our models *AutoTrade* and *AutoTrade+* outperformed the baseline methods, having 5% more stocks in which our models succeeded in producing profits on the test set.

Finally, table 3 shows the percentage of stocks in which our model outperformed the baseline methods (each individually and together) using different configurations of our model. We notice that single-link hierarchical clustering method for clustering the stocks is better than DBSCAN when using SAX and DTW for creating the distance matrix. In addition, Using SAX and single-link outperforms configuration using DTW. Observing our best results (SAX with 5 symbols and single-link clustering) we see that *AutoTrade+* outperformed the other baseline methods in most of the stocks, reaching 72.5% and 78.7% when comparing to RSI and SMA models respectively, and 66.6% compared to both RSI and SMA together.

Method	Profit
RSI	14.8
SMA	11.9
AutoTrade	34.4
AutoTrade+	**35.1**

Table 1: Average profit (percentage) at the end of the trading period (day 400) over 457 stocks. These results were achieved with the best configuration of our model as described in *Parameter Settings* subsection.

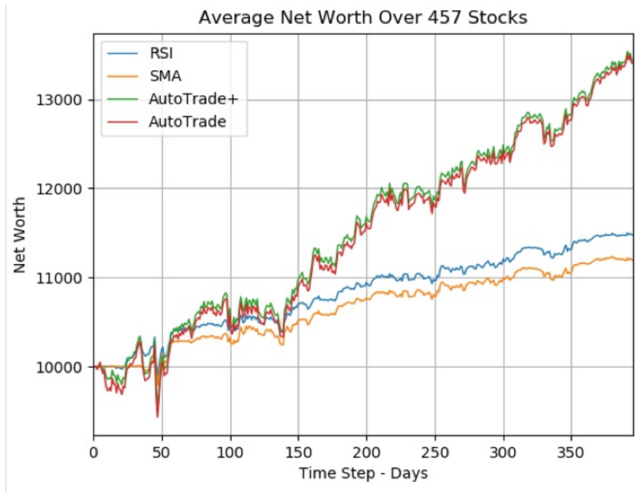

Figure 5: Average Net worth accumulation as a function of the number of days passed during the test period (400 days at most), comparing our models to the baseline trading methods. The figures that are shown are the average over 457 stocks in each trading day. As seen here, the trading period always starts with an initial balance (and Net Worth) of 10,000 US dollars.

Method	Profitable Rate
RSI	80.4
SMA	79.1
AutoTrade	84.8
AutoTrade+	**85.1**

Table 2: Percentage of stocks in which each model was profitable on the test set. These results were also achieved with the best configuration of our model as described in *Parameter Settings* subsection.

DM	Clustering	#SAX symbols	RSI	SMA	RSI+SMA
SAX	DBSCAN	3	45.3	52.3	39.2
SAX	DBSCAN	4	44.2	57.2	38.5
SAX	DBSCAN	5	48.7	58.8	43.7
SAX	DBSCAN	7	45.3	55.5	40.3
SAX	DBSCAN	10	43.2	52.3	37.2
SAX	Single-link	3	71.16	70.98	62.61
SAX	Single-link	4	70.1	69.1	60.4
SAX	Single-link	5	**72.5**	**78.7**	**66.6**
SAX	Single-link	7	67.9	71.2	59.6
SAX	Single-link	10	62.2	65.4	53.2
DTW	Single-link	n/a	61.4	58.3	48.9
DTW	DBSCAN	n/a	55.3	57.2	43.7

Table 3: Percentage of stocks in which *AutoTrade+* outperforms each of the baseline methods (RSI and SMA) or both of the methods together regarding the Net worth value in the end of the test period. The columns in the table are as follows: Distance matrix method (DM) used, Clustering method used (Clustering), number of SAX symbold in case it is relevant (#SAX symbols) and the rest are the percentage of stocks in which our model outperformed the baselines.

7 DISCUSSION

As we saw in the *Results* section, *AutoTrade* outperformes the other rule-based baseline methods. We also saw that the performance of *AutoTrade+* model yields slightly better results than *AutoTrade* model. This proves that using sentiment analysis created from social network discussion about the stocks can help forecasting the stock price in the future. In addition, using price predictions with ARIMA as features and features created from clusters of similar stocks helps to our task as well.

The fact that the Net worth obtained by AutoTrade's agent increased at the end of the test's trading period indicates that automatic trading is indeed possible and can result in a profitable trading strategy. Moreover, despite the fact that the natural average growth of the stocks prices was 34%, our model received a growth of 35% revenue at the end of the test period on average. This shows that our model performs better, to a small degree, than the naive strategy of Buy-and-Hold where one buys all the shares at the beginning of the trading period and sells everything at the end of the period. The fact that our model performs actions that resembles the Buy-and-Hold strategy shows that it learned that acting in a similar way to Buy-and-Hold is the best strategy to gain maximal Net worth over long periods of time.

This understanding also marks the possible limitation of our model. It is possible that when tested over trading periods where the average prices fall over time, AutoTrade would perform worse than the baseline methods. Further investigation is needed to establish a more comprehensive understanding of AutoTrade's ability to trade in these settings, however we did not conduct this analysis due to the fact that no such data is currently available. The fact that AutoTrade is profitable in more stocks than the baselines suggests that it is more adaptable to various price trends than the baselines.

8 CONCLUSIONS

We presented AutoTrade, a DRL-based framework for automatic trading. In our work, we conducted extensive time-series analysis and feature engineering that resulted in a model that outperforms baseline methods. We showed that using additional features from various domains can help in the tasks of forecasting the stocks prices and of automatic trading. Our model achieves results that are not far from a Buy-and-Hold strategy. This indicates that there is room for additional optimization and further analysis to achieve results that are significantly better. In addition, since our test data has an average natural growth of 34% it is also interesting to test our model on other testing sets that have a zero growth or negative growth over the test period.

REFERENCES

[1] John Bollinger. 2002. *Bollinger on Bollinger bands.* McGraw Hill Professional.
[2] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015. *Time series analysis: forecasting and control.* John Wiley & Sons.
[3] William Brock, Josef Lakonishok, and Blake LeBaron. 1992. Simple technical trading rules and the stochastic properties of stock returns. *The Journal of finance* 47, 5 (1992), 1731–1764.

[4] David P Brown and Robert H Jennings. 1989. On technical analysis. *The Review of Financial Studies* 2, 4 (1989), 527–551.

[5] AntóNio Canelas, Rui Neves, and Nuno Horta. 2013. A SAX-GA approach to evolve investment strategies on financial markets based on pattern discovery techniques. *Expert Systems with Applications* 40, 5 (2013), 1579–1590.

[6] Lijuan Cao and Francis EH Tay. 2001. Financial forecasting using support vector machines. *Neural Computing & Applications* 10, 2 (2001), 184–192.

[7] Ashraf Chaudhry and Helen L Johnson. 2008. The efficacy of the Sortino ratio and other benchmarked performance measures under skewed return distributions. *Australian Journal of Management* 32, 3 (2008), 485–502.

[8] Haiqiang Chen, Terence Tai-Leung Chong, and Xin Duan. 2010. A principal-component approach to measuring investor sentiment. *Quantitative Finance* 10, 4 (2010), 339–347.

[9] S Chen, X Lan, Y Hu, Q Liu, and Y Deng. 2014. The time series forecasting: from the aspect of network. *arXiv preprint arXiv:1403.1713* (2014).

[10] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).

[11] Terence Tai-Leung Chong and Wing-Kam Ng. 2008. Technical analysis and the London stock exchange: testing the MACD and RSI rules using the FT30. *Applied Economics Letters* 15, 14 (2008), 1111–1114.

[12] James Cumming, Dr Dalal Alrajeh, and Luke Dickens. 2015. *An investigation into the use of reinforcement learning techniques within the algorithmic trading domain.* Ph.D. Dissertation. Master's thesis, Imperial College London, United Kiongdoms, 2015. URL http ....

[13] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. 2016. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems* 28, 3 (2016), 653–664.

[14] Efficient-market hypothesis. 2012. Efficient-market hypothesis — Wikipedia, The Free Encyclopedia. (2012). https://en.wikipedia.org/wiki/Efficient-market_hypothesis [Online; accessed 19-December-2019].

[15] Xiang Gao. 2018. Deep reinforcement learning for time series: playing idealized trading games. *arXiv preprint arXiv:1803.03916* (2018).

[16] Matthew Hausknecht and Peter Stone. 2015. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*.

[17] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[18] Ozgur Ican, Taha Bugra Celik, et al. 2017. Stock market prediction performance of neural networks: A literature review. *International Journal of Economics and Finance* 9, 11 (2017), 100–108.

[19] Yakup Kara, Melek Acar Boyacioglu, and Ömer Kaan Baykan. 2011. Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange. *Expert systems with Applications* 38, 5 (2011), 5311–5319.

[20] Anthony JT Lee, Ming-Chih Lin, Rung-Tai Kao, and Kuo-Tay Chen. [n.d.]. An Effective Clustering Approach to Stock Market Prediction.

[21] Chiung-Hon Leon Lee, Alan Liu, and Wen-Sung Chen. 2006. Pattern discovery of fuzzy time series for financial prediction. *IEEE Transactions on Knowledge and data Engineering* 18, 5 (2006), 613–625.

[22] Anastasios George Malliaris, Jorge L Urrutia, et al. 1998. Volume and price relationships: hypotheses and testing for agricultural futures. *Journal of Futures Markets* 18, 1 (1998), 53–72.

[23] Gerasimos D Marketos, Konstantinos Pediaditakis, Yannis Theodoridis, and Babis Theodoulidis. 2004. *Intelligent stock market assistant using temporal data mining*. Citeseer.

[24] Larry Medsker and Lakhmi C Jain. 1999. *Recurrent neural networks: design and applications*. CRC press.

[25] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[27] Binoy B Nair, Nikhil Xavier, VP Mohandas, Adarsh Sathyapal, EG Anusree, Pawan Kumar, and Vignesh Ravikumar. 2014. A GA-optimized SAX–ANN based Stock Level Prediction System. *International Journal of Computer Applications* 106, 15 (2014), 7–12.

[28] Seyed Taghi Akhavan Niaki and Saeid Hoseinzade. 2013. Forecasting S&P 500 index using artificial neural networks and design of experiments. *Journal of Industrial Engineering International* 9, 1 (2013), 1.

[29] Tom RollingeR and Scott Hoffman. 2013. Sortino ratio: A better measure of risk. *Futures Magazine* 1, 02 (2013).

[30] Emad W Saad, Danil V Prokhorov, and Donald C Wunsch. 1998. Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks. *IEEE Transactions on neural networks* 9, 6 (1998), 1456–1470.

[31] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* (2015).

[32] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*. 1889–1897.

[33] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[34] Richard S Sutton and Andrew G Barto. 1998. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge.

[35] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[36] Mark P Taylor and Helen Allen. 1992. The use of technical analysis in the foreign exchange market. *Journal of international Money and Finance* 11, 3 (1992), 304–314.

[37] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. 2015. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581* (2015).

[38] Kuo-Ping Wu, Yung-Piao Wu, and Hahn-Ming Lee. 2014. Stock Trend Prediction by Using K-Means and AprioriAll Algorithm for Sequential Chart Pattern Mining. *J. Inf. Sci. Eng.* 30, 3 (2014), 669–686.

[39] Zhuoran Xiong, Xiao-Yang Liu, Shan Zhong, Hongyang Yang, and Anwar Walid. 2018. Practical deep reinforcement learning approach for stock trading. *arXiv preprint arXiv:1811.07522* (2018).