An Algorithmic Trading System Based On Deep Reinforcement Learning

Jeevitha A
Loyola-ICAM College of Engineering and Technology
Chennai, India
jeevitha.a@licet.ac.in

Sarah Prakriti Peters
Loyola-ICAM College of Engineering and Technology
Chennai, India
sarahprakritipeters.21cs@licet.ac.in

Ramadith M
Loyola-ICAM College of Engineering and Technology
Chennai, India
ramadith.21cs@licet.ac.in

John Richard
Loyola-ICAM College of Engineering and Technology
Chennai, India
johnrichard.21cs@licet.ac.in

ABSTRACT

Algorithmic trading is the automated process of buying and selling shares on the stock market. This paper presents a deep reinforcement learning approach to algorithmic trading. Algorithmic trading is traditionally carried out by human traders who follow stock market trends to buy, sell or hold shares. Human traders are prone to making errors due to human emotions which lead to losses. To tackle this problem, we propose to build an automated stock market trading system which uses a Deep Q-Network trader to buy, sell or hold stocks. For this system, we have collected stock market data for twenty companies from January 3rd, 2017 till January 6th, 2021. The automated algorithmic trader will predict which action should be taken during each point in time.

Keywords- *Stock market, Deep Learning, Trading, Q-Network*

INTRODUCTION

1.1. Problem Definition

Financial technology is a field that is developing at a rapid pace. The Fintech industry consists of sales of finance-based technological services. Fintech services range from traditional banking services to algorithmic trading. With the rapid development of fields such as Machine Learning and Artificial Intelligence, financial companies have been able to take advantage of fast and accurate technology to improve their services.

Algorithmic Trading, also generally known as Stock Market Trading, is the buying and selling of shares in the stock market based on an algorithm. It has grown in popularity since the 1980s. Human traders develop new strategies or improve existing strategies to create algorithms that will yield high returns. The main disadvantage that human traders face is that they are unable to control their emotions under situations of high pressure, such as trading large sums of money. This disadvantage can be overcome with an automated trading system that can predict when to buy, sell, or hold shares.

Reinforcement Learning is a machine learning technique in which an agent learns to take actions that yield positive rewards by observing its environment. Deep Reinforcement Learning uses a deep neural network to train the agent. A Deep Reinforcement Learning based agent can learn to take one of three actions - buy, sell, hold - by observing the stock market environment. Since the stock market environment is partially observable and dynamic, a reinforcement learning agent provides a sound foundation for a real-time algorithmic trading system.

1.2. Existing Systems

In traditional stock market trading, human traders follow complex stock market trends in order to take actions that will result in large returns. The human trader can choose to follow certain price points such as closing price, open price, high price, low price, adjusted close price, volume of the stock, et cetera. The price points are collected on a daily basis or even multiple times within the same day. For example, a human trader would follow the open, high and low price points to pick a trading action.

Current algorithmic trading systems or bots are fed in stock market data which is to be processed by the system or bot. Once data processing is complete, the system will train the agent using this data. An algorithm is defined which contains steps for each and every action and the subsequent returns are logged. This algorithm is called a trading policy or a trading strategy. The chosen strategy is evaluated based on different factors. For example, traders use profit gains, sharpe ratio, et cetera, as performance metrics.

1.3. Proposed System

The proposed system is based on Deep Reinforcement Learning. Therefore, the system is built upon a Deep Q-Network. In this system, Q-learning is used. It is based on the Q-table which contains Q-values for state-action pairs. Q-values are used to determine which action will be taken.

The DQN will form the agent. The agent should create the input state from the financial data input. The agent is trained on this data and the agent must predict the action at any given time. The proposed system will also have an inbuilt visualization module by which the trend of the close prices can be plotted, and along which the actions can be marked.

1.4. Methodology

For this project, stock market data has been collected for twenty companies from different industries over four years from January 3rd, 2017 till January 6th, 2021. For each stock, the close price is collected over four years and fed into the system.

The system generates the input state, which is the value of close price. Three output states are defined: buy, sell, and hold which creates the action space. The input state is fed into the first layer of the Deep Neural Network. The neural network will have three hidden layers. The output of the third hidden layer will be the Q-values which form the core of Q-learning.

Based on the epsilon-greedy policy in Q-learning, an action will be chosen. If a 'buy' action is predicted, the trader buys a share and subtracts the cost from its balance. If a 'sell' action is predicted, the trader sells the share and adds the price to its balance. If a 'hold' action is predicted, there will be no change to the balance.

A graph will be plotted in which the trendline of the close prices is visualized along with the predicted actions at those points. The idea here is to buy the shares at lower prices and sell the shares at higher prices to gain a profit. The trendline is plotted using differences in close prices over the period of four years. For each sale, the returns are calculated.

At each step, the trader's balance is updated. This system will be executed with the data of twenty different companies to see how it works with different stock close prices. Thus, this methodology is used to predict the actions given stock prices over a period of time.

2. System Architecture

The system is designed using a Deep Q-Network. The input dataset contains the values of the stock's close price. The dataset is split into 70% training data and 30% test data.

The training data is used to train the Deep Q-Network. During training, the system will get feedback in the form of a reward function. After training, the system is tested using the test data to perform trade actions.

Based on the test data, the system will predict whether to buy, sell, or hold a stock. The return from the trading process is calculated and displayed. Additionally, the days on which the stocks are sold or bought are displayed.

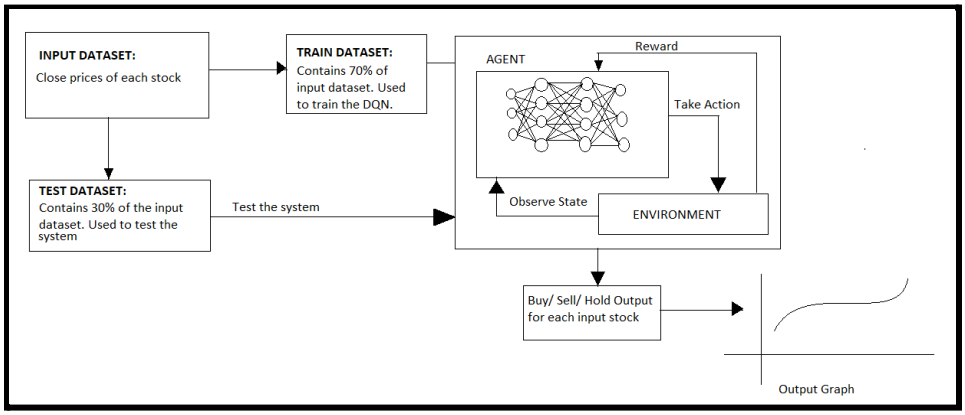

Fig 2.1. System Architecture

3. IMPLEMENTATION

3.1. Data Collection

For the trading and testing purposes, data has been collected from the Yahoo Finance API. Data was collected for twenty companies in different industries. The data collected consists of six points - open price, close price, high price, low price, adjusted close price, and volume. The companies for which data was collected fall under three major industries - Finance industry, technology industry, and media industry. The companies are Credit Suisse, JP

Morgan, BNP Paribas, HSBC, Bank of America Corporation, Bank of China, Wells-Fargo, Facebook, Amazon, Google, Twitter, Tesla, NVIDIA, Microsoft, Netflix, Reliance, AT &T, NBC, Disney, and Lionsgate. Data was collected from January 6th, 2017 to January 3rd, 2021.

3.2 Data Preprocessing

The data was downloaded from the Yahoo Finance API and stored as comma-separated values files. The data was first analysed using Python package Pandas. The missing values were replaced using K-Nearest Neighbours imputation. KNN imputation is used to identify 'k' samples in the dataset that are similar or close in the space. Then it uses these 'k' samples to estimate the value of the missing data points. Each sample's missing values are replaced using the mean value of the 'k'-neighbors found in the dataset. For the data, k = 5 was considered.

3.3 Deep Q-Network - Reinforcement Learning

Reinforcement Learning is a machine learning technique. Here, an agent will learn to perform a task by observing the environment, taking actions and obtaining rewards. The agent reaches different states by taking actions. The agent runs through episodes. One episode is the transition from the beginning state to the terminal state.

Q-learning is an off policy reinforcement learning algorithm that finds the best action for a state based on Q-values. Q-learning is off-policy because it updates its Q-values using the next state and the greedy action. Q-values are tabulated in the Q-table, which consists of states as rows and actions as columns. The below equation states that the Q-value obtained from the state s and performing action 'a' is the immediate reward r(s,a) along with the highest Q-value possible from the next state. Gamma, here, is the discount factor which controls the level of contribution of future rewards. Deep Learning is a subset of Machine Learning in which deep neural networks are used to train systems. In Deep Q-learning, the core of the system is not the Q-table, but rather the deep neural network that will be used to update the Q-table values and consequently to pick actions. In Deep Q-Learning, the state is fed into the neural network and it returns the Q-table values.

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

3.3.0.1 Activation Function

A neural network takes an input, passes it through hidden nodes, and outputs the combined input of all the nodes. The activation function decides whether the neuron's contribution to the network is important or not in the process of prediction. For this system, the activation function used is Leaky Rectified Linear Unit. Leaky Rectified Linear Unit, also known as Leaky ReLU, is an activation function based on a ReLU. However, unlike the ReLU activation function, where the gradient is 0 for all the values of inputs that are less than zero, Leaky ReLU has a small slope for negative values instead of a flat slope. This function returns x if it receives any positive input, else it returns a really small value which is 0.01 times x.

3.3.0.2 . Optimizer

Optimizers are algorithms or methods used to alter the weights and learning rate of the network to reduce the losses. An optimizer performs an optimization function so that the neural network can work effectively. For this system, two optimizers were considered. Adam Optimizer and Gradient Descent Optimizer. Gradient descent is an optimization algorithm that is commonly used to train a machine learning model. It alters its parameters repeatedly to reduce a given function to its local minimum. Adam is an adaptive learning method that calculates individual learning rates for different parameters. Based on the tabulated results, ADAM optimizer performed better than Gradient Descent Optimizer.

EPOCH NUMBER	LOSS (GD OPTIMIZER)	LOSS (ADAM)
50	0.004874	0.000054
100	0.009506	0.000051
150	0.008610	0.000055
200	0.006520	0.000018

Fig. 3.3.0.2. Gradient Descent Optimizer Vs ADAM Optimizer for JP Morgan Stock

3.3.0.3. Loss

To measure the efficiency of the system, we use a measure called loss. Our aim is to minimize the loss values. This function is referred to as a loss function and the value calculated by the loss function is referred to as "cost" or "loss". For this system, the loss function was implemented as Mean Squared Error. Mean Squared Error loss, or MSE, is calculated as the mean value of the squared differences between the predicted and actual output values.

3.3.1 Epsilon-Greedy Policy

In Q-learning, the agent picks an action based on its reward. The aim of the agent is to maximize rewards, and hence, pick optimal actions. For this system, the agent picks each action based on the epsilon greedy policy. The agent must have a balanced approach to picking an action, which is why the agent must be allowed to perform exploration and exploitation. When epsilon value is equal to or slightly less than 1.0, the agent explores the environment and chooses actions at random. As the epsilon value decays, the agent begins to exploit the environment and picks an action based on Q-values. The epsilon value ranges from 1.0 to 0.01, and decreases by a decay value of 0.999

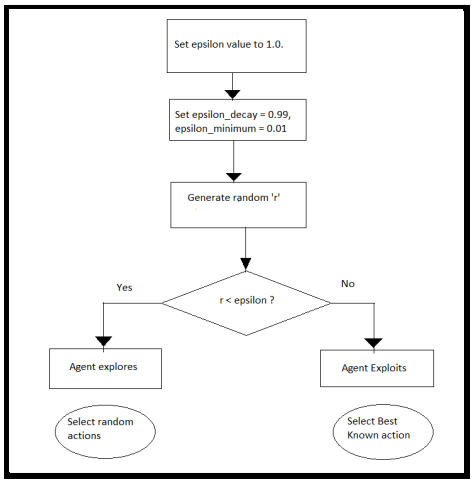

Fig 3.3.1. Epsilon-Greedy Policy

3.3.2 Input State Function

The input consists of the values of the close price point of each stock. The data is fed to the system using the get_state() function. This function will return the state as a Numpy array. Each array consists of thirty elements. First, calculate the id. When the id is positive, we create a state and if it is negative we append the close prices until we get to the duration. The function returns a state representation, defined as the adjacent stock price differences. For this system, the duration is set as thirty days**.**

3.3.3. Experience Replay

A subset of these experiences to update the Q-network. If we train our model on consecutive training examples, the model overfits due to correlation between training examples. To avoid this, we sample small batches of training examples. Batch sizes usually are 32/64/128 training examples This handles the problem of correlation between training examples, which could lead to errors during training.

Reward = Reward + [discount_factor * estimate of optimal future value]

To store the trader's (agent's) experiences, we used a data structure called a deque in Python's built-in collections library. It's a list that you can set an upper limit on so that if you try to append to the list and it is already full, it will remove the first item in the list and append the new item to the end of it. The experiences are tuples of [state, next state, reward, done, balance]. In this function, the Q-table values are updated, and the reward function values are updated.

3.3.4 Train Function

During training, the agent learns how to take an action based on the observed state. Within the training function, the agent observes the state, selects an action based on epsilon-greedy strategy and updates its memory. During training, the loss function is executed and loss values are logged. For each epoch, the logged values are displayed. The training function output consists of the running epochs, value of balance, and loss values.

3.3.5 Trade Function

During trading, the system takes the test data set as input. Within the trading function, two arrays named as buying_days and selling_days are updated based on the action taken (buy/sell respectively). The balance is also updated and tracked. The output of the trading function is displayed which shows on which days the shares are bought or sold.

3.4. Stock Analysis

The Deep Q-Network model gives us a satisfactory output while using close price values alone. However, it would not perform well in a real world scenario due to many reasons. First, stock market prices fluctuate constantly due to public interest, change in policies, change in decisions, et cetera. Second, a successful portfolio must also contain information about stock indicators such as Moving Average, Relative Strength Index, Williams Percent Range, Moving Average Convergence Divergence, et cetera.

These indicators provide additional information about the stock and how it will perform over a period of time, thereby allowing the agent to make better decisions. Third, stock market data over four years is considered a relatively short period of time. In most cases, portfolio managers consider stock market data over decades. For stock analysis, the Technical Analysis Python library (Ta-lib) was used. Technical Performance Indicators are calculations which are performed on different stock parameters such as open price, close price, volume et cetera. TA-Lib is an open-source python library. It is used to analyze the stock market's historical data such as share price in order to predict the future price or trends so that investments can be made accordingly.

3.4.1 Moving Average

The moving average is a comparatively simple technical stock analysis method that creates an updated average price and removes noise fluctuations. The average is taken over a specific period of time, like 3 days, 10 days, a month, or any time period the trader chooses. Frequently used applications of moving averages are to identify the direction in which the prices move, otherwise known as determining the trend.

3.4.2. RSI

The relative strength index (RSI) is an indicator that is used to measure stock conditions such as overbought and oversold conditions. Usually, when the RSI value crosses 30, it represents a bullish sign and when it decreases below 70, it represents a bearish sign. We can say that when RSI values are above 70, then a stock is overbought, and there may be a trend reversal. Similarly, an RSI value of less than 30 would represent an oversold stock. The RSI is represented as a graph that oscillates between 0 to 100. The relative strength index (RSI) is computed as:

$\mathbf{RSI}_{\text{step one}}$ =100−[100/ 1+ [Average gain / Average loss]]

$\mathbf{RSI}_{\text{step two}}$ = 100−[100 / 1 + [(Previous Average Gain×13) + Current Gain

---

−((Previous Average Loss×13) + Current Loss)

3.4.3. Williams %R

Williams %R, also called the Williams Percent Range, is a momentum indicator that moves between 0 and -100 and is used to measure both overbought and oversold levels. When the value falls below -20, it is an overbought stock. If the value is between -80 to -100, then we say the stock is oversold.

Wiliams %R = (Highest High−Close) / (Highest High−Lowest Low)

where

- Highest High → Highest price in the lookback period
- Close → Most recent closing price.
- Lowest Low → Lowest price in the lookback

4. Results

The system is trained for different company stocks using 70% of the data and tested using the remaining 30%. To showcase the different modules of the system, the training and trading screenshots are included, and the visualized output such as the loss function and trading function are displayed. Similarly, the stock analysis indicators are considered for the twenty companies and screenshots of those outputs are also displayed.

```
Epoch: 50       Reward: 2.850000      Balance: 835.090000     Loss: 0.010716
Epoch: 100      Reward: 0.000000      Balance: 864.210001     Loss: 0.006865
Epoch: 150      Reward: 0.000000      Balance: 969.370001     Loss: 0.014051
Epoch: 200      Reward: -1.549999     Balance: 920.889999     Loss: 0.003279
Epoch: 250      Reward: -1.860003     Balance: 998.139997     Loss: 0.002813
Epoch: 300      Reward: -9.940001     Balance: 884.650000     Loss: 0.003199
```

Fig 4.1. Training Function Output for AT&T Media

```
Day 48:         Buy at: $ 30.340000     Balance = $ 969.660000
Day 307:        Sell at: $ 37.619999    Balance = $ 1007.279999,
```

Fig 4.2. Trading Function Output for AT&T Media

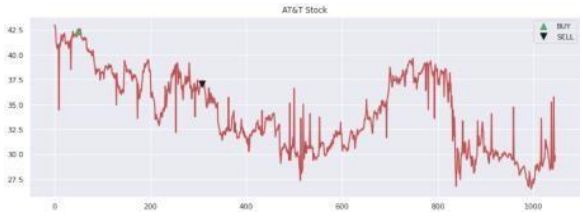

Fig 4.3: Visualized Stock Price & Actions

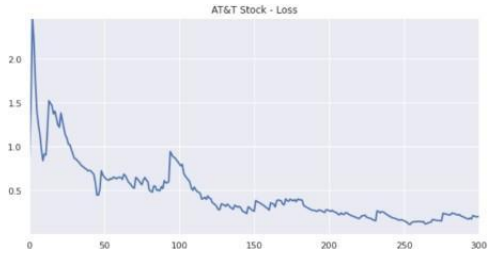

Fig 4.4: Visualized Loss

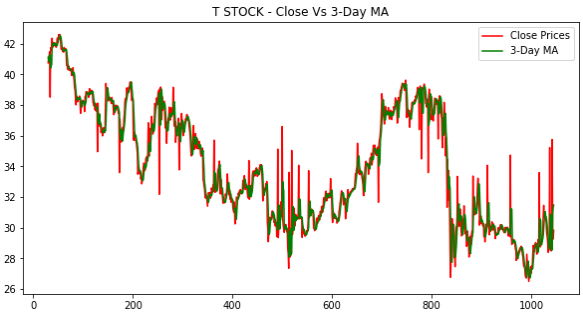

Fig 4.5: 3 Day Moving Average for AT&T Stock

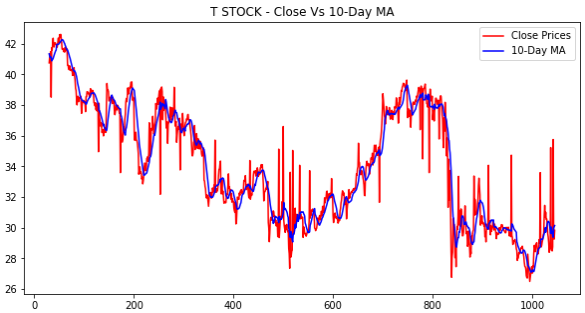

Fig 4.6: 10 Day Moving Average for AT&T Stock

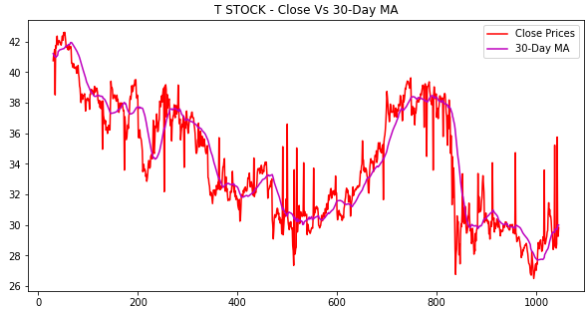

Fig 4.7: 30 Day Moving Average for AT&T Stock

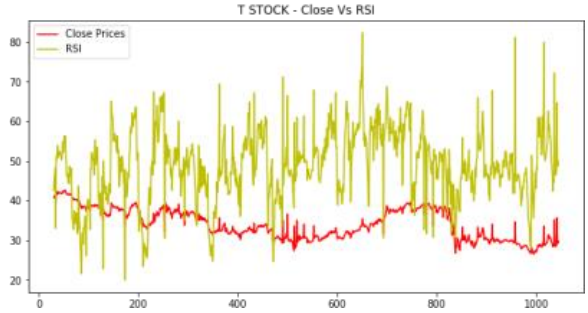

Fig 4.8: RSI - Period = 14 for AT&T Stock

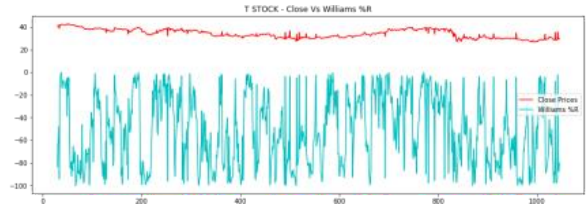

Fig 4.9: Williams %R - Period = 14 for AT&T Stock

5. CONCLUSION AND FUTURE WORK

A Deep Q-Network to perform stock market trading has been developed. The trader takes an action based on Reinforcement Learning techniques. The output of the trader is a set of actions - namely to buy, sell, or hold the stock. In order to train and test the system, data has been collected over a period of four years. The output is displayed in three ways -

1. The Profit/Loss values of each stock has been tabulated
2. The returns obtained from each stock have been calculated
3. The Loss function has been visualized in the form of a graph
4.

This system can serve as a foundation for a real-time portfolio management system. For a portfolio management system, the values of the risk indicators must also be taken into consideration along with the different price points of each stock. Additionally, a real-time portfolio will also track brokerage fees. As financial corporations do not release information on trading strategies created within the corporation, more research is needed to build a successful algorithmic trading system.

In terms of the Machine Learning implementation of the system, various algorithms such as Double Deep Q-Network or Long-Short Term Memory could be used to improve the performance. This can be compared using a loss function and calculating Profit/Loss values and subsequent returns.

REFERENCES

[1] Thibaut Theatea, Damien Ernsta, Montefiore Institute, University of Liege (Allee de la d´ecouverte 10, 4000 Li`ege, Belgium) - An Application of Deep Reinforcement Learning to Algorithmic Trading (April 2020)

[2] Rohan Pothumsetty, Department of Commerce, Christ University, Bengaluru, Karnataka – Application of Artificial Intelligence in Algorithmic Trading (May 2020)

[3] Irabaruta Jules – Effectiveness of Algorithmic Trading

[4] Kristian Bondo Hansen - The virtue of simplicity: On machine learning models in algorithmic trading (Big Data and Society, 2020)

[5] Ahmet Murat Ozbayoglua, Mehmet Ugur Gudeleka, Omer Berat Sezera - Department of Computer Engineering, TOBB University of Economics and Technology, Ankara, Turkey - Deep Learning for Financial Applications : A Survey (Article in Applied Soft Computing May 2020)

[6] Andres Arevalo, Jaime Nino1, German Hernandez, Javier Sandoval, Diego Leon, and Arbey Aragon - Algorithmic Trading Using Deep Neural Networks on High Frequency Data (Conference Paper – August 2017)

[7] PRIORITIZED EXPERIENCE REPLAY Tom Schaul, John Quan, Ioannis Antonoglou and David Silver Google DeepMind (ICLR 2016)

[8] @inproceedings{Theate2020, title={An Aplication of Deep Reinforcement Learning to Algorithmic Trading}, author={Theate, Thibaut and Ernst, Damien}, year={2020}}

[9] Albert Z Guo - Deep-Reinforcement-Stock-Trading - Github Profile

[10] Mike Wang – Deep Q-Learning Tutorial: minDQN – Medium Blog

[11] Druce Vertes – Deep Reinforcement Learning For Trading Applications – Alpha Architect

[12] Saeed349 – Deep Reinforcement Learning for Training – Github 77

[13] Windtoker – Trade_Learning_Network – Github

[14] XiaoYangLiu , TracyCuiYating, BruceYanghy-AI4Finance-LLC/NeoFinRL – Github

[15] Stefan Jansen – Machine Learning For Trading – Deep Reinfrocement Learning – Github

[16] Predicting Stock Prices Using Reinforcement Learning – AnalyticsVidhya

[17] Xiongpai Qin1,2, Huiju Wang1,2, Furong Li1,2, Jidong Chen3 Xuan Zhou1,2, Xiaoyong Du1,2, Shan Wang1,2 - Optimizing Parameters of Algorithm Trading Strategies
[18] Xavier Glorot, Yoshua Bengio - DIRO, Universite de Montreal, Montreal, Quebec, Canada - Understanding the difficulty of training deep feedforward neural networks
[19] Reinforcement Learning for Trading – AnalyticsVidhya
[20] Machine learning in algorithmic trading strategy optimization - implementation and efficiency - Przemysław Ryś*, Robert Ślepaczuk Quantitative Finance Research Group, Faculty of Economic Sciences, University of Warsaw
[21] ALGORITHMIC TRADING: EFFICIENCY OF LEOTRADE ALGORITHM STRATEGIZING - DISSERTATION BY TAKAVINGOFA MANZINI (M141885)
[22] Williams %R oscillator and Relative Strength Index (RSI): what's the difference? – Investopedia