Modifying Traditional Deep Reinforcement Learning with Sequential-Based Transformer Architecture for Stock Price Trading

Anindita Ghosh
Thomas Lord Department of Computer Science
University of Southern California
Los Angeles, CA, USA
ghosha@usc.edu

***Abstract*—Stock Price Trading is a lucrative hot spot in industries. Companies, farms, and even individual professionals obsess over investing in stock markets and getting the best returns. There are some classical, traditional stock-market trading practices but most of them fall short in maximizing returns. Algorithmic Trading [1] has been quite popular in recent times. The chances of better rewards following automated strategies are widely popular. Given the highly uncertain and volatile nature of stock markets [2], Reinforcement Learning has effectively helped leverage the best returns. This paper focuses on improving existing deep reinforcement learning (DRL) [3] by incorporating technical indicators and modern sequential models to enhance the action-value Q approximation, aiming to capture a more nuanced understanding of market dynamics. Our findings underscore possible potential for this integrated approach and highlight areas for further exploration. Our code can be found at https://github.com/ColeHoward/Trading_DRL/tree/master**

***Index Terms*—Deep Reinforcement Learning, Transformer Decoder, Stock Price Trading**

I. Introduction

Reinforcement Learning has been a valuable technique for stock trading environment because it can can learn from market conditions by interacting with the environment. The noise to signal ratio is high in stock data, making it difficult to model price trends and take appropriate actions based on these trends. This is where Deep Learning can be leveraged [4]. For any Reinforcement Learning agent, there's a constant trade-off : exploitation vs Exploration which becomes helpful for stock markets as all generated returns are subject to competition.

Over the years, deep learning models like RNN [5], LSTM [6] or GRU [7] have been used for time-series data analysis [8]. With more computational resources easily available, deep neural network architecture can be leveraged for pattern recognition. Stock data exhibit a stochastic nature so using deep learning models incredibly helps decide policies and trading strategies.

Deep Reinforcement Learning agents [9] interact sequentially with the environment, generalize information and accordingly update the states and learning parameters. They use neural networks with a framework of reinforcement learning and are most useful when the environment is very complex like stock market.

Reinforcement Learning techniques [4] have been used widely for Stock Price Trading but there are challenges. A Reinforcement Learning agent is heavily data-dependent so it performs poorly on small datasets since the variations in data are not too diverse making the learning agent unaware of actual market trends. Even though techniques like SMOTE [10] or inducing noise has been helpful in the past, original, authentic data is the best. The exploitation-exploration trade-off often helped the agent but there are situations when the agent is heavily penalized for more exploration.

Most Reinforcement Learning techniques are fed historical data which might not be related to current patterns of prices.

There are classical mathematical deductions to help denominate heavyweight terms like risk calculation, portfolio management, fraud detection but most of them are unaware of the highly stochastic nature of stock prices [11]. Both seasonal and non-seasonal trends in the stock price data make it necessary to have a tool that can use Deep Learning and Reinforcement Learning techniques for exploiting the nuances of the stock market.

We propose to use a Deep Reinforcement Learning agent on a sequential Transformer [12] architecture that will help to evaluate the varying trends of the stock price data to get a better estimate of the action-value function of the Q-network [1]. Our goal is to explore Deep Q Learning [13] and connect historical stock price trends to the future stock prices to maximize cumulative returns and improve existing strategies like the Trading Deep Q-Network (TDQN) algorithm. [1]. We have used technical indicators like MACD [14], EMA [15], RSI [16] while formulating the state of the learning agent. These indicators help assessing varying patterns of time-series data [17]. We have used regularization techniques like gradient clipping [18] or creative weight initialization methods like Xavier initialization [19] to induce uniformity to the deep neural network architecture.

II. Key Contributions

In this project, we took a multi-pronged approach to modifying Théate and Ernst's strategy.

1) **Enhanced Environment State Representation:** We expanded the environment state by adding several technical indicators (described in detail later), as well as past actions and rewards.
2) **Credit Assignment:** Reinforcement learning often faces challenges with delayed rewards, impacting algorithm stability. We address this using positional encodings and transformer decoder's self-attention for improved action-to-reward mapping.
3) **Refined Target Network Update Mechanism:** We enhance model stability by employing a gradual update process for the target network, using a decaying weighted averaging approach for smoother transitions and increased learning stability during training.

III. Background Description and Dataset

A. Deep Reinforcement Learning

Deep Reinforcement Learning optimizes sequential decisions, finding applications in robotics [20], natural language processing [21], machine translation [22], and gaining traction in financial trading.

Foundational Concepts: Deep Reinforcement Learning (DRL) combines reinforcement learning's adaptability with deep learning's pattern recognition, allowing models to autonomously learn optimal policies through trial and error guided by rewards and penalties. The core components that drive this process are:

- *Value Functions*: These functions estimate expected returns for states or actions, guiding the agent to maximize cumulative rewards.
- *Policies*: A policy guides an agent's actions based on its current state, with deterministic policies specifying clear actions and stochastic policies introducing probabilities for action selection.
- *States*: The state space encompasses all potential situations an agent may face, and the agent's precise perception of its state is vital for optimal decision-making.
- *Actions*: In a given state, the agent's decisions, guided by the policy, aim to maximize future rewards within the encompassing action space.

Challenges and Limitations: Deep RL faces challenges due to opaque decision-making, crucial in stock trading with high financial risks. Implementation stability is hindered by non-stationary stock prices, making learning an ever-changing task. Furthermore, delayed rewards often lead to wrong associations between actions and outcomes in various scenarios.

B. Stock Price Trading with Deep Learning

The volatile nature of stock prices necessitates the use of deep learning in trading. Models like RNN [5], LSTM [6], and GRU [7] are nearly unavoidable [23]. These models address gradient issues and demonstrated significant forecasting success in algorithmic trading environments [24].

Sequential models excel in swiftly analyzing vast datasets in stock trading, extracting insights from financial reports, news, and social media trends [25]. Integrated into algorithmic trading strategies, these models automate trades based on predefined criteria and market signals, adapting to changing conditions and reducing reliance on human intervention and emotional biases [26]. Additionally, in risk management [27], sequential models enhance portfolio robustness by continuously monitoring economic indicators and market updates, enabling investors to identify and address potential risks effectively.

C. An Application of Deep Reinforcement Learning to Algorithmic Trading

Théate and Ernst [1] have made significant strides in employing Double Deep Q Networks [13] for algorithmic stock trading, leveraging the principles of reinforcement learning. Their strategy centers around the approximation of the action-value function $Q_\theta(s,a)$ using a main network, complemented by a secondary target network $Q_{\theta^-}(s,a)$ which guides updates and stabilizes the training process. The target network periodically synchronizes with the main network's weights, allowing the model to capitalize on newly learned policies. The connection between these two networks is facilitated through the temporal difference error, formulated as:

$$L_{TD} = \frac{1}{2}\left[Q_\theta(s,a) - (r + \gamma Q_{\theta^-}(s',a'))\right]^2$$

To further refine this process, Théate and Ernst apply Huber Loss, defined as:

$$L = \begin{cases} L_{TD} & \text{if } \left|\sqrt{2 \cdot L_{TD}}\right| \le \delta, \\ \delta\left(\left|\sqrt{2 \cdot L_{TD}}\right| - \frac{1}{2}\delta\right) & \text{otherwise.} \end{cases}$$

In their approach, the environment state is defined through six key values: daily opening, closing, maximum, and minimum prices, daily trading volume, and the agent's current position. The action space is binary, consisting of 'long' and 'short' positions. A 'long' position entails buying stock with the expectation of a price increase, while a 'short' position involves borrowing and selling stock, anticipating a price decrease with the aim to repurchase at a lower price for profit.

D. Dataset Description

The primary dataset used is from Yahoo Finance [27], offering financial data and stock-related datasets. Data augmentation techniques such as signal shifting, filtering, and noise addition [1] are employed for additional data points. The dataset includes High, Low, Open, Close, Adjacent Close prices, and Volume. Different technical indicators [27], [28] are derived to capture stock price trends based on volume, volatility, and momentum [28]. To address Yahoo Finance's limited features, Google search trends [29] are incorporated to support stock token search patterns. The key technical indicators utilized are:

- **Exponential Moving Average (EMA):** Diminishes interference and highlights temporal patterns in the data.

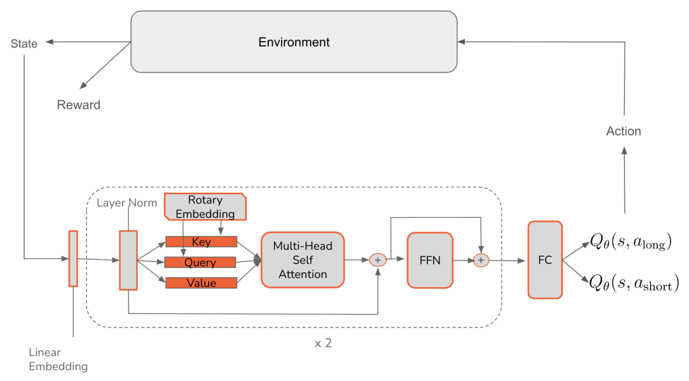

Fig. 1: Algorithm

- **Money Flow Index (MFI):** Assesses money flow in and out of a stock by considering price and volume, identifying overbought or oversold conditions.
- **Relative Strength Index (RSI):** Acts as a momentum gauge, determining whether a stock is overbought or oversold (low RSI) based on speed and extent of price changes.
- **Moving Average Convergence-Divergence (MACD):** Evaluates trend-following momentum by analyzing connection between slow and fast EMAs.
- **Commodity Channel Index (CCI):** Indicates trend by comparing current and historical price averages.
- **Ichimoku:** Utilizes components like Kijun-Sen, Senkou Span B/A, Tenkan-Sen, and Chiou Span in the Ichimoku Cloud indicator to identify trend direction, support and resistance levels.

IV. Model Architecture

We leveraged the Transformer model's proven success in sequence modeling to efficiently approximate the action-value function in reinforcement learning, capitalizing on its inherent strength in aligning actions with rewards.

Our design leverages the latest Transformer technology, specifically drawing inspiration from Apache's Mistral 7B model [30] for its efficiency and accuracy in complex tasks. A key modification includes adopting a full attention window, as opposed to sliding window attention, to better capture the temporal dynamics of stock market data in our application.

Our architecture largely retains the structure of the original Transformer decoder, with two key enhancements:

1) **Pre-Attention Layer Normalization:** We improved training efficiency by incorporating layer normalization before the self-attention mechanism, leading to faster and more stable convergence [31].
2) **Rotary Positional Embeddings (RoPE):** RoPE [32] enhances our model by directly incorporating positional information into the attention mechanism, crucial for capturing sequential dynamics in stock market data.

A. RoPE

We leverage RoPE as a pivotal component in modeling the stock market's state information. As described in [17], rotary embeddings represent an innovative approach in the domain of transformer models, offering a nuanced method of encoding positional information in a sequential data context.

Rotary embeddings differ fundamentally from traditional positional encoding methods. Instead of adding positional information as an external vector, RoPE integrates this information directly into the vector representation without loss of information. This integration is achieved through a rotation mechanism in the complex plane, where each dimension of the input embedding undergoes a position-specific rotation. Additionally, rather than transforming the input itself, only the queries and keys of the self-attention layers are transformed so that we can leverage the positional information to compute the attention weights, but retain all of the original information in the calculation of the values. see fig. 1.

Complex Representation: Each input embedding, x, is transformed into a complex number, where adjacent dimensions x_{2i} and x_{2i+1} are treated as real and imaginary parts, respectively.

Position-Specific Rotation: The embedding is then rotated in the complex plane by an angle determined by the positional encoding p. This is mathematically represented as $x'' = x' \cdot e^{i \cdot p}$, where x' is the complex representation of the embedding and i is the imaginary unit.

Extraction of Modified Embeddings: The resulting complex number is decomposed back into real and imaginary parts, which are then used as the modified embedding dimensions.

In the context of stock market analysis, rotary embeddings offer a distinct advantage in capturing the temporal dynamics of market states. Each time-step in our data set, representing a specific state of the stock market, is encoded with a unique rotational transformation. This approach allows the model to discern the relative positioning of market states in the temporal sequence, thereby enhancing its ability to understand and predict market trends based on historical data.

V. Algorithm Implementation

Our training procedure generally follows the standard RL approach whereby we gather experiences by repeatedly interacting with the environment using the current policy to the episode states, while simultaneously updating the model weights from previous experiences. Following each episode, we compute the training and validation metrics for later analysis. Through this process, we focus on balancing exploration and stability.

A. Exploration

Decaying Epsilon-Greedy: Rather than solely exploiting the current policy, epsilon greedy forces the agent to explore the environment randomly with some probability ϵ. As the agent gathers more experiences, ϵ is decayed according to the following formula:

$$\epsilon_i = \epsilon_{\min} + \frac{\epsilon_0 - \epsilon_{\min}}{e^{i/\delta}}$$

where ϵ_0 is the initial probability (set to 1), $\epsilon_{\min}$ is the minimum probability, i is the current iteration of the training procedure across all episodes, and δ is a scalar that defines the rate of decay for ϵ.

Opposite Action Exploration: In addition to storing the experience resultant of the agent's chosen action, we also generate and store the experience for the opposite action. This mechanism is designed to help the agent learn from a broader range of states and decouple previous versions of the model from future ones.

B. Stability

Decaying Target Network Updates: We build upon the strategy used in [1], where a target network is employed to stabilize learning. Unlike their method, which involves direct copying of weights from the main network to the target network, we introduce a more nuanced updating mechanism. Out method utilizes a decaying weighed average between the parameters of the main network (θ) and those of the target network (θ^-). The update rule is formalized as

$$\theta^- \leftarrow \tau_i\theta + (1-\tau_i)\theta^-$$

At the onset of training, both the main and target networks are initialized with random weights. During the early stages, the main network's weights undergo rapid changes as it learns from a diverse range of experiences. To accommodate this rapid learning phase, a higher τ_i is initially chosen. This setting allows the target network to more aggressively track the main network, ensuring that it quickly adapts to the significant updates occurring in the main network. As training progresses and the main network becomes more stable and reliable, the value of τ_i gradually reduces. More formally:

$$\tau_i = \tau_0^{i\alpha}$$

where α is treated as a hyper-parameter.

VI. Description of our experiments

A. Overall Performance

Training Overview: Our training process involved a comprehensive evaluation of each stock over 60 episodes, utilizing the computational power of an NVIDIA T4 GPU. On average, training for each stock was completed in approximately 75 minutes. The total computational resources expended over the entire set of experiments amounted to more than 300 Google Compute Engine units.

Data Utilization: The training dataset was compiled from historical stock data spanning from January 1, 2010, to January 1, 2018. To assess the model's performance and its generalization capability, we utilized a test set comprising data from January 1, 2018, to January 1, 2020. This division allowed us to train the model on a substantial historical dataset while testing its efficacy on recent and unseen market conditions.

Performance Metrics: In I, we detail the performance of our algorithm across various stocks, presenting a range of key financial metrics for each test set. These metrics offer a multifaceted view of the algorithm's effectiveness in different market scenarios.

Comparative Analysis: When compared to the benchmark results achieved by Théate and Ernst in II, our algorithm under-performed. Notably, the average Sharpe ratio - a measure of risk-adjusted return - across all stocks stood at -0.013 for our model, lower than the .45 achieved by the TDQN.

TABLE I: Portfolio Return Metrics

Stock	Sortino Ratio	Max Drawdown	Max Drawdown Duration	Volatility	ROI
AAPL	1.48462	31.951	37	0.38007	0.9343
AMZN	0.59741	39.17	60	0.31932	0.18144
BABA	-0.0632	67.41	153	0.58567	-0.41158
BIDU	-1.60657	66.666	314	0.36695	-0.58774
CCB	-0.70112	38.157	376	0.23345	-0.2108
DIA	-0.54534	27.288	193	0.16524	-0.10737
EWJ	0.97762	13.538	190	0.14049	0.11503
HSBC	2.50402	10.569	130	0.14869	0.43208
JPM	0.26278	26.947	181	0.27015	0.01339
KO	-1.11016	29.548	268	0.16945	-0.21396
NOK	0.23458	35.425	251	0.39271	-0.01771
PG	2.69889	9.597	6	0.18493	0.57696
QQQ	0.02512	26.419	80	0.21117	-0.0305
SIE	-0.84544	30.897	393	0.1995	-0.21291
SONY	-0.38768	36.854	129	0.37773	-0.23542
SPY	-0.74338	24.619	184	0.15412	-0.1325
TM	0.52422	17.336	179	0.18862	0.08161
TSLA	0.21439	57.936	276	0.55375	-0.11735
XOM	-0.8603	26.448	225	0.20629	-0.21951
Mean	0.14002	32.461	190.789	0.276	-0.008

TABLE II: Sharpe Ratio Comparison

Stock	Théate & Ernst	Team 1
DIA	0.684	-0.324
SPY	0.834	-0.468
QQQ	0.845	0.019
Nikkei	0.019	0.528
AAPL	1.424	1.218
AMZN	0.419	0.468
NOK	-0.094	0.169
SIE	0.426	-0.609
BIDU	0.080	-1.240
BABA	0.021	-0.077
SONY	0.424	-0.233
JPM	0.722	0.1634
HSBC	0.011	1.503
CCB	0.202	-0.483
XOM	0.098	-0.607
TSLA	0.621	0.16
TM	0.304	0.340
KO	1.068	-0.755
Mean	.45	-0.013

B. High Volatility Stocks

A critical aspect of any trading algorithm is its ability to manage risk, particularly in the context of high-volatility stocks. Stocks with high volatility present a unique challenge due to their rapid and significant price fluctuations, making them a stringent test case for the robustness and adaptability of our trading strategy.

Case Study - Tesla Stock: To evaluate our algorithm's performance in such a scenario, we specifically analyzed its behavior with Tesla (TSLA) stock, which is known for its high volatility. The results of this analysis are depicted in Figure 2, which illustrates the algorithm's performance on unseen Tesla stock prices.

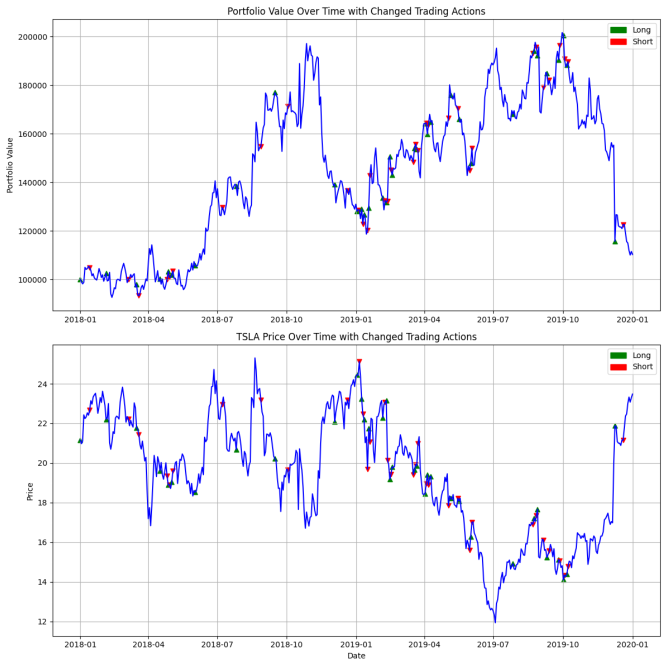

Fig. 2: Algorithm Performance on TSLA

Performance Analysis: Our observations indicate that while the algorithm shows promising performance for the majority of the episode, effectively navigating through the volatility, it is not immune to the challenges posed by such environments. This is particularly evident in the sharp decline in portfolio value observed during the final two months of the test period. This decline highlights a critical aspect of trading in high-volatility contexts – that gains accumulated over an extended period can be rapidly eroded by a few adverse market movements.

Lessons Learned and Strategy Adaptation: Rapid and drastic fluctuations in prices are exceedingly difficult to predict and manage. This underscores the importance of continuous risk assessment and the need for adaptive strategies that can respond quickly to sudden market changes. It also suggests potential areas for refinement in our algorithm, particularly in enhancing its capability to protect gains and mitigate losses during periods of extreme market volatility.

C. Stability

The project aimed to enhance algorithm stability, but analysis of training outcomes showed no improvement. The performance, exemplified by 3, mirrors results across all stocks. Despite Procter and Gamble's stable stock, our algorithm couldn't stabilize returns. The rewards from our target model poorly align with the model's overall returns, as evidenced by a low correlation coefficient of 0.0828 between cumulative rewards and portfolio value change.

VII. Future plans and conclusions

While the Transformer model has led to significant improvements in many fields, it requires a significant amount of data to properly train. Because training data, in this context, can

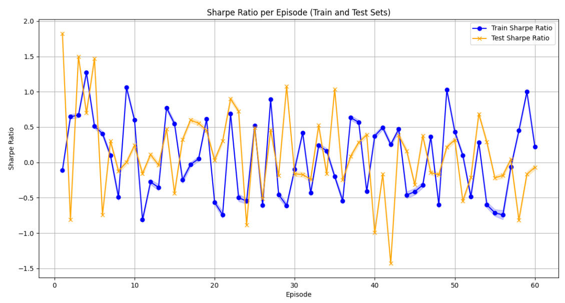

Fig. 3: Sharp Ratio across episodes for P&G

only be extracted from one particular stock, data quantity is a major issue. As we added more dimensions to the model via technical indicators, the data requirement only grew. Our compute resources were limited so we could not generate more data hence the approach did not demonstrate many improvements.

One interesting finding was that the distribution of actions for our algorithm were less extreme than the original TQDN algorithm. In a typical training session with the TQDN, action selection would often modulate between two extremes: a short or a long position at every time-step where the TQDN performed best. By contrast, our approach demonstrated a more even handed strategy where the actions were closer to even in quantity during training. While the cause of this discrepancy is still unclear, it would be interesting to explore which differences in the two approaches led to it and what this difference indicates about what each model learns during training.

We hypothesize that improvements in later works could occur through three approaches. First, if computational resources permit, data augmentation should be significantly increased. Next, pruning technical indicators can reduce the model's complexity and perhaps lose little information. Since technical indicators are derived from the price, much of the information can be implicitly derived through training purely on the prices, though some longer term information may still be useful to keep. Finally, to better approximate action-values, more complex approaches like TD-λ [33] may be explored.

References

[1] T. Théate and D. Ernst, "An application of deep reinforcement learning to algorithmic trading," *Expert Systems with Applications*, vol. 173, p. 114632, 2021.

[2] M. M. Kumbure, C. Lohrmann, P. Luukka, and J. Porras, "Machine learning techniques and data for stock market forecasting: A literature review," *Expert Systems with Applications*, vol. 197, p. 116659, 2022.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] O. B. Sezer, M. U. Gudelek, and A. M. Ozbayoglu, "Financial time series forecasting with deep learning: A systematic literature review: 2005–2019," *Applied soft computing*, vol. 90, p. 106181, 2020.

[5] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities." *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

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

[7] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[8] J. Huang, J. Chai, and S. Cho, "Deep learning in finance and banking: A literature review and classification," *Frontiers of Business Research in China*, vol. 14, no. 1, pp. 1–24, 2020.

[9] N. Vithayathil Varghese and Q. H. Mahmoud, "A survey of multi-task deep reinforcement learning," *Electronics*, vol. 9, no. 9, p. 1363, 2020.

[10] A. Li, M. Liu, and S. Sheather, "Predicting stock splits using ensemble machine learning and smote oversampling," *Pacific-Basin Finance Journal*, vol. 78, p. 101948, 2023.

[11] Y. Hu, K. Liu, X. Zhang, L. Su, E. Ngai, and M. Liu, "Application of evolutionary computation for rule discovery in stock algorithmic trading: A literature review," *Applied Soft Computing*, vol. 36, pp. 534–551, 2015.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[13] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[14] G. D. I. Anghel, "Stock market efficiency and the macd. evidence from countries around the world," *Procedia economics and finance*, vol. 32, pp. 1414–1431, 2015.

[15] D. S. Grebenkov and J. Serror, "Following a trend with an exponential moving average: Analytical results for a gaussian model," *Physica A: Statistical Mechanics and its Applications*, vol. 394, pp. 288–303, 2014.

[16] B. Anderson and S. Li, "An investigation of the relative strength index," *Banks & bank systems*, no. 10, Iss. 1, pp. 92–96, 2015.

[17] J. Jagwani, M. Gupta, H. Sachdeva, and A. Singhal, "Stock price forecasting using data from yahoo finance and analysing seasonal and nonseasonal trend," in *2018 Second international conference on intelligent computing and control systems (ICICCS)*. IEEE, 2018, pp. 462–467.

[18] J. Zhang, T. He, S. Sra, and A. Jadbabaie, "Why gradient clipping accelerates training: A theoretical justification for adaptivity," *arXiv preprint arXiv:1905.11881*, 2019.

[19] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[20] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The International Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, 2021.

[21] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," 2022.

[22] L. Wu, F. Tian, T. Qin, J. Lai, and T.-Y. Liu, "A study of reinforcement learning for neural machine translation," 2018.

[23] K. Khare, O. Darekar, P. Gupta, and V. Attar, "Short term stock price prediction using deep learning," in *2017 2nd IEEE international conference on recent trends in electronics, information & communication technology (RTEICT)*. IEEE, 2017, pp. 482–486.

[24] M. U. Gudelek, S. A. Boluk, and A. M. Ozbayoglu, "A deep learning based stock trading model with 2-d cnn trend detection," in *2017 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2017, pp. 1–8.

[25] D. Othan and Z. H. Kilimci, "Stock market prediction with new generation deep contextualized word representations and deep learning models using user sentiments," pp. 1–6, 2021.

[26] K. Cui, R. Hao, Y. Huang, J. Li, and Y. Song, "A novel convolutional neural networks for stock trading based on ddqn algorithm," *IEEE Access*, vol. 11, pp. 32 308–32 318, 2023.

[27] A. W. Li and G. S. Bastos, "Stock market forecasting using deep learning and technical analysis: A systematic review," *IEEE Access*, vol. 8, pp. 185 232–185 242, 2020.

[28] I. M. R. Malibari, N.; Katib, "Smart robotic strategies and advice for stock trading using deep transformer reinforcement learning," *Appl. Sci.*, vol. 12, p. 12526, 2022.

[29] I. Alsmadi, M. Al-Abdullah, and H. Alsmadi, "Popular search terms and stock price prediction," pp. 2107–2112, 2019.

[30] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7b," 2023.

[31] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T.-Y. Liu, "On layer normalization in the transformer architecture," 2020.

[32] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," 2023.

[33] R. S. Sutton and B. Tanner, "Temporal-difference networks," 2015.