IMPROVING DEEP REINFORCEMENT LEARNING FOR FINANCIAL TRADING USING DEEP ADAPTIVE GROUP-BASED NORMALIZATION

Angelos Nalmpantis *Nikolaos Passalis* *Avraam Tsantekidis* *Anastasios Tefas*

Aristotle University of Thessaloniki, Greece
{angelossn, passalis, avraamt, tefas}@csd.auth.gr

ABSTRACT

Deep Reinforcement Learning methods have provided powerful tools to train profitable agents for financial trading. However, the noisy and non-stationary nature of financial data often requires carefully designed and tuned input normalization schemes, since otherwise the agents are unable to consistently perform profitable trades. To overcome this limitation, in this work we propose a deep adaptive input normalization approach specifically designed to train DRL agents for financial trading directly using the raw price as input, without any additional pre-processing. The proposed method consists of two trainable neural layers that are designed to perform adaptive normalization, i.e., normalize the input observations after (implicitly) identifying the distribution that was used for generating them. Furthermore, instead of normalizing the whole input at once, the proposed approach performs group-based normalization, which allows for better capturing fine variations in the price trends. Despite being simple to implement and apply, the proposed method can lead to enormous improvements over existing the normalization methods, as demonstrated through the experiments conducted on two challenging FOREX currency pairs.

***Index Terms*—** Deep Reinforcement Learning, Financial Markets, Trading, Normalization

1. INTRODUCTION

Financial markets are the place where different assets are being traded and investors have the opportunity to make profitable decisions, given that they correctly predict the assets' movements. As a result, a large number of tools for financial trading has been developed, ranging from quantitative analysis [1], to more recent deep learning-based approaches [2–4], giving investors better tools to be able to perform profitable trades. A significant amount of research has originally focused on quantitative analysis, which uses mathematical and statistical models to better understand the behavior of various assets. However, the advent of automated trading, which allows for collecting an enormous amount of data in a short period of time, while also imposing strict time constraints for making prompt decisions, limits the usefulness of such quantitative methods.

The aforementioned observations have increased the use of Deep Learning (DL)-based approaches for financial trading, ranging from methods that attempt to predict the future behavior of various financial assets using supervised learning [2,3,5], providing useful signals for developing successful trading strategies, to fully automated reinforcement learning-based trading agents [4, 6–8]. The latter approaches allow for directly training an agent for the task at hand, i.e., learning how to perform profitable trades in stochastic and noisy financial environments, instead of training agents that solve proxy problems, such as predicting the short term price movements of an asset and using these prediction to develop hand-crafted strategies. Indeed, recent evidence suggests that such Deep Reinforcement Learning (DRL) approaches [6, 8], capable of combining the ability of DL to extract meaningful features from noisy environments and Reinforcement Learning (RL) to discover profitable policies, led to agents that can consistently perform profitable trades.

Despite its success on various tasks [9, 10], DRL often suffers from a significant drawback: it is notoriously difficult to train DRL agents, requiring careful tuning of the training hyper-parameters in order to ensure smooth convergence. As a result, a wide range of methods that can improve the convergence of the training process are usually applied, ranging from combinations of different empirical "tricks" [10], to reward shaping approaches [6, 11] that can further stabilize the training process. These inherent instabilities of DRL approaches are further exacerbated by the noisy nature of financial data, often requiring extensive testing of different input normalization approaches in order to select the most appropriate one for the task at hand. Indeed, as we also demonstrate in Section 3, even state-of-the-art DRL optimization methods often fail to converge for a wide range of input normalization strategies. It is worth noting that this behavior still emerges even when sophisticated hand-crafted stationary features are used for training. To better understand the effect of such *sub-optimal* normalization schemes we need to consider the following training dynamics: The last layers of a network are the first to adapt to the task at hand. However, if a sub-optimal

normalization scheme that suppresses useful features is employed, then the first layers will not be able to promptly recover these features, leading to converging into a sub-optimal local minimum. Indeed, recent evidence suggest that such critical learning periods exist in neural networks [12]. At the same time, most of the existing input normalization approaches, such as standardization, i.e., subtracting the mean value of the data and dividing the result with the standard deviation, are fixed and unable to promptly adapt to the task at hand, which could provide a way to partly alleviate this issue.

These observations lead us to the main research question of this paper: Is it possible to derive a trainable normalization scheme that will allow for quickly shifting the input into the appropriate region of the input space, stabilizing and accelerating the training process for DRL approaches? Using trainable input normalization schemes [13] can indeed lead to significantly improved performance for supervised Deep Learning applications. However, to the best of our knowledge, there has been no attempt to employ and further adapt such methods for use with DRL methods for financial trading tasks.

To this end, in this work we propose a deep adaptive input normalization approach, specifically designed for training DRL agents for financial trading using the raw input price as input. The proposed method consists of two trainable neural layers that are designed to perform *adaptive normalization*, i.e., normalize the input observations after (implicitly) identifying the distribution that was used for generating them. In this way, the proposed method allows for overcoming issues arising from the non-stationary nature of financial data. Furthermore, instead of normalizing the whole input at once, the proposed approach performs group-based normalization, which allows for better capturing fine variations in price trends. A group's normalization parameters are dynamically inferred using the statistics extracted from every group. In this way, a slightly different normalization scheme can be applied for each group. Finally, the proposed method is trained in an end-to-end fashion along with the rest of the model in order to solve a specific DRL task. This allows to promptly adapt the input of the DRL model to the task at hand, overcoming - to some extent - the aforementioned issues. The proposed method can be combined with any DRL method, i.e., both policy-based and value based approaches. The effectiveness of the proposed approach is demonstrated through experiments conducted on two FOREX currency pairs and compared to several other handcrafted, as well as state-of-the-art trainable normalization approaches, such as Layer Normalization [14]. Indeed, the proposed method significantly stabilizes the training process, as well as it also enables us to directly use raw prices for trading. In this way, it overcomes a significant obstacle of existing DL approaches (which typically require carefully designing and employing handcrafted stationary features). Finally, another interesting finding from our experiments was that the proposed method combined with simple models (MLPs) performed better than more powerful models, such as CNNs, regardless the used normalization approach and despite using less parameters. Therefore, we conjecture that using more powerful and adaptive normalization approaches might allow for reducing the overall complexity of the DL models, since distribution shift phenomena that might arise are reduced, allowing smaller models to perform as well as (or even better) than more complex ones.

The rest of the paper is structured as follows. First, the proposed method is analytically derived in Section 2, while the experimental evaluation is provided in Section 3. Finally, conclusions are drawn and future research directions are briefly discussed in Section 4.

2. PROPOSED METHOD

The proposed method is presented in this Section. First, essential information regarding financial trading and a detailed description of the proposed method are introduced. Then, the experimental approach employed for training and evaluating the DRL agents is provided.

Let $\mathbf{x}_t = [p_{t-N-1}, p_{t-N-2}, \ldots, p_t] \in \mathbb{R}^N$ be the input to a DRL agent at time t, where N is the length of the input and p_i is the price of an asset at time i. The goal of our agent is to learn a policy in order to perform profitable trades, i.e., hold a profitable position in the market. In this work, the agent supports three actions: a) "long position", when an asset is purchased, b) "short position", when a borrowed asset is sold, and c) "exit", where the agent exits the market. In order to support training DRL agent an appropriate simulation environment that returns the profit of each action performed by the agent was developed. Therefore, the agent can be trained to maximize a proxy to the Profit and Loss (PnL) metric, similar to the relevant literature on financial trading [4, 6]. The reward at each time step t is defined as:

$$r_t = \frac{(m_t - p_t)}{l_t} \cdot a_t - c_t, \tag{1}$$

where p_t is the current price of the asset, m_t is the mean value of the next ten prices of the asset, l_t is the last transacted price and a_t is the action that the agent has chosen, i.e., 1 for the long position, -1 for the short position and 0 when the agent has exited the market (so it received no profit or loss). Note that exiting the market does not terminate the training process and the agent has the opportunity to reenter in it. The employed reward is not equivalent to the actual PnL, since the average of the next 10 prices is used instead of the immediate next price. However, this modification allows for significantly improving the training stability, by filtering out potential noise. Note that during the evaluation the actual PnL is reported. The commission the agent pays at each time step is denoted by c_t in (1) and it is calculated as:

$$c_t = |a_t - a_{t-1}|c. \tag{2}$$

where a_t and a_{t-1} are the current and previous actions respectively and c is the commission fee. Again, c is typically set to a value slightly higher than the actual commission paid to account for price slippage and possible risks [4]. Note that the difference between the two actions is 0 when the agent holds the same position, 1 when the agent switches its position from 1 or -1 to 0 or vice versa and 2 when the agent switches from -1 to 1 or vice versa. Therefore, the agent is penalized when it frequently changes its position.

The proposed method employs two trainable layers: a) an adaptive shifting layer and b) an adaptive scaling layer. The way the proposed method works is summarized in Fig. 1. Before feeding the data into these two layers, the input data vector $\mathbf{x}_t \in \mathbb{R}^N$ is first reshaped into a tensor $\mathbf{x}_t^{(g)} \in \mathbb{R}^{g \times n}$, where g is the number of groups employed for the normalization and $n = \lceil N/g \rceil$ is the number of price values within each group. Note that each group contains successive price values. The tensor $\mathbf{x}_t^{(g)}$ can be appropriately padded with zeros if the length of input time series cannot be exactly divided with the number of groups. Then, a summary (average) representation is extracted from each input group as:

$$\boldsymbol{\mu}_t = \frac{1}{n}\mathbf{x}_t^{(g)}\mathbf{1}_n \in \mathbb{R}^g, \tag{3}$$

where $\mathbf{1}_n \in \mathbb{R}^n$ is an n-dimensional unit vector, i.e., a vector that contains only the value of 1. This summary representation is used to estimate the distribution from which the values of each group were sampled. This estimate is then used to calculate how the data from each group must be shifted in order to facilitate the task at hand. The shifting factor is calculated separately for each group by

$$\boldsymbol{\mu}(\mathbf{x}_t^{(g)}) = \mathbf{W}_1\boldsymbol{\mu}_t \in \mathbb{R}^g, \tag{4}$$

where the parameter matrix $\mathbf{W}_1 \in \mathbb{R}^{g \times g}$ is learned during the training process of the model and allows for taking into account the information extracted from all the groups before shifting the data. Therefore, the output of the first layer is calculated as:

$$\tilde{\mathbf{x}}_t^{(g)} = \mathbf{x}_t^{(g)} - \boldsymbol{\mu}(\mathbf{x}_t^{(g)}) \in \mathbb{R}^{g \times n}, \tag{5}$$

assuming that the values of $\boldsymbol{\mu}(\mathbf{x}_t^{(g)})$ are appropriately broadcasted (repeated n times). We did not explicitly redefine $\boldsymbol{\mu}(\mathbf{x}_t^{(g)})$ to avoid cluttering the used notation.

The second layer of the proposed method works similarly, but instead of shifting the input, it performs adaptive scaling on each input. Therefore, we first calculate a summary representation over the output of the previous layer as:

$$\boldsymbol{\sigma}_t = \frac{1}{n}\tilde{\mathbf{x}}_t^{(g)} \odot \tilde{\mathbf{x}}_t^{(g)}\mathbf{1}_n \in \mathbb{R}^g, \tag{6}$$

where the notation $\odot$ is used to refer to the element-wise (Hadamard product) multiplication between two matrices. Note that the quantity $\boldsymbol{\sigma}_t$ allows for estimating the variance of the original input $\mathbf{x}_t^{(g)}$ over each group. Then, we similarly estimate the scaling factor for each group as:

$$\boldsymbol{\sigma}(\mathbf{x}_t^{(g)}) = \mathbf{W}_2\boldsymbol{\sigma}_t \in \mathbb{R}^g, \tag{7}$$

where again the parameter matrix $\mathbf{W}_2 \in \mathbb{R}^{g \times g}$ is learned during the training process of the model and allows for taking into account the correlations between the groups before scaling the data. The final output of the proposed layer is obtained as:

$$\tilde{\tilde{\mathbf{x}}}_t^{(g)} = \tilde{\mathbf{x}}_t^{(g)} / (\boldsymbol{\sigma}(\mathbf{x}_t^{(g)}) + \epsilon) \in \mathbb{R}^{g \times n}, \tag{8}$$

assuming again that the values of $\boldsymbol{\sigma}(\mathbf{x}_t^{(g)})$ are appropriately broadcasted and ϵ is a small positive value used to ensure the numerical stability of the calculations. Then, the values of $\tilde{\tilde{\mathbf{x}}}_t^{(g)}$ can be flattened and fed to the network employed for learning the policy of the agent:

$$\mathbf{a}_t = f_{\mathbf{W}}(\tilde{\tilde{\mathbf{x}}}_t^{(g)}) \in \mathbb{R}^3, \tag{9}$$

where $\mathbf{W}$ are the parameters of the agent $f_{\mathbf{W}}(\cdot)$, while the vector $\mathbf{a}_t$ contains the probability estimation for each of the three actions supported by the agent (long, short, exit). The DRL agents employed in this work are directly trained using the Proximal Policy Optimization (PPO) algorithm [9]. The same model architecture, along with the proposed normalization layer, are used for both the actor and critic models, while no weight sharing is employed. The advantage is calculated as: $R_t(\lambda) - V_t$, where $R_t(\lambda)$ is the λ-return at time t and V_t is the value the critic estimate at time t. We used the TD(λ) return instead of the more conventional Monte-Carlo return that provided an unbiased but high-variance return [15]. On the contrary, TD(λ) is more appropriate for noisy tasks, such as financial trading, since the trade-off between variance and bias can more easily be controlled using the λ parameter. After running each episode and collecting the rewards needed for calculating the advantage, both actor and critic were updated. Note that the proposed layer is fully differentiable and can be trained in an end-to-end fashion along with the rest of the architecture using backpropagation. After the training process is completed the actor selects the action that corresponds to the highest probability. Finally, note that employing agent ensembles allows for taking into account the decision of multiple agents, increasing the decision making robustness [16].

3. EXPERIMENTAL EVALUATION

For developing the employed financial trading environment we used the Euro-Pound Sterling (EUR/GBP) and Australian Dollar-New Zealand Dollar (AUD/NZD) close prices. The total number of price samples is about 1.4 and 1.9 million, while no re-sampling was applied (every price corresponds to the minute close price). The last 400k and 500k prices

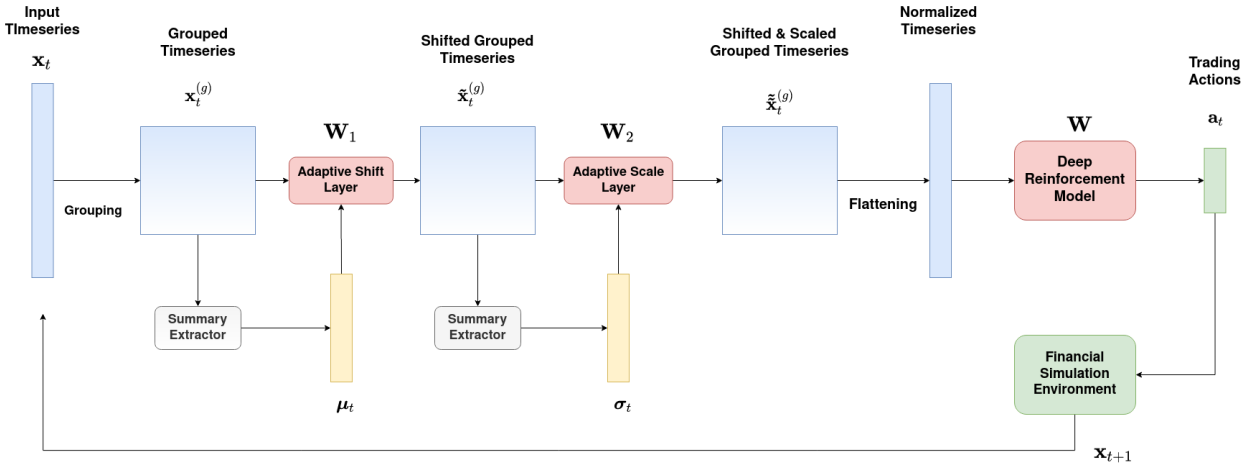

Fig. 1. Proposed Method: First, the input is grouped into a number of groups g that are normalized together. Then, the grouped observations are fed into the two proposed adaptive shifting and scaling layers. The output is then flattened and fed into the employed DRL model, which is used to perform an action to the financial simulation environment and obtain the updated observation.

were used for the evaluation, while the rest of them were used for training. We used different number of samples from the two pairs to evaluate and compare the proposed method in different settings.

The employed DRL agent consists of three hidden fully connected layers with 16 neurons and a final output layer with 1 output neuron for the critic and 3 output neurons for the actor. The PReLU activation function was used for the hidden layers [17]. Also, note that the action selected at the previous timestep is also fed to the network (by concatenating it with the output of the first fully connected layer). Providing the information of the action performed at the previous step to the network is crucial for performing profitable trades, since the position held by the agent affects the commission to be paid. The learning rate for the actor and critic was set to 10^{-4} and $3 \cdot 10^{-4}$ respectively. Each agent receives 120 prices in the input, while each simulation step leads to a forward move by 30 minutes. After an episode ends, both the actor and critic are updated using the AdamW algorithm [18], while a total of 10 batch-based updates are performed between each episode. The batch size was set to 32 for all the conducted experiments, while eligibility traces were employed to calculate the TD(λ) [19] (λ was set to 0.8, while the discount rate was set to $\gamma = 0.9$ for all the conducted experiments). The penalty for changing positions c was set to $2 \cdot 10^{-6}$, while the rewards were scaled by 10^4. Finally, to ensure that a good initialization is used for each of the employed agents, we performed multiple initializations for each agent and we chose the one that leads to the most balanced distribution between the available actions before initiating the training process [20].

For the proposed method we set the number of groups to $g = 4$, with each group having a length of $n = 30$. The learning rate was set to 10^{-10} and 10^{-8} for the two layers of the proposed method, since we experimentally found out that the proposed method leads to consistent gradients of quite large magnitude. As a result, a smaller learning rate is needed to ensure the smooth convergence of the resulting architecture. The parameters matrices $\mathbf{W}_1$, for the adaptive shifting layer, and $\mathbf{W}_2$, for the adaptive scaling layers, were initialized as being the identity matrices. Furthermore, increasing the proposed method's complexity, i.e., increasing the number of groups, did not lead to significant further improvements for the agents.

Finally, for all the conducted experiments we employed an ensemble of 7 agents, which ensures that a fair comparison is performed between the different evaluation approaches (since seven different agents have been trained, reducing the variance of the estimations). At the same time, the employed ensemble-based strategy led to significant performance improvements over using single agents, as we also experimentally demonstrate.

First, we evaluated five different baseline DRL agents trained using a) percentage changes (i.e., every price p_i in the input was replaced with $\frac{p_i - p_{i-1}}{p_{i-1}}$), b) differences (i.e, every price p_i in the input was replaced with $p_i - p_{i-1}$), c) min-max normalization, d) z-score standardization (i.e., subtracting the mean and dividing by the standard deviation), as well as e) Layer Normalization. The differences and percentage changes were standardized to ensure that the values will be in an appropriate range. To further validate the effectiveness of the proposed method, we also compared the proposed method to a CNN architecture that was also combined with different normalization schemes. The employed CNN consisted of 8 filters with size of 3 and was followed by the same number of fully connected layers as the proposed method.

The evaluation results are reported in Table 1, where the

PnL obtained by the ensemble agent that consists of the 7 individual agents is reported. Several interesting conclusions can be drawn from the reported results. First, note that normalization methods that do not lead to stationary features, i.e., min-max and standardization, collapse and do not allow for learning profitable policies. On the other hand, using more stationary handcrafted features, e.g., differences and percentage changes, leads to improved performance. The proposed method leads to the overall best results, significantly improving the PnL of the final ensemble model, since PnL increases by more than 150% and 20%, respectively, over the next best performing method. Furthermore, note that despite using a simpler MLP model, the proposed method performs better than the more powerful CNN model. This interesting observation might mean that part of the over-parameterization needed in modern DL architecture might actually be due to insufficient normalization of the input data. Using adaptive normalization structures allows for reducing the unecessary complexity, while also improving the accuracy of the models.

Table 1. Evaluation results using two FOREX currency pairs (EUR/GBP and AUD/NZD). The PnL of the ensemble model (composed of 7 individual models) is reported.

Method	EUR/GBP	AUD/NZD
Min-Max (MLP)	-0.946	-0.255
Min-Max (CNN)	-0.003	-0.251
Standardization (MLP)	-0.050	0.003
Standardization (CNN)	-0.013	-0.006
Percentage Changes (MLP)	0.112	0.546
Percentage Changes (CNN)	0.194	0.568
Differences (MLP)	0.119	0.505
Differences (CNN)	0.066	0.399
Proposed	$\mathbf{0.713}$	$\mathbf{1.267}$

Finally, we performed an additional ablation study to evaluate the impact of grouping the input and learning the parameters of the proposed adaptive normalization layer along with the rest of the model in an end-to-end fashion. We also included Layer Normalization in this study to evaluate the effect of learning a normalization scheme without applying any kind of grouping. It is worth noting that we also employed Batch Normalization [21] to normalize the raw input prices. However, this led to significantly worse results compared to Layer Normalization. The results are reported in Table 2, while the PnL curve on the EUR/GBP pair is provided in Figure 2. Indeed, as it is demonstrated, both training the parameters of the proposed method, as well as applying the proposed grouping operation, always leads to a more profitable policy, increasing both the individual agents' PnL, as well as ensemble model's PnL.

Table 2. Ablation study (comparing the effect of grouping and learning the parameters of the proposed method).

EUR/GBP

Method	Indv. PnL	Ensemble PnL
Layer Normalization	-0.286 ± 0.318	0.274
Proposed (no train)	0.124 ± 0.150	0.544
Proposed	$\mathbf{0.206 \pm 0.175}$	$\mathbf{0.713}$

AUD/NZD

Method	Indv. PnL	Ensemble PnL
Layer Normalization	0.363 ± 0.501	1.015
Proposed (no train)	0.377 ± 0.195	1.098
Proposed	$\mathbf{0.572 \pm 0.207}$	$\mathbf{1.267}$

"Layer Normalization" does not employ any kind of grouping, while "Proposed (no train)" just use the proposed normalization scheme without any end-to-end learning. Both the individual agent's PnL, as well the PnL of the ensemble model are reported.

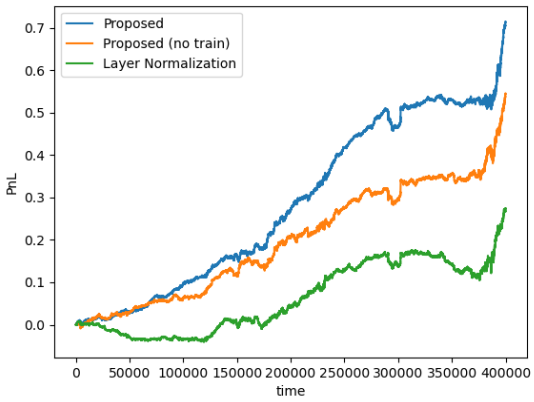

Fig. 2. Comparison of the obtained PnL (EUR/GBP pair).

4. CONCLUSIONS

In this work we presented a deep adaptive input normalization approach specifically designed for training DRL agents for financial trading using the raw input price as input. The proposed approach employs two trainable neural layers which can adaptively normalize the input by a) identifying the distribution from which the data were sampled, as well as b) using a group-based approach that can better capture the fine variations in the price. As we demonstrated through the conducted experiments on two FOREX pairs using the PPO algorithm, the proposed method performs better than other well-established input pre-processing methods, including one trainable normalization approach, validating the main hypothesis examined in this paper, i.e., using an appropriately-designed trainable normalization scheme leads to improved performance on trading tasks using DRL agents.

Acknowledgments: This work has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH - CREATE - INNOVATE (project code: T2EDK-02094).

5. REFERENCES

[1] Ganapathy Vidyamurthy, *Pairs Trading: quantitative methods and analysis*, vol. 217, John Wiley & Sons, 2004.

[2] Zihao Zhang, Stefan Zohren, and Stephen Roberts, "Deeplob: Deep convolutional neural networks for limit order books," *IEEE Trans. on Signal Processing*, vol. 67, no. 11, pp. 3001–3012, 2019.

[3] Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis, "Forecasting stock prices from the limit order book using convolutional neural networks," in *Proc. IEEE Conf. Business Informatics*, 2017, vol. 1, pp. 7–12.

[4] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 28, no. 3, pp. 653–664, 2016.

[5] Dat Thanh Tran, Martin Magris, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis, "Tensor representation in high-frequency financial data for price change prediction," in *Proc. IEEE Symposium Series on Computational Intelligence*, 2017, pp. 1–7.

[6] Konstantinos Saitas Zarkias, Nikolaos Passalis, Avraam Tsantekidis, and Anastasios Tefas, "Deep reinforcement learning for financial trading using price trailing," in *Proccedings of the IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 3067–3071.

[7] Zhengyao Jiang, Dixing Xu, and Jinjun Liang, "A deep reinforcement learning framework for the financial portfolio management problem," *arXiv preprint arXiv:1706.10059*, 2017.

[8] Amine Mohamed Aboussalah and Chi-Guhn Lee, "Continuous control with stacked deep dynamic recurrent reinforcement learning for portfolio optimization," *Expert Systems with Applications*, vol. 140, pp. 112891, 2020.

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

[10] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. AAAI Conf. Artificial Intelligence*, 2018.

[11] Pradyumna Tambwekar, Murtaza Dhuliawala, Lara J Martin, Animesh Mehta, Brent Harrison, and Mark O Riedl, "Controllable neural story plot generation via reward shaping," in *Proc. Int. Joint Conf. Artificial Intelligence*, 2019, pp. 5982–5988.

[12] Alessandro Achille, Matteo Rovere, and Stefano Soatto, "Critical learning periods in deep neural networks," *arXiv preprint arXiv:1711.08856*, 2017.

[13] Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis, "Deep adaptive input normalization for time series forecasting," *IEEE Trans. on Neural Networks and Learning Systems*, 2019.

[14] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton, "Layer normalization," 2016.

[15] Simone Parisi, Voot Tangkaratt, Jan Peters, and Mohammad Emtiyaz Khan, "Td-regularized actor-critic methods," *Machine Learning*, vol. 108, no. 8-9, pp. 1467–1501, 2019.

[16] Marco A Wiering and Hado Van Hasselt, "Ensemble algorithms in reinforcement learning," *IEEE Trans. on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 4, pp. 930–936, 2008.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. IEEE Int. Conf. Computer Vision*, 2015, pp. 1026–1034.

[18] Ilya Loshchilov and Frank Hutter, "Fixing weight decay regularization in adam," *CoRR*, vol. abs/1711.05101, 2017.

[19] Doina Precup, "Eligibility traces for off-policy policy evaluation," *Computer Science Department Faculty Publication Series*, p. 80, 2000.

[20] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem, "What matters in on-policy reinforcement learning? a large-scale empirical study," 2020.

[21] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.