Deep Adaptive Group-based Input Normalization for Financial Trading

Angelos Nalmpantis[a,*], Nikolaos Passalis[a], Avraam Tsantekidis[a], Anastasios Tefas[a]

[a]*Dept. of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece*

Abstract

Deep Reinforcement Learning (DRL) is among the state-of-the-art approaches for training agents for decision-making problems, such as financial trading. However, training DRL agents for such tasks is not always straightforward, since the noisy and non-stationary nature of financial data aggravate the already unstable training of DRL models. As a result, using DRL methods for such tasks require devoting significant effort for hyper-parameter tuning, as well as for designing the appropriate input pre-processing schemes. The latter is especially important, given the non-stationary nature of financial data, along with the ability of DRL agents to easily overfit the data, limiting their generalization abilities. In this work, we propose overcoming these limitations by introducing a differentiable, parameterized normalization scheme that allows for learning how the data should be normalized, along with the DRL model. More specifically, we propose dynamically normalizing the input according to various time-series statistics, which allows for adapting the model on-the-fly to the current mode of the data. At the same time, employing a segmentation scheme for extracting the statistics of the data allows for better capturing the variations of the input time-series and leading to more stationary representations. The proposed method is formulated as a series of neural layers that can be efficiently implemented using virtually any DL framework. The effectiveness of the proposed method against various normalization approaches is validated using two FOREX datasets and a state-of-the-art policy-based DRL approach.

1. Introduction

Different assets are constantly being traded in financial markets, giving investors the opportunity to take profitable decisions. Therefore, capturing and predicting an asset's movement is essential for investors. Powerful tools have been developed to help them construct their trading policy and obtain higher profits, while lowering the risk of the investments. In the past years, quantitative analysis was among the most prevailing methods used by investors, with the majority of research focusing on it. Quantitative analysis uses mathematical and statistical models providing predictions about possible future events of interest (Vidyamurthy, 2004). At the same time, the advent of automated trading provided important opportunities (Sarlin, 2013; Li and Jung, 2021), e.g., high-frequency trading (Gomber and Haferkorn, 2015; Passalis et al., 2020), allowing for further expanding the opportunities for using such models. However, the enormous amount of data that are available, along with strict time-restriction to take an action eventually led to the development of more sophisticated methods. Quantitative analysis was no longer a sufficient tool because of its limitations, since its reliance on handcrafted features and the need for human intervention limited its usefulness.

Indeed, Deep Learning (DL) has nowadays displaced such traditional tools to a large extent (Abe and Nakayama, 2018; Zhang et al., 2019b), also following recent advances in time-series analysis (de Mattos Neto et al., 2017; Zhang et al., 2019a). Several methods were proposed, ranging from classification models providing

*Corresponding author.

Email addresses: angelossn@csd.auth.gr (Angelos Nalmpantis), passalis@csd.auth.gr (Nikolaos Passalis), avraamt@csd.auth.gr (Avraam Tsantekidis), tefas@csd.auth.gr (Anastasios Tefas)

an asset's future movement to regression models trying to predict an asset's price in the near future (Zhang et al., 2019b; Tsantekidis et al., 2017, 2020). Yet, these methods still required hand-crafted policies in order to translate the output of a model into a trading policy that could be used in a real market environment. The integration of Reinforcement Learning (RL) in DL allowed for overcoming some of the aforementioned limitations. As a result, Deep Reinforcement Learning (DRL) has recently became the approach of choice for efficiently training agents for trading (Deng et al., 2016; Zarkias et al., 2019). DRL is capable for providing a profitable policy that can be directly used in financial markets, while promptly handling enormous amounts of data (given the appropriate computing resources).

Despite DRL's success in various fields, training DRL agents is often especially unstable, rendering the training process challenging. Indeed, many methods have been proposed to ensure the agent's convergence and enhance its performance ranging from employing several heuristics to increase the stability of the training process in inherently unstable estimators, such as Q-learning-based networks (Hessel et al., 2018), to reward shaping approaches (Tambwekar et al., 2019; Zarkias et al., 2019). It is worth noting that the noisy nature of financial data are further exacerbating these phenomena. As a result, developing DRL agents for trading often requires the exhaustive testing of many different pre-processing pipelines, such as normalization schemes and/or detrending methods, along with the rigorous tuning of the hyperparameters of the model.

To tackle the aforementioned problems, we propose incorporating a trainable normalization layer to DRL models to allow for learning how to properly shift and scale an asset's raw prices as we train the model. This layer aims to *pre-calibrate* the current input time-series in order to better match the distribution that the DRL model expects, minimizing in this way the effect of distribution shift phenomena and allowing for increasing the generalization capabilities of the resulting DRL agents. It is worth noting that the proposed method *does not* just learn some *static* normalization parameters. Instead, it learns to infer the normalization parameters that should be applied according to the statistics of current input window. To this end, the proposed method employs a group-based approach for extracting useful statistics from the input,

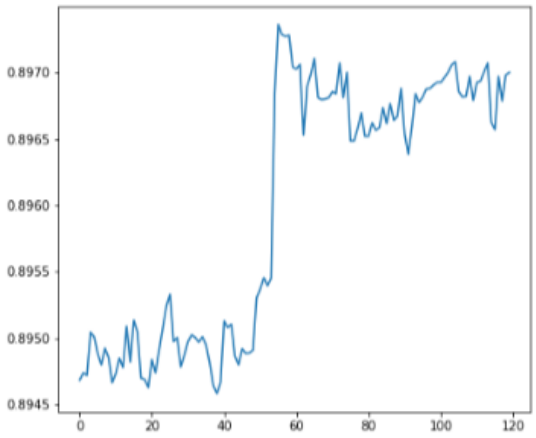

Figure 1: An example of a window of 120 close prices of EUR/GBP currency pair. Note that the data can drift away from their mean, requiring an appropriately designed method to handle the sub-groups that often exist within the same input window.

while also allowing for applying a different normalization scheme at different parts of the input time-series. To better understand why such approach can lead to more stationary representations of the input time-series we can consider the example shown in Fig. 1, where a window of 120 close prices of the EUR/GBP currency pair is depicted. Note that the time-series can often drift away from its mean, even for such short windows. Therefore, using an appropriately designed method to handle the sub-groups that often exist within the same input window can allow for extracting a more stationary time-series representation, which in turn can improve the performance of the subsequent DL model.

Even through, adaptive normalization methods have been used in the context of classification problems (Passalis et al., 2019), this is the first time, to the best of our knowledge, that a deep adaptive input normalization approach is adapted for use with DRL models and effectively used for developing trading DRL agents. Indeed, as we demonstrated in Section 2, applying such method on raw price time-series is not trivial, requiring several structural changes, such as employing segmentation schemes for extracting the input statistics and normalization the input values. The effectiveness of the proposed method is demonstrated using two FOREX cur-

rency pairs and a policy-based DRL algorithm. The proposed method is compared to several other baseline normalization methods, as well as to other trainable normalization approaches, e.g., (Ioffe and Szegedy, 2015; Ba et al., 2016). The proposed method not only allows for obtaining the most profitable policy, but also seems to exhibit greater robustness (compared to the rest of the evaluated methods) in a wide range of different model's hyperparameters.

The rest of the paper is as follows. In Section 2 we analytically derive the proposed method. Then, in Section 3 we describe the experimental setup, provide the experimental evaluation and discuss the obtained results. Finally, Section 4 concludes the paper and provides future research directions.

2. Proposed Method

The proposed method is presented in this Section. First, the necessary background information and notation regarding the task of financial trading are introduced. Then, the proposed method is presented in detail and discussed.

2.1. Background

Let $\mathbf{x}_t = [p_{t-N-1}, \ldots, p_{t-1}, p_t] \in \mathbb{R}^N$ be a vector that holds the last N close prices p_i of an asset at time t. An agent has to take a decision for the next time step $t+1$ based on the input $\mathbf{x}_t$. The agent's available actions are: a) "long position", when an asset is purchased, b) "short position", when a borrowed asset is sold and c) "exit" when the agent exits the market. To train a DRL agent a proper simulation of financial markets was developed. The agent's goal is to maximize its Profit and Loss (PnL) by taking and holding the supported positions. The PnL-based reward r_t at each time step t is calculated 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 an asset at time t, m_t is the average of the next 10 prices, l_t is the last transacted price, i.e., the price that the agent opened its position, a_t is the performed action and c_t is the applied commission. We used the mean value of the next 10 prices m_t, instead of the actual next close price p_{t+1}, since this smoothing allows for more easily training DRL agents (Tsantekidis et al., 2020). During evaluation (backtesting) we used the next price p_{t+1} to calculate the PnL that corresponds to an agent that performs trading in a real market.

Note that the action a_t takes one of the three values -1, 0, 1 corresponding to short, exit and long. When the agent exits the market, the simulation continues with the agent having the option to reenter in it. The obtained reward r_t can be scaled by a constant factor. Finally, the value of the commission c_t, that is paid at time t, is affected by the alteration of positions and is defined as:

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

with c being the commission fee. Note that the agent pays the double of the commission fee when it alters its position from long to short or vice versa, it pays c when one of the actions correspond to exit and it pays no commission when it held the same position as in the previous time step.

2.2. Group-based Input Normalization

The proposed layer comprises two sub-layers: a) *an adaptive shifting layer* and b) *an adaptive scaling layer*. The input vector $\mathbf{x}_t \in \mathbb{R}^N$ is first reshaped into a matrix $\mathbf{x}_t^{(g)} \in \mathbb{R}^{g \times n}$, where g is the number of groups and n is their length. If $g \cdot n \neq N$, then zero padding can be applied. Note that the sequence of prices is retained both within and out of the groups. The parameters of the normalization scheme that was applied to every group are calculated based on the statistics of each group. To this end, we first extract a summary representation, the average of every group, to estimate the distribution that produced the groups' values :

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

where $\mathbf{1}_n \in \mathbb{R}^n$ is a unit vector, i.e., a vector with n number of ones. The groups' shifting factors are calculated as:

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

with $\mathbf{W}_1 \in \mathbb{R}^{g \times g}$ being a parameter matrix holding the weights of the adaptive shifting layer. During the training, the agent learns how to exploit the distributions and extract information from every group to properly shift them. The output of the first layer is defined as:

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

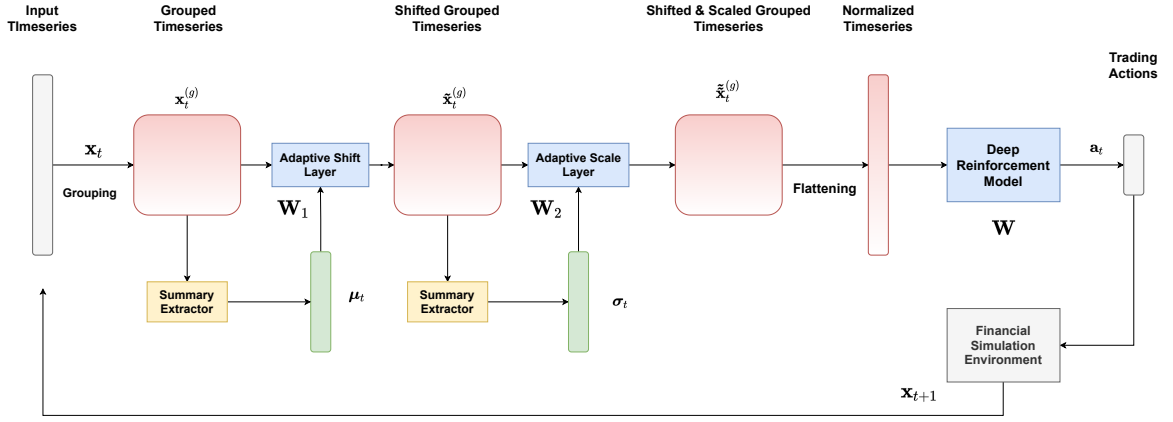

Figure 2: Overview of the proposed method. The input time-series is segmented into a number of groups g. The values that belong to each group are normalized together using the adaptive shifting and scaling layers. After normalizing the time-series using the proposed method, the output is fed into the employed DRL model. The whole architecture is trained in an end-to-end fashion to perform financial trading.

with $\boldsymbol{\mu}(\mathbf{x}_t^{(g)})$ being properly broadcasted to calculate the element-wise subtraction. Note that we did not redefine $\boldsymbol{\mu}(\mathbf{x}_t^{(g)})$ to avoid cluttering the notation.

The adaptive scaling layer works similarly to the first sub-layer with its goal being the proper scaling of the groups. First, we extract the standard deviation from every group:

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

where the notation $\odot$ is referring to the Hadamard product, i.e, the element-wise multiplication of two matrices. Then, the scaling factors are calculated as:

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

with $\mathbf{W}_2 \in \mathbb{R}^{g\times g}$ being a parameter matrix that holds the second layer's weights. Similarly, the extracted information from every group is used to individually calculate the scaling factor of every group. The second sub-layer's output is defined as:

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

Again, $\boldsymbol{\sigma}(\mathbf{x}_t^{(g)})$ is appropriately broadcasted without defining a new notation.

Therefore, the proposed method works by first splitting the input into groups and calculating some statistics regarding these groups, as described in Eq. (3) and (6). Then, these statistics are used to estimate the values to shift/scale the input, as described in Eq. (4) and Eq. (7), as well as in (5) and (8). These calculations involve a linear combination of the input statistics using the weights $\mathbf{W}_1$ and $\mathbf{W}_2$, which indicate how the statistics of each group contribute into the normalization of each other group. Note that these weights are adjusted during the training process and kept static during inference. However, in contrast to other normalization approaches that directly use the statistics of the input/each group without any kind of transformation, the proposed method can be considered *adaptive* since it can learn how to correlate the statistics of multiple input groups in order to normalize each of the groups *dynamically* during inference.

Finally, $\tilde{\tilde{\mathbf{x}}}_t^{(g)}$ is flattened and fed to the rest of the network. The agent's output is calculated as:

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

where $\mathbf{a}_t$ holds the estimation probability for each supported action (long, short and exit) produced by the agent $f_{\mathbf{W}}(\cdot)$ and W are the parameters that adjust the policy.

The DRL agent is trained with the algorithm Proximal Policy Optimization (PPO) (Schulman et al., 2017). Both

actor and critic used the proposed layer along with the same followed architecture without sharing any weights. The agent was trained using the advantage which is defined as $A_t = R_t - V_t$, where R_t is referring to the accumulative return and V_t to the critic's estimation at timestep t. We used eligibility traces to calculate R_t, as it provides a controllable trade-off between bias and variance instead of the more conventional and unbiased Monte-Carlo return (Parisi et al., 2019). After running each episode and collecting the rewards needed for calculating the advantage, the actor and the critic were updated. The actor was updated as:

$$\Delta \mathbf{W}_a = -\eta \frac{\partial \mathcal{L}_a}{\partial \mathbf{W}_a}, \tag{10}$$

$$\Delta \mathbf{W}_{1a} = -\eta \frac{\partial \mathcal{L}_a}{\partial \mathbf{W}_{1a}}, \tag{11}$$

and

$$\Delta \mathbf{W}_{2a} = -\eta \frac{\partial \mathcal{L}_a}{\partial \mathbf{W}_{2a}}, \tag{12}$$

where η is the used learning rate, $\mathcal{L}_a$ is the PPO-derived loss of the actor and the notation $\mathbf{W}_a$, $\mathbf{W}_{1a}$ and $\mathbf{W}_{2a}$ is used to refer to the parameters of the actor, along with the parameters of the two employed sub-layers. Note that since the employed layers are fully differentiable, the resulting architecture can be trivially trained in an end-to-end fashion using backpropagation. The critic's model is similarly trained. After training the actor, the action that corresponds to the maximum probability according to the learned policy is selected. Finally, if more than one model is trained for the same task, an ensemble model can be employed (Wiering and Van Hasselt, 2008) with the probabilities over each action being averaged.

3. Experimental Evaluation

The experimental evaluation is provided in this Section. First we briefly review the developed simulation environment, network architecture, and evaluation setup. Then, the experimental results are provided and discussed.

3.1. Experimental Setup

To evaluate the proposed method, we used the Euro - Pound Sterling (EUR/GBP) and the Australian Dollar - New Zealand Dollar (AUD/NZD) currency pairs. We sampled the minute close prices from 2017 to 2020 and

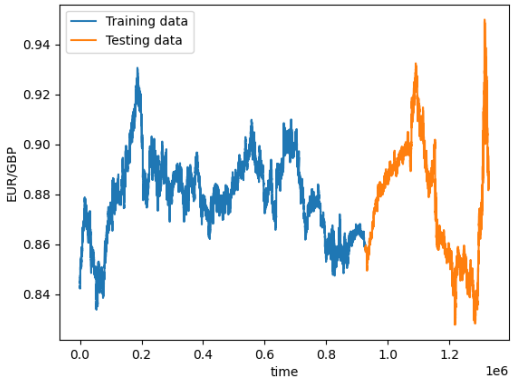

Figure 3: EUR-GBP data: Training data are denoted using blue color, while testing data are denoted using orange color. Figure best viewed in color.

2015 to 2020 respectively. We used more data for the AUD/NZD to evaluate each method in different settings, i.e., observe how much they can generalize the given data when overfitting is more likely to occur. The data used for training and testing data presented in Fig. 3 and 4 respectively.

The actor and critic models follow the same architecture composed of 3 fully connected layers with 16 neurons each. Models did not share any weights. The actor's output was consisted of three neurons corresponding to the available actions, while the critic's of one neuron. The activation function PReLU was used for the hidden layers (He et al., 2015), while the softmax activation was used for actor's final layer (Sutton and Barto, 2018). No activation function was used in the last layer of the critic. The input to the model comprised 120 sequential close prices. Following other DRL architectures for trading (Deng et al., 2016), late fusion was used for including the feedback from the previous performed action, allowing for introducing the information regarding the previous action closer to the last layers of the network. This can potentially increase the impact of this information to the network, ensuring that the model will consider the previous position in the market, which in turn, affects the commission that may be paid. The employed architecture is depicted in Fig. 5. During the training process, the agent's step in the environment was set to 30 min-

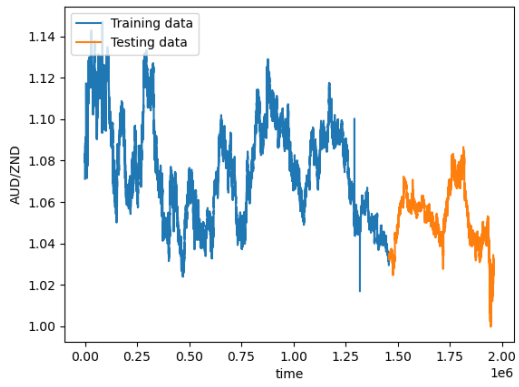

Figure 4: AUD-NZD data: Training data are denoted using blue color, while testing data are denoted using orange color. Figure best viewed in color.

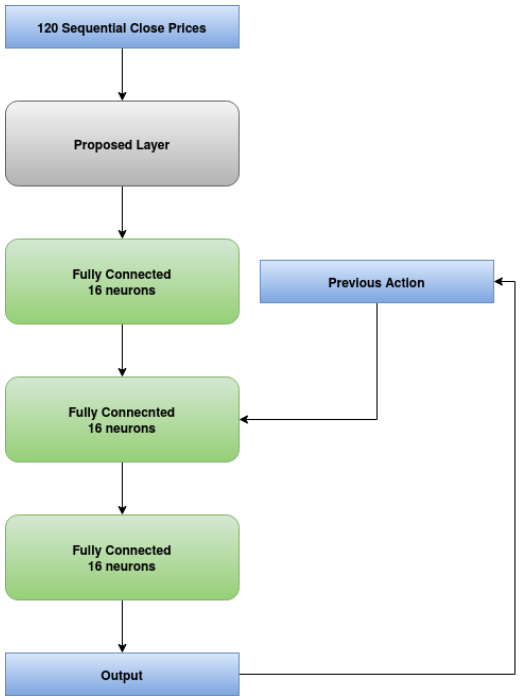

Figure 5: Network architecture that was employed for the experiments reported in this paper. Both the actor network and the critic network receive as input the input time-series, along with the action that was performed at the previous time step. The previous action is encoded using one-hot encoding.

utes. The learning rate was set to 10^{-4} and $3 \cdot 10^{-4}$ for the actor and the critic respectively. There was a total of 2 episodes and at the end of each one both models were updated with batches of size 32. The memory with the experiences that was split in batches was used 10 times for the updates before being completely deleted. The order of the batches retained the chronological sequence of the data. We used the AdamW optimizer (Loshchilov and Hutter, 2018) and the PPO algorithm (Schulman et al., 2017) for updating the networks. Eligibility traces were employed for calculating the accumulative return and the hyperparameters λ, which controls the trade-off between variance and bias, and γ (discount rate) were set to 0.8 and 0.9 (Precup, 2000). The commission c that was applied for changing positions was set to $2 \cdot 10^{-6}$. Finally, we ensured that the initialization of the weights was leading to a balanced action distribution before initiating the training (Andrychowicz et al., 2020).

The proposed layer used 4 groups with each one having length equal to 30 ($g = 4, n = 30$). We experimentally noticed that by increasing the method's complexity, i.e., increasing the number of groups, the agents' performance did not significantly improve. Moreover we observed that the returned gradients had large magnitude, thus the layer required a smaller learning rate. We set the first and second sub-layer's learning rate equal to 10^{-8} and 10^{-10} correspondingly. The initialization of the weights of each of the sub-layers (W_1 and W_2) was the identity matrix.

We evaluated different DRL agents using five different input normalization approaches:

1. Standardization, i.e., subtracting the mean value from the input and dividing by the standard deviation
2. Price Differences, i.e., every input's price p_i was replaced with $p_i - p_{i-1}$,
3. Percentage Changes, i.e., every price p_i of the input was replaced with $\frac{p_i - p_{i-1}}{p_{i-1}}$,
4. Layer Normalization (Ba et al., 2016), and
5. the proposed method.

Note that we standardize the percentage changes and differences to be in a more appropriate scale and ensure that every method's input will be in a similar range. The original purpose of Layer Normalization may differ, however

it is an interesting comparison with the proposed method. Both of them are trainable layers and have similar effect to the input with Layer Normalization bearing resemblance to the proposed layer when no grouping is applied ($g = 1, n = 120$). Also, we experimented with batch normalization (Ioffe and Szegedy, 2015) as a preprocessing tool at the model's input, but the agents' training was not successful. Therefore, we omitted reporting the corresponding results.

The conducted experiments with the evaluated methods were employed in 3 different environment setups regarding the way the agents are rewarded:

1. we used the actual next price p_{t+1} instead of the mean value of the next 10 prices m_t in the reward's equation 1,
2. we used the reward as it was defined in equation 1, and
3. the reward in equation 1 was scaled by a factor of 10,000

The first setup was the most unstable and challenging for an agent to be trained while using the average of the next 10 prices m_t smoothed the training process and benefited all of the methods. Finally, the third setup increased the loss' magnitude and led to faster algorithm's convergence. Evaluating in different training simulations shows the effectiveness of each method along with its robustness to hyper-parameter tuning.

Furthermore, apart from evaluating the individual PnL of each agent, we also employed an ensembling strategy, where we combined seven individual agents. This is a strategy that is known to significantly improve the performance of RL agents (Wiering and Van Hasselt, 2008), while also allowing for drawing more safe conclusions, since the variability that might exist due to differences in the initialization of each separate agent is minimized.

3.2. Experimental Results

The experimental results are reported in Table 1 and 2 for the two different currency pairs, where the ensemble's PnL, along with the individual agent's mean PnL are reported. The proposed method leads to highest PnL in all evaluated setups for the individual agents. Only a few combinations of hyper-parameter values and normalization methods lead to positive PnL, while the proposed

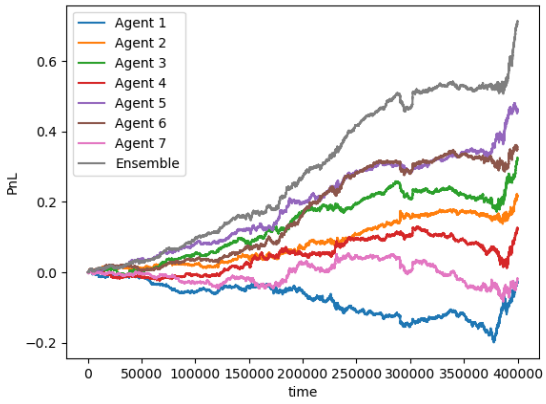

Figure 6: PnL curve for the proposed method (best setup) on EUR/GBP currency pair (calculated on the test set). The PnL curves for the seven individual agents are plotted, along with the PnL curve for the ensemble model.

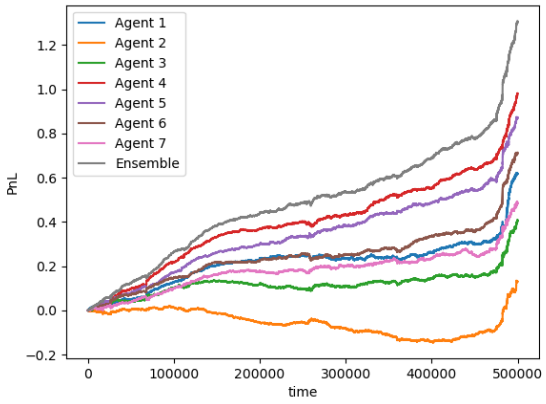

Figure 7: PnL curve for the proposed method (best setup) on AUD/NZD currency pair (calculated on the test set). The PnL curves for the seven individual agents are plotted, along with the PnL curve for the ensemble model.

method always allows for learning an agent that can perform profitable trades. This demonstrates the ability of the proposed method to reduce the effect of these hyper-parameters on the training of DRL models.

Examining the ensembles' efficiency, we observe that most methods benefit from it by obtaining higher prof-

Table 1: PnL of individual agents, as well as for the ensemble agent for different methods and setups trained on the EUR/GBP currency pair. PnL is calculated on the test set.

Normalization	**Indv. PnL**	**Ensemble PnL**
Setup 1		
Input Standardization	−0.119 ± 0.232	0.127
Price Differences	−0.507 ± 0.277	−0.513
Percentage Changes	−0.518 ± 0.300	−0.458
Layer Normalization	−0.486 ± 0.391	−0.523
Proposed	**0.050 ± 0.199**	**0.527**
Setup 2		
Input Standardization	−0.107 ± 0.157	0.261
Price Differences	−0.516 ± 0.326	−0.413
Percentage Changes	−0.527 ± 0.316	−0.379
Layer Normalization	−0.455 ± 0.440	−0.077
Proposed	**0.087 ± 0.250**	**0.550**
Setup 3		
Input Standardization	0.198 ± 0.187	0.679
Price Differences	−0.227 ± 0.203	0.119
Percentage Changes	−0.237 ± 0.203	0.112
Layer Normalization	−0.286 ± 0.318	0.274
Proposed	**0.206 ± 0.175**	**0.713**

its than their individual agents. In Fig. 6 and 7, we also provide the PnL plot for the proposed method, including both the plots for the seven individual agents, as well as the plot for the final ensemble model composed of the individual agents. The actions of the ensemble agents are selected after averaging the outputs of the individual agents and selecting the most probable action. The proposed method provides the best overall performance with its ensemble obtaining the highest profit among all the individual agents and their ensembles. Comparing the proposed method and input standardization, which is the second best-performing method, we obtained on average 84% higher profit with the EUR/GBP pair from the individual agents and 143% from their ensembles. Indeed, the proposed method outperformed the input standardization significantly with the first and second setup. Again, even

Table 2: PnL of individual agents, as well as for the ensemble agent for different methods and setups trained on the AUD/NZD currency pair. PnL is calculated on the test set.

Method	**Indv. PnL**	**Ensemble PnL**
Setup 1		
Input Standardization	−0.009 ± 0.283	0.543
Price Differences	−0.537 ± 0.629	−0.745
Percentage Changes	−0.780 ± 0.597	−0.592
Layer Normalization	−0.366 ± 0.468	0.132
Proposed	**0.394 ± 0.275**	**1.140**
Setup 2		
Input Standardization	0.246 ± 0.250	1.072
Price Differences	−0.545 ± 0.472	−0.558
Percentage Changes	−0.613 ± 0.432	−0.318
Layer Normalization	−0.585 ± 0.756	−0.009
Proposed	**0.600 ± 0.267**	**1.305**
Setup 3		
Input Standardization	0.518 ± 0.297	**1.280**
Price Differences	−0.132 ± 0.407	0.505
Percentage Changes	−0.103 ± 0.383	0.546
Layer Normalization	0.363 ± 0.501	1.015
Proposed	**0.572 ± 0.207**	1.267

Table 3: Effect of reward scaling on PnL of ensemble model (EUR/GBP currency pair).

Reward scaling factor	10^3	10^4	2×10^4
Input Standardization	0.480	0.679	0.684
Proposed	**0.562**	**0.713**	**0.739**

when ensemble is used, most methods are strongly dependent on the environment's settings, while the proposed method shows the highest robustness to hyper-parameter tuning and does not rely with the same degree on it, i.e., it is the least affected by the three different setups, obtaining a positive PnL for all of them.

Moreover, it is quite remarkable that individual agents with poor performance, when combined, are able to provide a profitable policy. Exemplary cases are the agents that used differences and percentage changes in the third setup, where the average individual agent had significant loss, while their ensembles managed to gain profit. Furthermore, layer normalization, which is the method that is most similar to the proposed one, was not very consistent with its performance among the different setups and its policy was not as profitable as the other methods.

Considering the three environment setups, we can conclude that the third setup offers the best agent's generalization with all ensembles having a profitable policy. Apart from the proposed method, all other methods obtained the highest profit with the third setup in both pairs. When the pair AUD/NZD was used, the proposed method led to the best PnL when combined with the second setup. This is probably due to the higher amount of data with which it was able to converge to profitable policy without needing the extra scaling of the reward whose purpose was the faster algorithm's convergence. Additional experiments were also conducted to further evaluate the impact of reward scaling on the behavior of the agents. The effect of reward scaling into the PnL of the ensemble agents is provided in Table 3 comparing the proposed method to the second best performing method (Input Standarization). Using higher reward scaling leads to more profitable agents, confirming the previous observations. Note that the proposed method always leads to the highest PnL, regardless of the employed scaling factor.

Table 4: Ablation study (comparing the effect of using grouping and learning the parameters of the proposed method). The EUR/GBP currency pair was used for the conducted experiments.

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

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

We also performed an ablation study evaluating the effect of training the layers employed in the proposed method. The results are reported in Table 4. Note that the proposed method is initialized to be equivalent to Layer Normalization (with groups). Therefore, we also included Layer Normalization in these ablation experiments. However, the proposed method is also capable of adjusting the way the data are normalized according to the input statistics, as the model is trained in an end-to-end fashion. Indeed, as it is demonstrated, training the parameters of the proposed method, as well as applying the proposed grouping operation, always lead to a more profitable policy, increasing both the individual agents' PnL, as well as ensemble model's PnL. We also evaluated the effect of using a different number of groups, as shown in Table 5. The optimal number of groups is probably related to the size of the input window. In this work, the optimal performance is obtained for groups that process 30 observations (4 groups). We hypothesize that the optimal value for this parameter is dependant to the nature of the used dataset, as well as to the size of the input.

4. Conclusions

In this work, a deep adaptive input normalization method, which aims to better exploit the various trends by employing a group-based approach, was presented. The proposed method uses two sub-layers that shift and scale

Table 5: Effect of number of groups on the individual agents and their ensemble model's PnL using the proposed method. The results are reported using the EUR/GBP currency pair

# of groups	Indv. PnL	Ensemble PnL
2 Groups	0.079 ± 0.226	0.498
4 Groups	**0.206 ± 0.175**	**0.713**
6 Groups	0.103 ± 0.224	0.471

the input which comprises sequential raw close prices from FOREX. To validate its effectiveness, we used the PPO algorithm and evaluated it in different conditions with two FOREX currency pairs. As it was demonstrated through the conducted experiments, the proposed method obtained the most profitable policy in comparison with other well-known and established methods. At the same time, the experimental evaluation also indicates that the number of blocks used in the proposed method can have a significant impact on the performance of the proposed method.

The proposed method paves the way for developing more sophisticated input normalization approaches for DRL agents for financial data, such as using differentiable and trainable stationary feature extraction layers (Tsantekidis et al., 2017), as well as using normalization schemes that do not discard the mode information, which can be potentially useful for further improving the performance of the employed DRL agents, such as using more advanced summary representations, such as Bag-of-Features-based representations (Iosifidis et al., 2014) and Fished-based encoding (Perronnin et al., 2010; Gosselin et al., 2014). Finally, the proposed method could be extended in order to be used for other tasks that might involve potentially non-stationary time-series as an input. This is further supported by recent evidence on a different, yet related domain, where it was demonstrated that adaptive normalization approaches can improve the forecasting performance for tasks that range from trend prediction (Passalis et al., 2019) to short-term load forecasting (Passalis and Tefas, 2020).

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).

References

Abe, M., Nakayama, H., 2018. Deep learning for forecasting stock returns in the cross-section, in: Pacific-Asia conference on knowledge discovery and data mining, pp. 273–284.

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., et al., 2020. What matters in on-policy reinforcement learning? a large-scale empirical study. arXiv preprint arXiv:2006.05990 .

Ba, J.L., Kiros, J.R., Hinton, G.E., 2016. Layer normalization, in: NIPS Deep Learning Symposium.

Deng, Y., Bao, F., Kong, Y., Ren, Z., Dai, Q., 2016. Deep direct reinforcement learning for financial signal representation and trading. IEEE Transactions on Neural Networks and Learning Systems 28, 653–664.

Gomber, P., Haferkorn, M., 2015. High frequency trading, in: Encyclopedia of Information Science and Technology, Third Edition. IGI Global, pp. 1–9.

Gosselin, P.H., Murray, N., Jégou, H., Perronnin, F., 2014. Revisiting the fisher vector for fine-grained classification. Pattern Recognition Letters 49, 92–98.

He, K., Zhang, X., Ren, S., Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE International Conference on Computer Vision, pp. 1026–1034.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D., 2018. Rainbow: Combining improvements in deep reinforcement learning, in: Proceedings of the AAAI Conference on Artificial Intelligence.

Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: Proceedings of the International Conference on Machine Learning, pp. 448–456.

Iosifidis, A., Tefas, A., Pitas, I., 2014. Discriminant bag of words based representation for human action recognition. Pattern Recognition Letters 49, 185–192.

Li, G., Jung, J.J., 2021. Dynamic relationship identification for abnormality detection on financial time series. Pattern Recognition Letters .

Loshchilov, I., Hutter, F., 2018. Fixing weight decay regularization in adam. arXiv preprint arXiv:1711.05101 .

de Mattos Neto, P.S., Cavalcanti, G.D., Madeiro, F., 2017. Nonlinear combination method of forecasters applied to pm time series. Pattern Recognition Letters 95, 65–72.

Parisi, S., Tangkaratt, V., Peters, J., Khan, M.E., 2019. Td-regularized actor-critic methods. Machine Learning 108, 1467–1501.

Passalis, N., Tefas, A., 2020. Global adaptive input normalization for short-term electric load forecasting, in: Proceedings of the IEEE Symposium Series on Computational Intelligence, pp. 1–8.

Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., Iosifidis, A., 2019. Deep adaptive input normalization for time series forecasting. IEEE Transactions on Neural Networks and Learning Systems .

Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., Iosifidis, A., 2020. Temporal logistic neural bag-of-features for financial time series forecasting leveraging limit order book data. Pattern Recognition Letters 136, 183–189.

Perronnin, F., Liu, Y., Sánchez, J., Poirier, H., 2010. Large-scale image retrieval with compressed fisher vectors, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 3384–3391.

Precup, D., 2000. Eligibility traces for off-policy policy evaluation. Computer Science Department Faculty Publication Series , 80.

Sarlin, P., 2013. Decomposing the global financial crisis: A self-organizing time map. Pattern Recognition Letters 34, 1701–1709.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 .

Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction. Second ed., The MIT Press.

Tambwekar, P., Dhuliawala, M., Martin, L.J., Mehta, A., Harrison, B., Riedl, M.O., 2019. Controllable neural story plot generation via reward shaping, in: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 5982–5988.

Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., Iosifidis, A., 2017. Forecasting stock prices from the limit order book using convolutional neural networks, in: Proceedings of the IEEE Conference on Business Informatics, pp. 7–12.

Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., Iosifidis, A., 2020. Using deep learning for price prediction by exploiting stationary limit order book features. Applied Soft Computing , 106401.

Vidyamurthy, G., 2004. Pairs Trading: quantitative methods and analysis. volume 217. John Wiley & Sons.

Wiering, M.A., Van Hasselt, H., 2008. Ensemble algorithms in reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 38, 930–936.

Zarkias, K.S., Passalis, N., Tsantekidis, A., Tefas, A., 2019. Deep reinforcement learning for financial trading using price trailing, in: Proccedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 3067–3071.

Zhang, Z., Zhang, G., Zhang, Z., Chen, G., Zeng, Y., Wang, B., Hancock, E.R., 2019a. Structural network inference from time-series data using a generative

model and transfer entropy. Pattern Recognition Letters 125, 357–363.

Zhang, Z., Zohren, S., Roberts, S., 2019b. Deeplob: Deep convolutional neural networks for limit order books. IEEE Transactions on Signal Processing 67, 3001–3012.