Noname manuscript No.
(will be inserted by the editor)

Forecasting Financial Time Series using Robust Deep Adaptive Input Normalization

Nikolaos Passalis · Juho Kanniainen · Moncef Gabbouj · Alexandros Iosifidis · Anastasios Tefas

Received: date / Accepted: date

Abstract Deep Learning provided powerful tools for forecasting financial time series data. However, despite the success of these approaches on many challenging financial forecasting tasks, it is not always straightforward to employ DL-based approaches for highly volatile and non-stationary time financial series. To this end, in this paper, an adaptive input normalization layer that can learn to identify the distribution from which the input data were generated and then apply the most appropriate normalization scheme is proposed. This allows for promptly adapting the input to the subsequent DL model, which can be especially important, given recent findings that hint at the existence of critical learning periods in neural networks. Furthermore, the proposed method operates on a sliding window over the time series allowing for overcoming non-stationary issues that often arise. It is worth noting that the main difference with existing approaches is that the proposed method does not just learn to perform static normalization, e.g., using a fixed set of parameters, but instead it adaptively calculates the most appropriate normalization parameters, signif-

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).

N. Passalis and A. Tefas
Computational Intelligence and Deep Learning Group
Department of Informatics, Faculty of Sciences
Aristotle University of Thessaloniki, Greece
E-mail: passalis@csd.auth, tefas@csd.auth.gr

J. Kanniainen and M. Gabbouj
Faculty of Information Technology and Communication
Tampere University, Finland
E-mail: juho.kanniainen@tuni.fi, moncef.gabbouj@tuni.fi

A. Iosifidis
Department of Engineering, Electrical and Computer Engineering
Aarhus University, Denmark
E-mail: ai@eng.au.dk

icantly improving the robustness of the proposed approach when distribution shifts occur. The effectiveness of the proposed formulation is verified using extensive experiments on three challenging financial time-series datasets.

Keywords Financial Forecasting · Deep Learning · Adaptive Normalization

1 Introduction

Deep Learning (DL) provided powerful tools for time series analysis, ranging from financial time series [5,6,17,27,32] to fault diagnosis systems [10,11] and speech analysis [15,37]. These neural network-based approaches typically use powerful models, such as Convolutional and Recurrent Neural Networks [12, 31,33], that are capable of extracting higher order temporal information from the data, leading to impressive results on various applications, ranging from time series forecasting [27], to developing deep reinforcement learning agents for trading [9]. However, despite the success of these approaches on many financial forecasting tasks, it is not always straightforward to employ DL-based approaches for highly volatile and non-stationary financial time series.

This often severely limits the effectiveness of DL-based approaches and leads to the need of developing and evaluating different normalization approaches that can overcome some of these limitations. Among the most widely used normalization approaches is z-score normalization: the data are standardized by subtracting the mean and dividing by their standard deviation. Despite its popularity for various financial time series analysis tasks using deep learning, z-score normalization suffers from significant limitations. For example, non-stationary time series cannot be effectively handled using this normalization scheme, leading to the development of more advanced normalization approaches, e.g., [21,23,26], and/or to the development of carefully designed feature extractors that are capable of extracting stationary features [22,34]. These approaches can indeed improve the accuracy of deep learning models. However, they crucially depend on handcrafted normalization and feature extraction schemes that must be manually and carefully designed, often employing several heuristics and ignoring the actual behavior of the data. At the same time, they provide no guarantee that they will be optimal for the task at hand.

One question that naturally arises from the aforementioned observations is why normalization seems to affect the performance of DL algorithms to this extent. To better understand this we need to consider two different phenomena that often arise. First, the employed normalization scheme significantly affects the generalization of the model on unknown data. For example, when the raw price is used as a feature, then a DL model operates on an fully unknown region of the input space when the price exceeds the range of price values used during the training. Methods such as sample-based standarization and instance normalization [14] can mitigate the effects of this to some extent, leading to improved generalization results. It is also worth noting that recent literature demonstrated that applying the appropriate input pre-processing

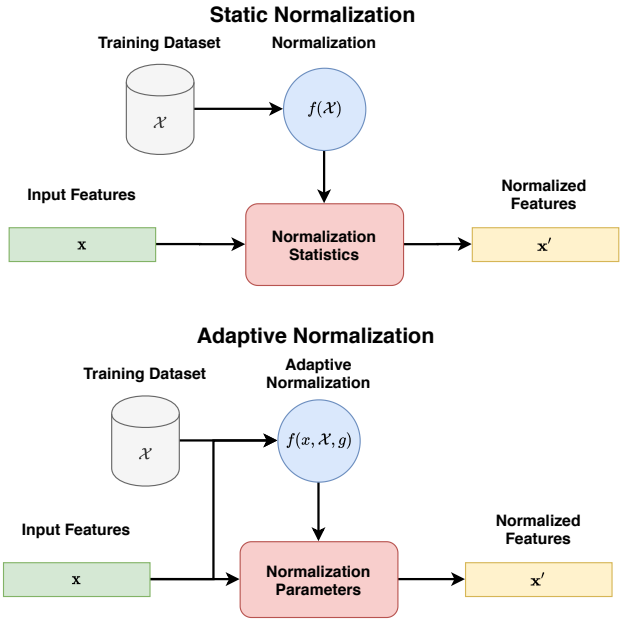

Fig. 1 Comparing static normalization approaches to the proposed adaptive normalization method, which takes into account not only the statistics of the dataset $\mathcal{X}$, but also the distribution from which the current input sample $\mathbf{x}$ is drawn.

schemes, such as including the normalized first order differences, emphasizing on specific characteristics of the data, e.g., zero-crossing rate, can lead to significant improvements in forecasting precision and trading performance [18, 19], further highlighting the importance of appropriately pre-processing the input time series. The second reason is intrinsic to the nature of the gradient descent-based optimization that is employed for training DL models. Even though the input features carry the same amount of information regardless the scaling applied to the individual features, the gradient descent's updates that are used for training the model are affected by the scale of the input features. Therefore, features with large magnitude have a larger effect on the direction of the descent, often leading the model toward different local minima than a model for which all the input feature were normalized in the same scale. The normalization therefore acts as a way to ensure that each feature will have "equal" chances for affecting the optimization.

In this paper, we propose a deep data-driven input normalization layer that is capable of *learning* how to perform normalization according to the actual distribution of the data and the task at hand. This allows for tackling both of the aforementioned issues. First, it provides a way for easily adjusting the importance of each feature *globally* for the model by accumulating the gradients for each of them. In this way, it provides a prompt way for quickly adapting the input to the subsequent DL model, which can be especially im-

portant, given recent findings that hint at the existence of critical learning periods in neural networks [3]. Furthermore, the proposed method operates on a sliding window over the time series allowing for overcoming issues that often arise due to the non-stationary nature of the data. It is worth noting that the main difference with existing approaches is that the proposed method *does not perform static normalization*, e.g., using a fixed set of parameters, but instead it *adaptively* calculates the most appropriate normalization parameters for each input sample *on the fly* by estimating the distribution from which it was generated, as shown in Fig. 1. In this way, the proposed method is capable of enabling deep learning models to handle non-stationary and multi-modal data, as well as efficiently generalize the encoded knowledge over diversified financial data, e.g., stocks with vastly different behavior. As we also demonstrate in this paper, the proposed method also leads to improved invariance to various distribution shifts. The proposed method is easy to implement and can be trained in an end-to-end fashion along with the rest of the parameters of the model, significantly increasing the forecasting accuracy over traditional normalization approaches. Apart from this, the proposed method allows for directly training deep learning models on raw data, without even applying any kind of normalization, significantly increasing the flexibility and ease of use of deep learning models.

The proposed method is composed of three sublayers. The first one learns how to shift each input data sample into the appropriate region of the space (shifting layer), the second one learns how to scale the features into the appropriate range (scaling layer), while the last ones learns which (and when) to suppress various input features (gating layer). Furthermore, the proposed method employs an auxiliary static normalization stream that allows for easily mitigating the effect of using learning rates that are not finetuned for each each individual task. In this way, it renders the proposed method more robust to less carefully selected hyper-parameters and allows for using it with the default hyper-parameters for a wide range of applications, as experimentally demonstrated in this paper. Again it should be noted that the proposed normalization scheme is adaptive, i.e., the normalization scheme adapts automatically to the distribution from which the input data were drawn. For example, if data from two different stocks were fed to the model, the normalization layer will normalize them differently in order to better facilitate the task at hand. This is in contrast with existing normalization methods, such as batch normalization [16], instance normalization [14], layer normalization [4] and group normalization [38], which are commonly used with deep learning models and are merely based on the statistics that are calculated during the training/inference. It is worth noting that using non-linear neural layers for adaptively normalizing the data is not straightforward, since neural layers require normalized data in the first place in order to function correctly. This issue is addressed in this work by first shifting and scaling the data, using two linear layers, that appropriately normalize the data before feeding them into a non-linear gating layer that can further suppresses the input features.

This paper is an extended version of our previous work presented in [25]. In this paper, an improved normalization formulation is provided by including an additional normalization stream that improves the stability of the proposed method (with respect to the selected learning rates), which was among the main difficulties that we observed when developing the proposed method. Therefore, we further extended our work by a) providing an improved robust formulation of the proposed approach that can make the proposed method easier to use and less sensitive the used hyper-parameter, b) extensively evaluating the proposed method using two additional datasets and several additional experiments (ablation studies, sensitivity analysis and qualitative evaluation), as well as conducting experiments using the improved formulation proposed in this paper, c) demonstrating how the proposed method can be used in ensemble models to further improve the forecasting accuracy for various financial tasks, d) evaluating the proposed method both for classification-based tasks, as well as on regression-based tasks, e) providing additional profit-based evaluations and f) better explaining the motivation for the proposed method by including a more detailed description of the issues that led us to developing the proposed method.

The rest of the paper is structured as follows. First, the proposed method is introduced in detail and discussed in Section 2. Then, the experimental evaluation is provided in Section 3. Finally, conclusions are drawn in Section 4.

2 Proposed Method

Consider a window of a time series that is composed of L d-dimensional measurement vectors denoted by $x = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_L\}$, where $\mathbf{x}_i \in \mathbb{R}^d$. Assuming that each measurement was generated by a Gaussian distribution, then the data can be effectively normalized as:

$$[\mathbf{x}']_l = ([\mathbf{x}]_l - \mu_l)/\sigma_l, \tag{1}$$

where the notation $[\mathbf{x}]_l$ is used to refer to the l-th element of a measurement vector $\mathbf{x}$ and μ_l and σ_l are the mean and standard deviation (of each separate variable) of the measurements, respectively. Note that if the previous assumption does not hold, i.e., the measurements were generated by a multimodal distribution, then employing z-score normalization can lead to sub-optimal results, especially if the statistics around each mode differ significantly from each other. In cases like this, data should be normalized in a *mode-aware* fashion.

In this work it is assumed that the measurements were generated by a Gaussian Mixture Model described by following density function:

$$p(\mathbf{x}) = \sum_{i=1}^{N} \phi_i \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), \tag{2}$$

where ϕ_i is the weight of the i-th Gaussian with mean $\boldsymbol{\mu}_i$ and covariance $\boldsymbol{\Sigma}_i$, which is denoted by $\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$. Note that even when this assumption does not

hold, the data can be similarly modeled using Kernel Density Estimation [28]. Therefore, in order to normalize the data we can identify the Gaussian from which they were generated and, then, the measurements can be normalized using the estimated mean and co-variance of the corresponding Gaussian. Note that even though this discards the mode information, it allows the subsequent model to generalize the knowledge that it has encoded regarding the behavior of time series.

The proposed method aims to *learn* how the measurements $\mathbf{x}_i$ of a window should be normalized by first appropriately shifting and then scaling them:

$$\mathbf{x}_s = (\mathbf{x} - \boldsymbol{\alpha}) \odot \boldsymbol{\beta}, \tag{3}$$

where $\odot$ is the Hadamard (entry-wise) multiplication operator. Note that by setting $\boldsymbol{\alpha} = E[x]$ and $\boldsymbol{\beta} = E[(x - E[x])^2]^{-1}$ the static sample-based z-score normalization is obtained. However, as it was already discussed, due to the non-stationary nature of the data, using fixed values for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ is not optimal for normalizing the input data, since the distribution of the data might significantly drift in time, which will in turn invalidate the previous choice for these parameters. Even if these parameters are calculated based on a recent windows of the time series, there is still no guarantee that they will be optimal for the task at hand, as discussed in the Introduction. Note that this issue is becoming even more important in the case of multi-modal data, e.g., when data from different stocks with different statistics (price levels, trading frequency, volumes, etc.) are used for training the model. The proposed method is capable of overcoming these limitations by dynamically learning how to estimate the normalization parameters and normalizing separately each input sample according to the statistics of the distribution in which it belongs. Therefore, for normalizing the input time series we employ an adaptive scheme:

$$\mathbf{x}' = (\mathbf{x} - \boldsymbol{\alpha}(x)) \odot \boldsymbol{\beta}(x), \tag{4}$$

where the functions $\boldsymbol{\alpha}(x)$ and $\boldsymbol{\beta}(x)$ are not fixed, but depend on the distribution in which the the current input sample belongs.

First, we extract a *summary representation* by averaging the L measurements:

$$\mathbf{s}_\alpha(x) = \frac{1}{L} \sum_{i=1}^{L} \mathbf{x}_i \in \mathbb{R}^d. \tag{5}$$

This summary representation will be used to identify the distribution to which the current time series belongs, allowing for applying the most appropriate normalization procedure. Then, we define the shifting function $\alpha(\mathbf{x})$ by employing a linear transformation of the extracted summary representation as:

$$\boldsymbol{\alpha}(x) = \mathbf{W}_\alpha \mathbf{s}_\alpha(x) + \mathbf{b}_\alpha \in \mathbb{R}^d, \tag{6}$$

where the notation $\mathbf{W}_\alpha \in \mathbb{R}^{d \times d}$ is used to refer to the parameter matrix of the first layer, which is responsible for shifting the measurements across each dimension, and $\mathbf{b}_\alpha \in \mathbb{R}^d$ is the corresponding bias term. Note that using a

linear transformation layer allows the method to be less sensitive to the scale of input data. As a result, the proposed method can be effectively trained in an end-to-end fashion without having to deal with stability issues, such as vanishing/exploding gradients and/or saturating the activation functions. We call this *adaptive shifting layer*, since this process corresponds to estimating the Gaussian from which the data were generated and then subtracting the corresponding mean.

After applying the adaptive shifting layer using (6), the data must be appropriately scaled using the shifting function $\boldsymbol{\beta}(\mathbf{x})$. Instead of using the previously calculated summary representation, we use the updated centered vectors and updating the summary representation before calculating $\boldsymbol{\beta}(\mathbf{x})$. Therefore, the updated summary representation used for defining the scaling function is calculated as:

$$\mathbf{s}_\beta(x) = \sqrt{\frac{1}{L}\sum_{i=1}^{L}(\mathbf{x}_i - \boldsymbol{\alpha}(x))^2} \in \mathbb{R}^d, \tag{7}$$

which corresponds to the standard deviation of the input features. Then, the scaling function is defined as an additional linear transformation over this representation:

$$\boldsymbol{\beta}(x) = (\mathbf{W}_\beta \mathbf{s}_\beta(x) + \mathbf{b}_\beta)^{-1} \in \mathbb{R}^d, \tag{8}$$

where the values of $(\mathbf{W}_\beta \mathbf{s}_\beta(x) + \mathbf{b}_\beta)$ are inverted element-wise. Note that by setting $\mathbf{W}_\alpha = \mathbf{W}_\beta = \mathbf{I}_d$, where $\mathbf{I}_d \in \mathbb{R}^{d\times d}$ is the identity matrix, and $\mathbf{b}_\alpha = \mathbf{b}_\beta = 0$ the proposed layer is initialized to perform sample-based normalization.

After applying these two transformation we obtain the normalization representation $\mathbf{x}'$ according to (4). Then, to allow for further suppressing irrelevant values we employ an additional attention-like gating mechanism [24,36] after calculating a new summary representation:

$$\mathbf{s}_\gamma(x) = \frac{1}{L}\sum_{i=1}^{L}\mathbf{x}'_i \in \mathbb{R}^d. \tag{9}$$

Then, the gating function is defined as a non-linear transformation over this summary representation allowing for performing gating on each of the shifted measurements:

$$\boldsymbol{\gamma}(x) = \text{sigm}(\mathbf{W}_\gamma \mathbf{s}_\gamma(x) + \mathbf{b}_\gamma) \in \mathbb{R}^d, \tag{10}$$

where $\text{sigm}(x) = 1/(1+\exp(-x))$, while $\mathbf{W}_\gamma \in \mathbb{R}^{d\times d}$ and $\mathbf{b}_\gamma \in \mathbb{R}^d$ are the weight matrix and biases of the gating layer, respectively. We call this layer *gating layer*. Note that in contrast to $\mathbf{W}_\alpha$ and $\mathbf{W}_\beta$ matrices, where using the identity matrix for their initialization has a direct interpretation, since it leads to sample-based standardization, there is no constraint on how the matrix $\mathbf{W}_\gamma$ should be initialized. So, in this is paper the standard Glorot uniform method is used for initializing this matrix [13]. Note that we did not observe significant differences when different methods were used for initializating $\mathbf{W}_\gamma$, compared to $\mathbf{W}_\alpha$ and $\mathbf{W}_\beta$, where using the identity matrix was crucial for

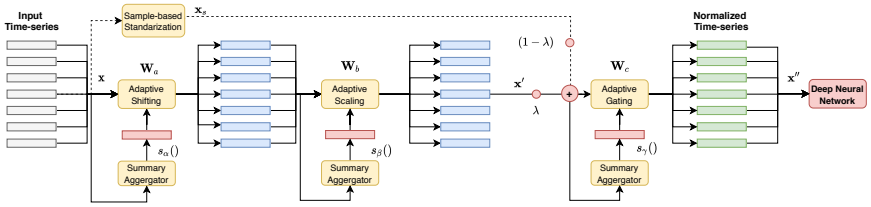

Fig. 2 Overview of the proposed method: The proposed method employs three normalization layers, as well as an auxiliary normalization stream that allows for increasing the learning stability and overall robustness of the proposed method to various distribution shifts, also demonstrated through the conducted experiments.

the performance of the proposed method. Also, using a non-linear activation function for this layer is not expected to cause any significant issue during the training, since the data are already appropriately standardized. However, if such issues indeed exist, then an additional trainable scaling factor can be used to appropriately scale the data. Therefore, the final normalized data are obtained as:

$$\mathbf{x}'' = ((\mathbf{x} - \boldsymbol{\alpha}(x)) \odot \boldsymbol{\beta}(x)) \odot \boldsymbol{\gamma}(x), \tag{11}$$

where note that $\boldsymbol{\beta}(\cdot)$ performs a linear scaling, while $\boldsymbol{\gamma}(\cdot)$ non-linear scaling. This process is illustrated in the lower branch of Fig. 2.

To increase the convergence speed and make the proposed method more robust to instabilities that might arise during the training process an auxiliary normalization stream is introduced. The employed auxiliary normalization stream performs fixed sample-based normalization and it contributes to the final normalization output by a factor of λ, where λ is a trainable parameter initialized to 0.5. In this way, even if the learning rate is not appropriately selected for the first two normalization layers, the resulting model can easily recover by altering the value of the parameter λ. On the other hand, if the sample-based normalization does not provide any useful information, then the value of λ will be increased, reducing the contribution of the employed auxiliary stream. Therefore, the input to the gating layer $\mathbf{x}'$ is calculated as a weighted sum of the adaptively normalized features and the introduced auxiliary stream as:

$$\mathbf{x}' = \lambda(\mathbf{x} - \boldsymbol{\alpha}(x)) \odot \boldsymbol{\beta}(x) + (1 - \lambda)\mathbf{x'}_s. \tag{12}$$

where $\mathbf{x'}_s$ is the sample-based standardized input features, as described in (3). This process is also illustrated in Fig. 2.

The parameters of the proposed data-driven input normalization layer are optimized using stochastic gradient descent:

$$\begin{aligned} &\Delta\Big[\mathbf{W}_\alpha, \mathbf{W}_\beta, \mathbf{W}_\gamma, \mathbf{b}_\alpha, \mathbf{b}_\beta, \mathbf{b}_\gamma, \lambda, \mathbf{W}\Big] \\ &= -\eta\Big[\eta_\alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}_\alpha}, \eta_\beta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_\beta}, \eta_\gamma \frac{\partial \mathcal{L}}{\partial \mathbf{W}_\gamma}, \eta_\alpha \frac{\partial \mathcal{L}}{\partial \mathbf{b}_\alpha}, \eta_\beta \frac{\partial \mathcal{L}}{\partial \mathbf{b}_\beta}, \eta_\gamma \frac{\partial \mathcal{L}}{\partial \mathbf{b}_\gamma}, \frac{\partial \mathcal{L}}{\partial \lambda}, \frac{\partial \mathcal{L}}{\partial \mathbf{W}}\Big] \end{aligned} \tag{13}$$

where the notation $\mathcal{L}$ is used to refer to the loss function used for training the network, while $\mathbf{W}$ refers to the parameters of the neural network that follows the proposed layer. It is worth noting that the proposed layer can be directly used on top of any deep learning network, allowing for fine-tuning the normalization process for any task (classification, regression, etc.) using the regular back-propagation algorithm. Separate learning rates are used for the parameters of the first, second and third sub-layers, i.e., η_α, η_β and η_γ. In contrast with existing adaptive input normalization formulations [25], which require carefully tuning each of the learning rates, employing the proposed auxiliary stream allows the proposed method to work quite well for a wide range of learning rates. The default parameters that are suggested for most tasks, as they were calculated through the experiments conducted in Section 3, are $\eta_\alpha = \eta_\beta = 10^{-3}$ and $\eta_\gamma = 10^{-1}$. Of course, fine-tuning these values for the task at hand can have a positive effect on the overall behavior of the resulting model. However, it was experimentally established that for most tasks the impact of using these, potentially sub-optimal, parameters is minimal.

3 Experimental Evaluation

The experimental evaluation of the proposed method is provided in this Section. First, the used datasets, evaluation metrics and network architectures are briefly introduced. Then, the experimental evaluation on one large scale limit order book dataset is provided, followed by extensive ablation and sensitivity studies. Finally, evaluation results on two more financial datasets consisting of cryptocurrency data are provided, further validating the effectiveness of the proposed approach.

3.1 Experimental Setup

Three different financial datasets were used for evaluating the proposed method: a) the FI-2010 dataset [22], b) a bitcoin dataset containing Bitcoin data at 1-min intervals [1], as well as c) a similar Ethereum dataset [2]. The FI-2010 dataset is a challenging large-scale dataset that contains limit order book data gathered from 5 different Finnish companies which were traded in the Helsinki Exchange (operated by Nasdaq Nordic): Kesko Oyj (retail industry), Outokumpu Oyj (steel industry), Sampo Oyj (insurance industry), Rautaruukki (steel industry), and Wartsila Oyj (manufacturing industry). For each time-step the 10 highest bid and 10 lowest ask orders prices and volumes were collected. The data were gathered over a period of 10 business days (1st June 2010 to 14th June 2010) and a total number of 4.5 million limit orders were collected. The preprocessing and feature extraction pipeline described in [22] was employed leading to extracting 453,975 144-dimensional feature vectors. The performance of the algorithms was evaluated using the following setup: The 15 most recently extracted feature vectors were used to predict the direction (up,

stationary or down) of the mean mid price after k time-steps, following the setup used in [22]. The proposed method was evaluated for the prediction of the mid price's direction for the next 10 and 50 time-steps, while the threshold for considering a stock stationary was set to 0.01% and 0.02% respectively. An anchored evaluation setup was used for all the conducted experiments [30], i.e., the first day was used for training and the next day for testing, then the first two days were used for training and the third one for testing, etc. The dataset contains a total of 9 such splits and we report two different evaluation metrics, macro-F1 and Cohen's κ [8], unless otherwise stated. When multiple splits are used for the evaluation, then the mean and standard deviation of the evaluation metrics is reported.

The second dataset is a Bitcoin dataset (BTC) [1], which contains USD to Bitcoin exchanges rates collected from 1-Jan-2014 to 22-Apr-2020. The open, high, low, and close price are reported at 1 minute intervals using data collected from the Bitstamp exchange. Furthermore, the volume of transacted Bitcoins and USD, as well as the volume-weighted average price is reported for this dataset. For all the conducted experiments using the BTC dataset, the data were downsampled to one-day candles, while the last year were used for testing and the remaining data were used for training the evaluated models. Average-based downsampling was employed, even though more sophisticated approached can be also used to retain more information. Features from the last 50 days were used for predicting the average close price of the next 10 days. The training and testing data do not overlap at any point, i.e., the 50 days that follow the data used for the training were discarded to ensure that the models are tested on completely unseen data. A similar approach was used for the Ethereum dataset (ETH), which contains data from 9-May-2016 to 16-Apr-2020. The minute data were similarly preprocessed for this dataset, while the testing was limited to the last 60 days, due to the smaller collection interval compared to the BTC dataset. Again, the dataset contains Ethereum to USD exchange rates, while the prediction target was set to be the average of the next 3 days, since a smaller number of data points are available for this dataset. Instead of directly predicting the direction of the exchange rates, the models were trained to regress the next close price (as a percentage change over the current one). Therefore, three different evaluation metrics were used for this dataset: a) Mean Absolute Error (MAE), R2-score (R2), also known as the coefficient of determination, as well as c) maximum prediction error (Max Error).

For all the conducted experiments we used three different network architectures, ranging from simpler fully connected architectures to recurrent and convolutional ones, as demonstrated in Fig. 3. Please note that the proposed method is used directly after the raw input features, while other normalization layers can be also used in place of the proposed method, as demonstrated in the conducted experiments. The simplest architecture that was used is based on a multilayer perceptron (MLP) consisting one hidden layer and one output layer. The flattened observation window is directly fed to the employed MLP. For example, for the FI-2010 dataset each window contains 15 144-

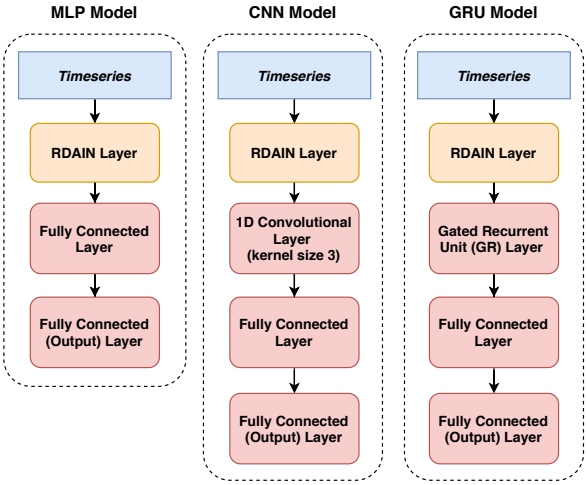

Fig. 3 The proposed method can be combined with different neural network architectures. In this paper, the proposed normalization layer was combined with a) an MLP model, b) a CNN model and c) a GRU model.

dimensional observations. Therefore, the MLP is fed with a flatten vector of dimension 2,160 (15×144). The hidden layer was composed of 512 neurons for the FI-2010 dataset. Dropout with rate 0.5 was used after the fully connected layer [29]. For the BTC and ETH datasets 32 and 128 neurons were used respectively, after performing cross validation experiments to identify the best number of neurons for each problem. The output layer consists of 3 neurons for the FI-2010 dataset (each neuron is responsible for predicting one of the three possible directions of the mid-price, i.e., down, stationary, and up), while one output neuron (responsible for predicting the percentage price change) was used for the BTC and ETH datasets. A convolutional architecture (CNN) was also used for the conducted experiments. The employed CNN is composed of one convolutional layer with kernel size 3, after which the output tensor is flattened and fed to the same MLP as described before. The number of filters was set to 256 for the FI-2010 and to 8 for the rest of the datasets. Similarly, the recurrent architecture that was used for the conducted experiments uses a Gated Recurrent Unit (GRU) [7] layer with 256 units (16 for the BTC and ETH datasets) before the fully connected layers. The ReLU [20] activation function was used for all the hidden layers employed for the FI-2010 dataset, while the *tanh* activation was used both for the fully connected and the output layers for the BTC and ETH datasets. The latter choice ensures that the output of the networks used for regressing the percentage changes in the close price are bounded in the -1...1 interval. Note that no price movement in the BTC and ETH datasets exceeds this interval. If the price movement are expected to exceed 100% of the previous price, then the output can be appropriately scaled.

The RMSProp algorithm with a learning rate of 10^{-4} for the FI-2010 and BTC datasets (10^{-3} for the ETC dataset) was used for training the models. The optimization ran for 30 epochs for the FI-2010 dataset, 100 epochs for the BTC dataset and 50 epochs for the ETH dataset. The categorical cross-entropy loss was used for predicting the direction of the mid-price, while the mean square error was used for optimizing the networks for directly predicting the percentage change of the close price. The data were sampled with probability inversely proportional to their label frequency for the FI-2010 dataset, since this is a highly unbalanced dataset [22]. Finally, the proposed method was also compared to a number of competitive approaches: a) using the standard z-score normalization (denoted by "Standardization"), b) subtracting the average measurement vector from each observation window (denoted by "Sample Average"), and c) standardizing the input values (z-score) using the statistics of each observation window (denoted by "Sample Standardization" or "Sample Std."). Furthermore, the use of Batch Normalization (denoted by "Batch Norm.") [16] and "Instance Normalization" (denoted by "Instance Norm.") [14] as the input layer of the network was also considered, despite that these methods were not designed to handle such tasks. Finally, the proposed method was also compared to the regular Deep Adaptive Input Normalization (abbreviated as "DAIN"), that is simply composed of the three normalization layers without the proposed auxiliary stream, to better demonstrate the benefits of the proposed robust formulation. Note that proposed method is also abbreviated as "RDAIN" (Robust Deep Adaptive Input Normalization).

3.2 FI-2010 Evaluation

The results on the FI-2010 dataset using the first prediction horizon (next 10 timesteps) are reported in Table 1. The last four days of the FI-2010 dataset were used for the evaluation reported in Tables 1 and 3, while the first five days were used for the remaining ablation studies and sensitivity analysis experiments. We used these two disjoint splits to ensure that validation set used for the hyper-parameter selection is not used during the evaluation of the models. Also, forecasting results for three different model architectures are reported, i.e., MLP, CNN and RNN. Several interesting conclusions can be drawn from the reported results. First, note that the best performing model is affected by the employed normalization method. For example, when standard normalization methods are used, the CNN model performs better when combined with standardization or sample average normalization, while the MLP model seems to perform better when combined with sample standardization and batch normalization. It is worth noting that the feature extracted for the FI-2010 dataset, unlike the rest of the datasets, already encode significant temporal information, as described in [22], rendering the use of models capable of capturing the temporal information, such as CNNs and RNNs, less crucial. Indeed, the plain MLP seems to lead to the best classification performance when combined with the proposed approach. Finally, the effectiveness of the

Table 1 FI-2010 evaluation: Evaluating the performance of different normalization methods for the first prediction horizon (next 10 timesteps)

Model	Method	F1-score	κ
MLP	Standarization	56.79 ± 0.47	0.3536 ± 0.0071
MLP	Sample Average	56.21 ± 1.80	0.3423 ± 0.0235
MLP	Sample Standarization	63.86 ± 1.31	0.4468 ± 0.0188
MLP	Batch Normalization	56.26 ± 0.51	0.3455 ± 0.0085
MLP	Instance Normalization	60.77 ± 0.92	0.4006 ± 0.0114
MLP	DAIN	69.50 ± 0.35	0.5346 ± 0.0051
MLP	Proposed	$\mathbf{69.94 \pm 0.55}$	$\mathbf{0.5413 \pm 0.0082}$
CNN	Standarization	57.27 ± 1.12	0.3563 ± 0.0183
CNN	Sample Average	58.41 ± 0.63	0.3663 ± 0.0101
CNN	Sample Standarization	60.44 ± 1.02	0.3972 ± 0.0167
CNN	Batch Normalization	55.93 ± 1.08	0.3372 ± 0.0182
CNN	Instance Normalization	60.17 ± 0.69	0.3917 ± 0.0120
CNN	DAIN	48.87 ± 3.07	0.2488 ± 0.0372
CNN	Proposed	$\mathbf{66.77 \pm 0.48}$	$\mathbf{0.4926 \pm 0.0086}$
RNN	Standarization	55.48 ± 1.29	0.3285 ± 0.0207
RNN	Sample Average	54.24 ± 1.38	0.3030 ± 0.0221
RNN	Sample Standarization	59.62 ± 0.91	0.3850 ± 0.0151
RNN	Batch Normalization	54.85 ± 1.32	0.3193 ± 0.0194
RNN	Instance Normalization	58.33 ± 0.84	0.3649 ± 0.0135
RNN	DAIN	67.86 ± 0.52	0.5102 ± 0.0085
RNN	Proposed	$\mathbf{67.90 \pm 0.80}$	$\mathbf{0.5106 \pm 0.0122}$

Table 2 FI-2010 Evaluation: Comparing the proposed model without any training ('no train'), using a default choice for the learning rates ("Default") and a carefully tuned selection of learning rates using a validation set ("Optimal").

Model	Method	F1-score	κ score
MLP	No train	61.77 ± 2.58	0.4121 ± 0.0383
MLP	Default	68.60 ± 1.60	0.5182 ± 0.0232
MLP	Optimal	$\mathbf{68.65 \pm 1.67}$	$\mathbf{0.5194 \pm 0.0245}$
RNN	No train	56.56 ± 2.18	0.3347 ± 0.0329
RNN	Default	66.38 ± 1.93	0.4848 ± 0.0282
RNN	Optimal	$\mathbf{66.72 \pm 1.92}$	$\mathbf{0.4896 \pm 0.0279}$
CNN	No train	56.07 ± 2.41	0.3284 ± 0.0366
CNN	Default	64.42 ± 1.90	0.4554 ± 0.0277
CNN	Optimal	$\mathbf{65.31 \pm 1.90}$	$\mathbf{0.4667 \pm 0.0279}$

proposed method is demonstrated in Table 1, since for all the conducted experiments the proposed method leads to the overall best results compared to the rest of the evaluated methods.

Note that in some cases, such as for the CNN, the performance gap with the CNN is large, while for other, such as for the RNN, the differences are quite smaller. This demonstrates the unstable nature of DAIN, which can lead to suboptimal results when the learning rates for its three layers are not carefully

tuned (note that for all the conducted experiments we used the learning rates selected through the ablation study provided in Section 3.3). Indeed, even though the DAIN layer can be almost always appropriately tuned to achieve competitive results, the proposed robust formulation of DAIN leads to significantly better stability, without further finetuning. This is also demonstrated in the results reported in Table 2, where the ability of the proposed method to work without any learning rate fine-tuning was evaluated (using a default learning rates of $\eta_\alpha = \eta_\beta = 10^{-3}$ and $\eta_\gamma = 10^{-}1$). Three different variants of the proposed method were evaluated: a) using no training ('no train'), using a default choice for the learning rates reported above ("Default") and c) using a carefully tuned selection of learning rates using a validation set ("Optimal"), as also reported in Section 3.3. First, it is demonstrated, by comparing the results of not training the employed layer ("No train") to the rest of the choices, that learning the parameters of the proposed method is crucial for achieving good results, confirming the importance of training the input normalization layers to adapt to the task at hand. Fine-tuning the learning rate for each model can lead to performance improvements, however, using the default settings also leads to acceptable results. On the other hand, the plain DAIN formulation seems to collapse when the selection of learning rates is sub-optimal, as shown in the case of CNN in Table 1. These results demonstrate the practical usefulness of the proposed RDAIN formulation, which allows the proposed model to be typically used without the extensive hyper-parameter finetuning that was often required for DAIN [24].

Finally, the proposed method was evaluated using the second prediction horizon of the FI-2010 dataset in Table 3. Again similar conclusions can be drawn. The MLP model seems to work the best for this dataset, since higher order temporal information is already encoded in the input features. The proposed method again leads to overall the best performance regardless the employed network architecture, demonstrating its ability to be successfully combined with any network architecture.

3.3 Ablation and Sensitivity Analysis using the FI-2010 dataset

First, the four individual components of the proposed input normalization layer was evaluated using the first five days of the FI-2010 dataset and the three different neural network architectures. The evaluation results are reported in Table 4. Note that the scaling layer leads to the most significant improvements, both for the F1 and κ score, for all the evaluated models. Then, employing adaptive gating, as well as the auxiliary static normalization stream, allows for further improving performance of the model. It is worth noting that for the CNN model the same behavior as in the Table 1 is reported, i.e., using the adaptive scaling and gating layer leads to worse performance, due to the sub-optimal selection of the learning rates for these two layers. However, when the proposed auxiliary normalization stream is employed, the performance of the model is stabilized, allowing for successfully training the model by gradually

Table 3 FI-2010 evaluation: Evaluating the performance of different normalization methods for the second prediction horizon (next 50 timesteps)

Model	Method	F1-score	κ score
MLP	Standarization	56.73 ± 0.74	0.3531 ± 0.0084
MLP	Sample Average	60.10 ± 0.41	0.3913 ± 0.0074
MLP	Sample Standarization	63.20 ± 1.52	0.4316 ± 0.0203
MLP	Batch Normalization	56.37 ± 0.76	0.3450 ± 0.0076
MLP	Instance Normalization	60.92 ± 1.72	0.3992 ± 0.0214
MLP	DAIN	65.22 ± 0.49	0.4629 ± 0.0057
MLP	Proposed	$\mathbf{66.36 \pm 0.63}$	$\mathbf{0.4802 \pm 0.0080}$
CNN	Standarization	56.73 ± 0.57	0.3389 ± 0.0106
CNN	Sample Average	59.81 ± 1.13	0.3828 ± 0.0132
CNN	Sample Standarization	60.95 ± 0.44	0.3951 ± 0.0061
CNN	Batch Normalization	56.55 ± 0.49	0.3392 ± 0.0081
CNN	Instance Normalization	60.68 ± 0.75	0.3900 ± 0.0099
CNN	DAIN	60.08 ± 0.72	0.3955 ± 0.0103
CNN	Proposed	$\mathbf{62.89 \pm 0.48}$	$\mathbf{0.4267 \pm 0.0062}$
RNN	Standarization	54.94 ± 0.26	0.3113 ± 0.0068
RNN	Sample Average	55.84 ± 0.65	0.3176 ± 0.0096
RNN	Sample Standarization	59.80 ± 0.20	0.3785 ± 0.0026
RNN	Batch Normalization	55.24 ± 0.34	0.3163 ± 0.0063
RNN	Instance Normalization	58.73 ± 0.31	0.3611 ± 0.0053
RNN	DAIN	64.40 ± 0.41	0.4496 ± 0.0064
RNN	Proposed	$\mathbf{64.60 \pm 0.42}$	$\mathbf{0.4524 \pm 0.0057}$

Table 4 Ablation study of the proposed method using the FI-2010 dataset ("Aux." refer to using the proposed auxiliary stream)

Model	Shifting	Scaling	Gating	Aux.	F1-score	κ score
MLP	✓	✗	✗	✗	54.61 ± 2.35	0.3197 ± 0.0317
MLP	✓	✓	✗	✗	68.01 ± 1.64	0.5092 ± 0.0236
MLP	✓	✓	✓	✗	68.13 ± 1.71	0.5109 ± 0.0244
MLP	✓	✓	✓	✓	$\mathbf{68.65 \pm 1.67}$	$\mathbf{0.5194 \pm 0.0245}$
CNN	✓	✗	✗	✗	53.15 ± 2.65	0.3019 ± 0.0335
CNN	✓	✓	✗	✗	46.98 ± 5.53	0.2302 ± 0.0623
CNN	✓	✓	✓	✗	44.07 ± 5.14	0.2032 ± 0.0523
CNN	✓	✓	✓	✓	$\mathbf{65.31 \pm 1.90}$	$\mathbf{0.4667 \pm 0.0279}$
RNN	✓	✗	✗	✗	53.96 ± 3.74	0.2998 ± 0.0550
RNN	✓	✓	✗	✗	65.70 ± 1.93	0.4746 ± 0.0284
RNN	✓	✓	✓	✗	66.70 ± 1.35	0.4888 ± 0.0193
RNN	✓	✓	✓	✓	$\mathbf{66.72 \pm 1.92}$	$\mathbf{0.4896 \pm 0.0279}$

introducing the gradients into the trainable branch of the proposed normalization structure. The ability of employed auxiliary normalization stream to improve the convergence of the model is also illustrated in the learning curves depicted in Fig. 4.

Next, the ability of various normalization methods to withstand various distribution shifts was evaluated in Table 5, using the last day of the testing

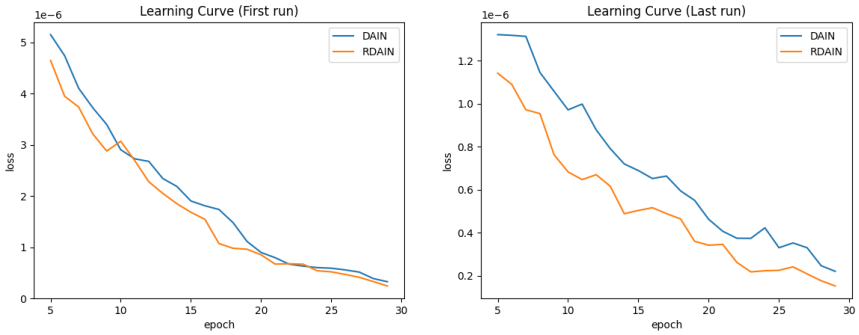

Fig. 4 Learning curves of the DAIN and RDAIN (proposed) methods for the first and last validation splits. The proposed method leads to faster convergence in most cases.

split. For these experiments, the models were first trained using the original dataset and then evaluated using corrupted data that were shifted according to a set of predefined transformation functions. The best performing MLP model was used for the evaluation, while the following four shifting functions were used:

1. Shift 1: $f(x) = x - (c_1 + \epsilon)$
2. Shift 2: $f(x) = x + (c_1 + \epsilon)$
3. Shift 3: $f(x) = x \cdot (c_2 + c_4 \cdot \epsilon)$
4. Shift 4: $f(x) = x * \cdot (c_3 + c_4 \cdot \epsilon)$,

where $c_1 = 9$, $c_2 = 1.5$, $c_3 = 0.8$, $c_4 = 10^{-3}$ and ϵ is a random number in the range 0...1 (drawn form a corresponding uniform distribution). The first two distribution shifts correspond to translations by an almost constant factor, while the third and fourth shift scale up and down all the input features. As expected, standardization is more robust to scaling transformations (shift 3 and 4), while sample-based normalization is more robust to additive transformations (Shift 1 and 2). Instance normalization and DAIN show relatively good performance across all distribution shifts, while the proposed method outperforms all the other evaluated normalization approaches by maintaining virtually the same performance for three out the four evaluated distribution shifts.

Finally, the effect of the learning rates of three layers employed by the proposed layer on the final forecasting accuracy of the models was evaluated. The results for three individual models are reported in Figures 5, 6 and 7. The validation split was used for all these experiments. Note that the optimal learning rates were used for all the conducted experiments, unless otherwise stated. It is also worth noting that it was demonstrated (Table 2) that even though the selected optimal learning rates indeed lead to improved performance, just us-

Table 5 Evaluation of the ability of various normalization methods to withstand various distribution shifts. "'Original" refers to using the original data without using any distribution shift. The Cohen's κ metric is reported.

Method	Original	Shift 1	Shift 2	Shift 3	Shift 4
Standarization	0.3638	0.0041	-0.0001	0.3767	0.3410
Sample Average	0.3223	0.3219	0.3219	0.1590	0.2488
Sample Standarization	0.4250	0.2278	0.2292	0.2565	0.2471
Batch Normalization	0.3517	0.0000	0.0000	0.3309	0.3432
Instance Normalization	0.3886	0.2860	0.2822	0.3136	0.2965
DAIN	0.5320	0.5325	0.5308	0.5115	0.4691
Proposed	**0.5435**	**0.5378**	**0.5367**	**0.5216**	**0.4705**

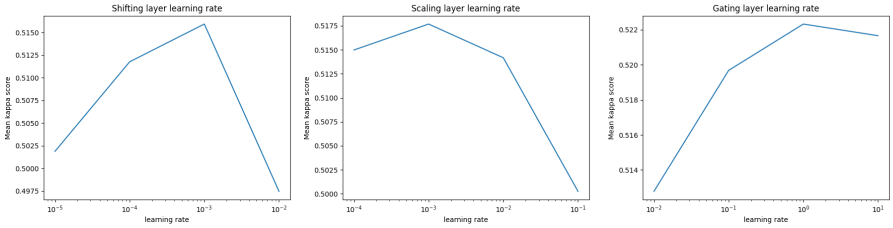

Fig. 5 Learning rate sensitivity analysis for the MLP model. The mean Cohen's κ score across the five validation splits is reported.

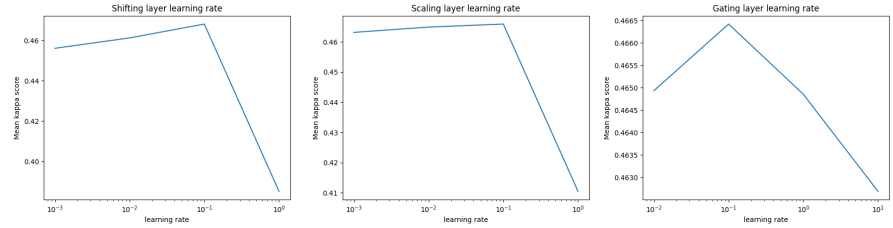

Fig. 6 Learning rate sensitivity analysis for the CNN model. The mean Cohen's κ score across the five validation splits is reported.

ing a typical choice of $\eta_\alpha = \eta_\beta = 10^{-3}$ and $\eta_\gamma = 10^{-1}$ leads to adequate results for most of the cases. Therefore, it is expected that in most applications these learning rates could be used without any additional fine-tuning. This is also verified by the sensitivity evaluation reported in the aforementioned figures, where it is demonstrated that in most cases the Cohen's κ metric only slightly decreases when these parameters are used.

3.4 BTC and ETH Evaluation

The evaluation results using the BTC dataset are provided in this Subsection. In contrast with the previous experiments, which treated the price forecasting problem as a classification problem into three classes, for the experiments

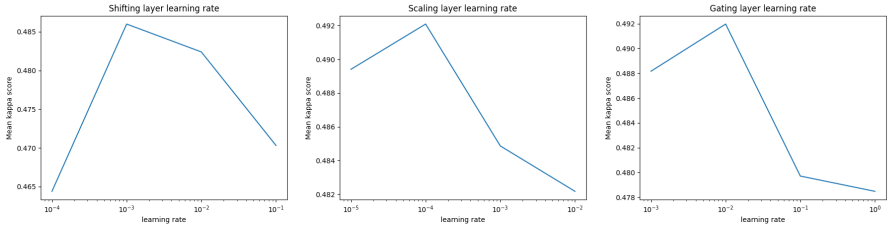

Fig. 7 Learning rate sensitivity analysis for the RNN model. The mean Cohen's κ score across the five validation splits is reported.

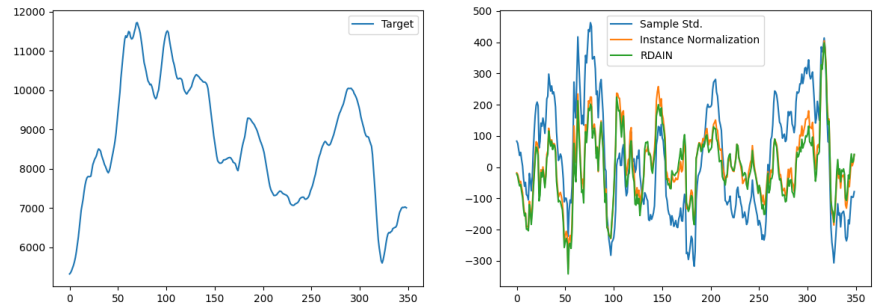

Fig. 8 BTC Evaluation: comparing the three forecasting methods on the test set of the BTC dataset. In the left figure the actual time series is of the averaged close prices is depicted, while the prediction error for the three evaluated models is depicted in the right figure.

conducted in this section the forecasting problem is tackled as a regression problem. Therefore, the models are directly trained to predict the percentage change for the close price. The evaluation results are reported in Table 6. The mean and standard deviation of 10 individual models, which were separately trained and evaluated are reported. Note that for these experiments, where the input does not encode any higher order temporal information, the RNN and CNN models lead to improved performance over the plain MLP model for most normalization methods. Again, the proposed method leads to the best results for all the evaluated metrics (MAE, R2 and maximum error) in most of the cases. Note that for the mean average error ("MAE") and maximum error ("Max Error") metrics lower values indicate better precision. The improved performance of the proposed method can be also qualitatively validated in the forecasting plots provided in Fig. 8. Note that the proposed method leads to the lowest overall deviation over the actual predicted values. The default learning rates led to the optimal performance for these experiments, with an exception for the RNN model, for which setting $\eta_\alpha = \eta_\beta = \eta_\gamma = 1$, led to slightly better forecasting accuracy.

Then, the individual models were combined into one ensemble model by averaging the output of the 10 trained models and the same evaluation was

Table 6 BTC Evaluation: Forecasting results for three different models

Model	Method	MAE	R2	Max Error
MLP	Standardization	921.98 ± 225.24	0.2505 ± 0.3132	5020.73 ± 667.96
MLP	Sample Average	89.21 ± 2.19	0.9942 ± 0.0004	428.10 ± 16.09
MLP	Sample Std.	192.71 ± 207.25	0.9572 ± 0.0743	607.45 ± 393.79
MLP	Batch Norm.	157.94 ± 47.30	0.9839 ± 0.0084	518.78 ± 52.74
MLP	Instance Norm.	89.93 ± 1.94	0.9942 ± 0.0002	405.63 ± 19.52
MLP	DAIN	91.11 ± 4.21	0.9939 ± 0.0007	412.71 ± 34.32
MLP	Proposed	$\mathbf{86.99 \pm 2.15}$	$\mathbf{0.9946 \pm 0.0002}$	$\mathbf{397.97 \pm 9.01}$
RNN	Standardization	660.27 ± 266.34	0.6956 ± 0.2313	1940.19 ± 600.36
RNN	Sample Average	107.65 ± 10.42	0.9917 ± 0.0014	460.24 ± 42.80
RNN	Sample Std.	94.50 ± 3.50	0.9936 ± 0.0006	422.92 ± 22.55
RNN	Batch Norm.	124.43 ± 11.69	0.9902 ± 0.0016	412.20 ± 37.00
RNN	Instance Norm.	99.40 ± 6.28	0.9931 ± 0.0007	415.22 ± 6.96
RNN	DAIN	92.04 ± 1.63	0.9939 ± 0.0003	420.81 ± 11.73
RNN	Proposed	$\mathbf{86.29 \pm 2.69}$	$\mathbf{0.9946 \pm 0.0003}$	$\mathbf{395.70 \pm 23.01}$
CNN	Standardization	1202.19 ± 320.40	0.0041 ± 0.4924	4310.38 ± 1323.06
CNN	Sample Average	91.43 ± 4.23	0.9938 ± 0.0004	400.21 ± 17.43
CNN	Sample Std.	87.63 ± 1.41	0.9945 ± 0.0001	403.23 ± 9.84
CNN	Batch Norm.	91.56 ± 1.90	0.9937 ± 0.0002	425.54 ± 15.70
CNN	Instance Norm.	87.84 ± 1.64	0.9944 ± 0.0002	411.71 ± 8.74
CNN	DAIN	85.28 ± 0.65	0.9946 ± 0.0001	$\mathbf{397.14 \pm 17.83}$
CNN	Proposed	$\mathbf{84.11 \pm 0.89}$	$\mathbf{0.9947 \pm 0.0001}$	408.49 ± 4.25

performed. The evaluation results are reported in Table 7. Similar results as before are obtained, demonstrating the ability of the proposed method to form effective ensembles that can significantly improve the forecasting precision. It is worth noting that this can potentially allow for capturing more fine variations of the input data, since the individual models can be specialized for detecting a specific type of input patterns by appropriately changing the normalization layers.

Furthermore, the ability of the trained model to lead to profitable trades was evaluated in Table 8. The mean profit per trade is reported, as well as the hit ratio, i.e., the percentage of trades that led to profit. The trained models were evaluated by developing a trading agent to either hold a long or short position in the market whenever a price movement larger than 0.1% is predicted. The position is closed after 10 days and the profit per trade is calculated based on the corresponding close price, as reported by the dataset. The developed agents are evaluated using both the individual models, as well as the ensemble models. First, note that most of the agents are capable of leading to profitable trades. Also, an interesting observation is that the ability of an agent to correctly forecast the time series, based on the evaluation metrics reported in Tables 6 and 7 does not always guarantee that the corresponding agent will lead to profitable trades. Indeed, an agent that performs less trades, yet more accurate ones, can be often more profitable, as also reported in the corresponding literature [9,35]. In all of the evaluated cases and models, the proposed

Table 7 BTC Evaluation: Forecasting results for three different models using ensemble architectures (10 individual models were trained and combined by averaging their decisions)

Model	Method	MAE	R2	Max Error
MLP	Standardization	854.84	0.3618	5020.73
MLP	Sample Average	85.82	0.9946	419.35
MLP	Sample Standardization	155.57	0.9857	462.86
MLP	Batch Normalization	104.89	0.9922	456.37
MLP	Instance Normalization	88.70	0.9944	404.65
MLP	DAIN	86.44	0.9945	395.75
MLP	Proposed	**85.69**	**0.9948**	**397.97**
RNN	Standardization	494.41	0.8553	1324.57
RNN	Sample Average	94.95	0.9934	388.81
RNN	Sample Standardization	89.87	0.9941	422.92
RNN	Batch Normalization	122.35	0.9907	403.89
RNN	Instance Normalization	96.44	0.9935	415.22
RNN	DAIN	86.70	0.9944	420.81
RNN	Proposed	**84.45**	**0.9948**	**395.70**
CNN	Standardization	795.59	0.5827	3119.35
CNN	Sample Average	84.68	0.9946	393.70
CNN	Sample Standardizatio	87.01	0.9945	403.23
CNN	Batch Normalization	90.22	0.9939	415.86
CNN	Instance Normalization	86.84	0.9945	411.71
CNN	DAIN	84.18	0.9948	**394.50**
CNN	Proposed	**83.27**	**0.9948**	408.49

Table 8 BTC Trading Evaluation: The performance of a trading agent is evaluated using both the individual models, as well as the ensemble models

		Individual Models		Ensemble Model	
Model	**Method**	**Profit**	**Hit ratio**	**Profit**	**Hit ratio**
MLP	Sample. Std.	104.3 ± 28.2	51.97 ± 2.0	55.03	53.11
MLP	Instance Norm.	82.3 ± 78.2	51.07 ± 3.10	115.05	51.30
MLP	DAIN	105.5 ± 127.5	53.57 ± 04.73	144.30	58.96
MLP	Proposed	$\mathbf{200.4 \pm 91.7}$	$\mathbf{57.39 \pm 2.92}$	**285.24**	**61.27**
RNN	Sample. Std	21.3 ± 89.9	51.53 ± 3.42	-24.28	50.43
RNN	Instance Norm.	-49.0 ± 47.4	49.37 ± 2.04	-77.30	48.01
RNN	DAIN	77.3 ± 80.4	54.20 ± 2.66	55.65	52.89
RNN	Proposed	$\mathbf{144.53 \pm 49.80}$	$\mathbf{56.46 \pm 1.74}$	**259.01**	**61.76**
CNN	Sample. Std	139.6 ± 20.5	53.56 ± 1.78	133.60	52.75
CNN	Instance Norm.	125.4 ± 25.7	52.53 ± 1.62	118.46	51.59
CNN	DAIN	147.0 ± 122.4	53.68 ± 5.84	137.66	53.30
CNN	Proposed	$\mathbf{187.0 \pm 35.4}$	$\mathbf{55.44 \pm 2.40}$	**231.40**	**55.46**

method leads to the most profitable trades and to the highest hit ratio, which is also further improved when combined with ensemble models, confirming the ability of the proposed approach to be effectively used in ensemble models.

Finally, the proposed method was evaluated using the ETH dataset, as shown in Table 9. Again, the proposed method leads to the overall best results

Table 9 ETH Evaluation: Forecasting results for three different models

Model	Method	MAE	R2	Max Error
MLP	Sample Std.	8.035 ± 2.44	0.9546 ± 0.0248	26.43 ± 5.77
MLP	Instance Norm.	6.132 ± 0.92	0.9696 ± 0.0081	25.71 ± 4.49
MLP	DAIN	6.210 ± 1.29	0.9689 ± 0.0143	25.21 ± 3.87
MLP	Proposed	$\mathbf{5.663 \pm 0.97}$	$\mathbf{0.9742 \pm 0.0089}$	$\mathbf{24.44 \pm 2.20}$
RNN	Sample Std.	6.259 ± 1.15	0.9680 ± 0.0128	25.80 ± 3.01
RNN	Instance Norm.	5.937 ± 0.71	0.9712 ± 0.0075	25.64 ± 1.92
RNN	DAIN	5.530 ± 0.32	0.9752 ± 0.0032	24.94 ± 1.49
RNN	Proposed	$\mathbf{5.520 \pm 0.34}$	$\mathbf{0.9757 \pm 0.0046}$	$\mathbf{24.61 \pm 2.11}$
CNN	Sample Std.	6.360 ± 1.56	0.9670 ± 0.0164	25.30 ± 3.76
CNN	Instance Norm.	5.989 ± 0.83	0.9715 ± 0.0096	25.17 ± 3.43
CNN	DAIN	6.567 ± 1.64	0.9684 ± 0.0137	25.06 ± 3.72
CNN	Proposed	$\mathbf{5.539 \pm 0.72}$	$\mathbf{0.9760 \pm 0.0071}$	$\mathbf{23.81 \pm 3.05}$

The optimal learning rates for these experiments are $\eta_\alpha = \eta_\beta = 1$, and $\eta_\gamma = 10$, except from the MLP model, for which η_γ was set to 1.

with respect to all of the evaluated metrics. Also, note that in this dataset,the RNN and CNN models again lead to improved performance over the MLP models, further confirming their ability to capture temporal information more effectively and successfully combined with the proposed approach.

4 Conclusions

An adaptive input normalization layer that can learn to identify the distribution from which the input data were generated and then apply the most appropriate normalization scheme was proposed in this paper. The proposed method allows for promptly adapting the input to the subsequent DL model, which can be especially important, given recent findings that hint at the existence of critical learning periods in neural networks. At the same time the proposed method operates on a sliding window over the time series, which enables the proposed formulation to overcome non-stationary issues that often arise. The main difference with existing approaches is that the proposed method does not just learn to perform static normalization, e.g., using a fixed set of parameters, but instead it adaptively calculates the optimal normalization parameters, significantly improving the robustness of the proposed approach when distribution shifts occur. Furthermore, the proposed method provides a more robust formulation, which can work well for a wide range of different hyper-parameters, enabling researchers to more easily use and extend the proposed approach. The effectiveness of the proposed formulation was verified using extensive experiments on three challenging financial time-series datasets and three different neural networks architectures, ranging from simpler networks composed of fully connected layers to convolutional and recurrent neural networks. Finally, note that combining the proposed method with other input

pre-processing methods, such as [18,19], could also possibly further improve its forecasting precision.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. Kaggle, Bitcoin Historical Data. https://www.kaggle.com/mczielinski/bitcoin-historical-data. Accessed: 1 June 2020
2. Kaggle, Ethereum Historical Data. https://www.kaggle.com/prasoonkottarathil/ethereum-historical-dataset. Accessed: 1 June 2020
3. Achille, A., Rovere, M., Soatto, S.: Critical learning periods in deep networks. In: Proceedings of the International Conference on Learning Representations (2018)
4. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)
5. Bao, W., Yue, J., Rao, Y.: A deep learning framework for financial time series using stacked autoencoders and long-short term memory. PloS ONE **12**(7), e0180944 (2017)
6. Chatzis, S.P., Siakoulis, V., Petropoulos, A., Stavroulakis, E., Vlachogiannakis, N.: Forecasting stock market crisis events using deep and statistical machine learning techniques. Expert systems with applications **112**, 353–371 (2018)
7. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 (2014)
8. Cohen, J.: A coefficient of agreement for nominal scales. Educational and psychological measurement **20**(1), 37–46 (1960)
9. Deng, Y., Bao, F., Kong, Y., Ren, Z., Dai, Q.: Deep direct reinforcement learning for financial signal representation and trading. IEEE Transactions on Neural Networks and Learning Systems **28**(3), 653–664 (2016)
10. Eren, L., Ince, T., Kiranyaz, S.: A generic intelligent bearing fault diagnosis system using compact adaptive 1d cnn classifier. Journal of Signal Processing Systems **91**(2), 179–189 (2019)
11. Gao, S., Wang, X., Miao, X., Su, C., Li, Y.: Asm1d-gan: An intelligent fault diagnosis method based on assembled 1d convolutional neural network and generative adversarial networks. Journal of Signal Processing Systems **91**(10), 1237–1247 (2019)
12. Gharehbaghi, A., Lindén, M.: A deep machine learning method for classifying cyclic time series of biological signals using time-growing neural network. IEEE Transactions on Neural Networks and Learning Systems **29**(9), 4102–4115 (2018)
13. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the International Conference on Artificial Intelligence and Statistics, pp. 249–256 (2010)
14. Huang, X., Belongie, S.J.: Arbitrary style transfer in real-time with adaptive instance normalization. In: Proceedings of the International Conference on Computer Vision, pp. 1510–1519 (2017)
15. Huijuan, Z., Ning, Y., Ruchuan, W.: Coarse-to-fine speech emotion recognition based on multi-task learning. Journal of Signal Processing Systems pp. 1–10 (2020)
16. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of the International Conference on Machine Learning, pp. 448–456 (2015)
17. Korczak, J., Hemes, M.: Deep learning for financial time series forecasting in a-trader system. In: Proceedings of the Federated Conference on Computer Science and Information Systems, pp. 905–912 (2017)
18. Lin, Y.F., Huang, T.M., Chung, W.H., Ueng, Y.L.: Forecasting fluctuations in the financial index using a recurrent neural network based on price features. IEEE Transactions on Emerging Topics in Computational Intelligence (2020)

19. Lin, Y.F., Ueng, Y.L., Chung, W.H., Huang, T.M.: Stock price range forecast via a recurrent neural network based on the zero-crossing rate approach. In: Proceedings of the IEEE Conference on Computational Intelligence for Financial Engineering & Economics, pp. 1–9 (2019)
20. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the International Conference on Machine Learning (2010)
21. Nayak, S., Misra, B., Behera, H.: Impact of data normalization on stock index forecasting. International Journal of. Computer Information Systems and Industrial Management Applications **6**, 357–369 (2014)
22. Nousi, P., Tsantekidis, A., Passalis, N., Ntakaris, A., Kanniainen, J., Tefas, A., Gabbouj, M., Iosifidis, A.: Machine learning for forecasting mid price movement using limit order book data. arXiv preprint arXiv:1809.07861 (2018)
23. Ogasawara, E., Martinez, L.C., De Oliveira, D., Zimbrão, G., Pappa, G.L., Mattoso, M.: Adaptive normalization: A novel data normalization approach for non-stationary time series. In: Proceedings of the International Joint Conference on Neural Networks, pp. 1–8 (2010)
24. Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., Iosifidis, A.: Deep adaptive input normalization for time series forecasting. IEEE Transactions on Neural Networks and Learning Systems (2019)
25. Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., Iosifidis, A.: Adaptive normalization for forecasting limit order book data using convolutional neural networks. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 1713–1717 (2020)
26. Shao, X.: Self-normalization for time series: a review of recent developments. Journal of the American Statistical Association **110**(512), 1797–1817 (2015)
27. Siami-Namini, S., Namin, A.S.: Forecasting economics and financial time series: Arima vs. lstm. arXiv preprint arXiv:1803.06386 (2018)
28. Silverman, B.W.: Density estimation for statistics and data analysis. Routledge (2018)
29. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15**(1), 1929–1958 (2014)
30. Tomasini, E., Jaekle, U.: Trading Systems. Harriman House Limited, Hampshire, UK (2011)
31. Tran, D.T., Iosifidis, A., Kanniainen, J., Gabbouj, M.: Temporal attention-augmented bilinear network for financial time-series data analysis. IEEE Transactions on Neural Networks and Learning Systems (2018)
32. Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., Iosifidis, A.: Using deep learning to detect price change indications in financial markets. In: Proceedings of the European Signal Processing Conference, pp. 2511–2515 (2017)
33. Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., Iosifidis, A.: Using deep learning to detect price change indications in financial markets. In: Proceedings of the European Signal Processing Conference, pp. 2511–2515 (2017)
34. Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., Iosifidis, A.: Using deep learning for price prediction by exploiting stationary limit order book features. arXiv preprint arXiv:1810.09965 (2018)
35. Tsantekidis, A., Passalis, N., Toufa, A.S., Saitas-Zarkias, K., Chairistanidis, S., Tefas, A.: Price trailing for financial trading using deep reinforcement learning. IEEE Transactions on Neural Networks and Learning Systems (2020)
36. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Proceedings of the Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
37. Wen, Z., Li, K., Huang, Z., Lee, C.H., Tao, J.: Improving deep neural network based speech synthesis through contextual feature parametrization and multi-task learning. Journal of Signal Processing Systems **90**(7), 1025–1037 (2018)
38. Wu, Y., He, K.: Group normalization. In: Proceedings of he European Conference on Computer Vision (2018)