Deep Adaptive Input Normalization for Time Series Forecasting

Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis

***Abstract*— Deep learning (DL) models can be used to tackle time series analysis tasks with great success. However, the performance of DL models can degenerate rapidly if the data are not appropriately normalized. This issue is even more apparent when DL is used for financial time series forecasting tasks, where the nonstationary and multimodal nature of the data pose significant challenges and severely affect the performance of DL models. In this brief, a simple, yet effective, neural layer that is capable of adaptively normalizing the input time series, while taking into account the distribution of the data, is proposed. The proposed layer is trained in an end-to-end fashion using backpropagation and leads to significant performance improvements compared to other evaluated normalization schemes. The proposed method differs from traditional normalization methods since it learns how to perform normalization for a given task instead of using a fixed normalization scheme. At the same time, it can be directly applied to any new time series without requiring retraining. The effectiveness of the proposed method is demonstrated using a large-scale limit order book data set, as well as a load forecasting data set.**

***Index Terms*— Data normalization, deep learning (DL), limit order book data, time series forecasting.**

I. Introduction

Forecasting time series is an increasingly important topic, with several applications in various domains [1]–[7]. Many of these tasks are, nowadays, tackled using powerful deep learning (DL) models [8]–[12], which often lead to state-of-the-art results outperforming the previously used methods. However, applying DL models to time series is challenging due to the nonstationary and multimodal nature of the data. This issue is even more apparent for financial time series since financial data can exhibit significantly different behaviors over time due to a number of reasons, e.g., market volatility.

To allow training for DL models with time series data, the data must be first appropriately normalized. Perhaps, the most widely used normalization scheme for time series when using DL is the z-score normalization, i.e., subtracting the mean value of the data and dividing by their standard deviation. However, z-score normalization is unable to efficiently handle nonstationary time series since the statistics used for the normalization are fixed both during the training and inference. Several recent works attempt to tackle this issue either by employing more sophisticated normalization schemes [13]–[15] or by using carefully handcrafted stationary features [16]. Even though these approaches can indeed lead to slightly better performance when used to train DL models, they exhibit significant drawbacks since they are largely based on heuristically designed normalization/feature extraction schemes, e.g., using price change percentages instead of absolute prices, and so on, while there is no actual guarantee that the designed scheme will indeed be optimal for the task at hand.

To overcome these limitations, we propose a deep adaptive input normalization (DAIN) layer that is capable of: 1) learning how the data should be normalized and 2) *adaptively* changing the applied normalization scheme during inference, according to the distribution of the measurements of the current time series, allowing effectively handling of nonstationary and multimodal data. The proposed scheme is straightforward to implement, can be directly trained along with the rest of the parameters of a deep model in an *end-to-end* fashion using backpropagation, and can lead to impressive improvements in the forecasting accuracy. In fact, as we experimentally demonstrate in Section III, the proposed method allows for directly training DL models without applying any form of normalization to the data since the raw time series is directly fed to the used DL model.

The main contribution in this brief is the proposal of a DL layer that learns how the data should be normalized according to their distribution instead of using fixed normalization schemes. To this end, the proposed layer is formulated as a series of three sublayers, as shown in Fig. 1. The first layer is responsible for *shifting* the data into the appropriate region of the feature space (centering), while the second layer is responsible for linearly *scaling* the data in order to increase or reduce their variance (standardization). The third layer is responsible for performing *gating*, i.e., nonlinearly suppressing features that are irrelevant or not useful for the task at hand. Note that the aforementioned process is adaptive, i.e., the applied normalization scheme depends on the actual time series that is fed to the network, and it is also trainable, i.e., the way the proposed layers behave is adapted to the task at hand using backpropagation. The effectiveness of the proposed approach is evaluated using a large-scale limit order book data set that consists of 4.5 million limit orders [17], as well as a load forecasting data set [18]. An open-source implementation of the proposed method, along with code to reproduce the experiments conducted in this brief, are available at https://github.com/passalis/dain.

To the best of our knowledge, this is the first time that an adaptive and trainable normalization scheme is proposed and effectively used in deep neural networks. In contrast to regular normalization approaches, e.g., z-score normalization, the proposed method: 1) *learns* how to perform normalization for the task at hand (instead of using some fixed statistics calculated beforehand) and 2) effectively exploits information regarding all the available features (instead of just using information for each feature of the time series separately). The proposed approach is also related to existing normalization approaches for deep neural networks, e.g., batch normalization [19], instance normalization [20], layer normalization [21], and group normalization [22]. However, these approaches are not actually designed for normalizing the input data and, most importantly, they are merely based on the statistics that are calculated during the training/inference, instead of *learning* to dynamically normalize the data. It is worth noting that it is not straightforward to use nonlinear neural layers for adaptively normalizing the data since these layers usually require normalized data in the first place in order to function correctly. In this

Manuscript received January 9, 2019; revised June 27, 2019; accepted September 26, 2019. This work was supported by H2020 Project BigDataFinance (http://bigdatafinance.eu) through Training for Big Data in Financial Research and Risk Management under Grant MSCA-ITN-ETN 675044. *(Corresponding author: Nikolaos Passalis.)*

N. Passalis, J. Kanniainen, and M. Gabbouj are with the Faculty of Information Technology and Communication, Tampere University, 33100 Tampere, Finland (e-mail: nikolaos.passalis@tuni.fi; juho.kanniainen@tuni.fi; moncef.gabbouj@tuni.fi).

A. Tefas is with the School of Informatics, Aristotle University of Thessaloniki, 541 24 Thessaloniki, Greece (e-mail: tefas@csd.auth.gr).

A. Iosifidis is with the Department of Engineering, Electrical and Computer Engineering, Aarhus University, 8000 Aarhus, Denmark (e-mail: alexandros.iosifidis@eng.au.dk).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2019.2944933

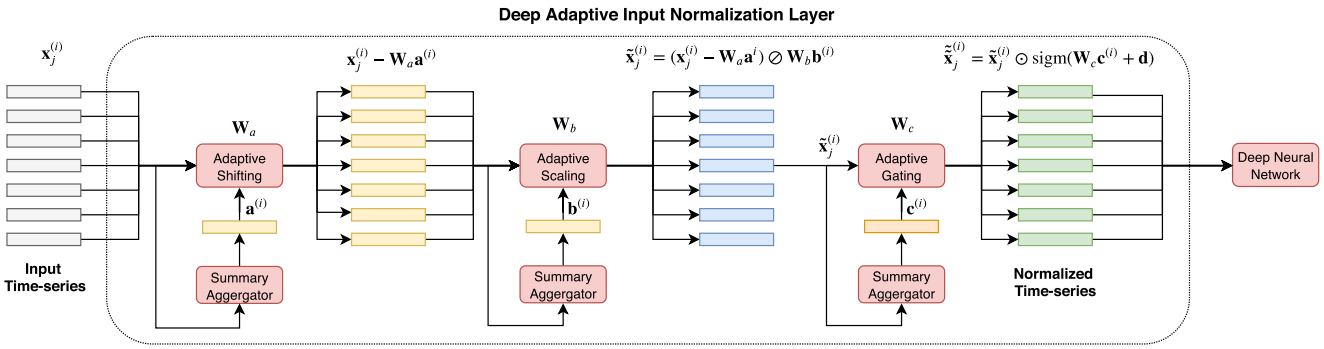

Fig. 1. Architecture of the proposed DAIN layer.

brief, this issue is addressed first by using two robust and carefully initialized linear layers to estimate how the data should be centered and scaled and then performing gating on the data using a nonlinear layer that operates on the output of the previous two layers, effectively overcoming this limitation.

The rest of this brief is structured as follows. First, the proposed method is analytically described in Section II. Then, an extensive experimental evaluation is provided in Section III, while conclusions are drawn in Section IV.

II. Deep Adaptive Input Normalization

Let $\{\mathbf{X}^{(i)} \in \mathbb{R}^{d\times L}; i = 1,\dots,N\}$ be a collection of N time series, each of them composed of L d-dimensional measurements (or features). The notation $\mathbf{x}_j^{(i)} \in \mathbb{R}^d$, $j = 1,2,\dots,L$ is used to refer to the d features observed at time point j in time series i. Perhaps the most widely used form of normalization is to perform z-score scaling on each of the features of the time series. Note that if the data were not generated by a unimodal Gaussian distribution, then using the mean and standard deviation can lead to suboptimal results, especially if the statistics around each mode significantly differ from each other. In this case, it can be argued that the data should be normalized in a *mode-aware* fashion, allowing for forming a common representation space that does not depend on the actual mode of the data. Even though this process can discard useful information, since the mode can provide valuable information for identifying each time series, at the same time, it can hinder the generalization abilities of a model, especially for forecasting tasks. This can be better understood by the following example: assume two tightly connected companies with very different stock prices, e.g., 1$ and 100$, respectively. Even though the price movements can be very similar for these two stocks, the trained forecasting models will only observe very small variations around two very distant modes (if the raw time series is fed to the model). As a result, discarding the mode information completely can potentially improve the ability of the model to handle such cases, as we will be further demonstrated in Section III, since the two stocks will have very similar representations.

The goal of the proposed method is to *learn* how the measurements $\mathbf{x}_j^{(i)}$ should be normalized by appropriately shifting and scaling them

$$\tilde{\mathbf{x}}_j^{(i)} = \left(\mathbf{x}_j^{(i)} - \boldsymbol{\alpha}^{(i)}\right) \oslash \boldsymbol{\beta}^{(i)} \tag{1}$$

where $\oslash$ is the Hadamard (entrywise) division operator. Note that global z-score normalization is a special case with $\boldsymbol{\alpha}^{(i)} = \boldsymbol{\alpha} = [\mu_1, \mu_2, \dots, \mu_d]$ and $\boldsymbol{\beta}^{(i)} = \boldsymbol{\beta} = [\sigma_1, \sigma_2, \dots, \sigma_d]$, where μ_k and σ_k refer to the global average and standard deviation of the kth input feature, respectively

$$\mu_k = \frac{1}{NL}\sum_{i=1}^{N}\sum_{j=1}^{L} x_{j,k}^{(i)}, \quad \sigma_k = \sqrt{\frac{1}{NL}\sum_{i=1}^{N}\sum_{j=1}^{L}\left(x_{j,k}^{(i)} - \mu_k\right)^2}.$$

However, as it was already discussed, the obtained estimations for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ might not be the optimal solution for normalizing every possible measurement vector since the distribution of the data might significantly drift, invalidating the previous choice for these parameters. This issue becomes even more apparent when the data are multimodal, e.g., when training model using time series data from different stocks that exhibit significantly different behaviors (price levels, trading frequency, etc.). To overcome these limitations, we propose to dynamically estimate these quantities and *separately* normalize each time series by *implicitly* estimating the distribution from which each measurement was generated. Therefore, in this brief, we propose normalizing each time series so that $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are *learned and depend on the current input*, instead of being the global averages calculated using the whole data set.

The proposed architecture is summarized in Fig. 1. First, a *summary representation* of the time series is extracted by averaging all the L measurements

$$\mathbf{a}^{(i)} = \frac{1}{L}\sum_{j=1}^{L}\mathbf{x}_j^{(i)} \in \mathbb{R}^d. \tag{2}$$

This representation provides an initial estimation for the mean of the current time series and, as a result, it can be used to estimate the distribution from which the current time series was generated, in order to appropriately modify the normalization procedure. Then, the shifting operator $\boldsymbol{\alpha}^{(i)}$ is defined using a linear transformation of the extracted summary representation as

$$\boldsymbol{\alpha}^{(i)} = \mathbf{W}_a\mathbf{a}^{(i)} \in \mathbb{R}^d \tag{3}$$

where $\mathbf{W}_a \in \mathbb{R}^{d\times d}$ is the weight matrix of the first neural layer, which is responsible for shifting the measurements across each dimension. Employing a linear transformation layer ensures that the proposed method will be able to handle data that are not appropriately normalized (or even not normalized at all), allowing for training the proposed model in an end-to-end fashion without having to deal with stability issues, such as saturating the activation functions. This layer is called the *adaptive shifting layer* since it estimates how the data must be shifted before feeding them to the network. Note that this approach allows for exploiting possible correlations between different features to perform more robust normalization.

After centering the data using the process described in (3), the data must be appropriately scaled using the scaling operator $\boldsymbol{\beta}^{(i)}$. To this

end, we calculate an updated summary representation that corresponds to the standard deviation of the data as

$$b_k^{(i)} = \sqrt{\frac{1}{L}\sum_{j=1}^{L}(x_{j,k}^{(i)} - \alpha_k^{(i)})^2}, \quad k = 1, 2, \ldots, d. \tag{4}$$

Then, the scaling function can be similarly defined as a linear transformation of this summary representation, allowing for scaling each of the shifted measurements

$$\boldsymbol{\beta}^{(i)} = \mathbf{W}_b \mathbf{b}^{(i)} \in \mathbb{R}^d \tag{5}$$

where $\mathbf{W}_b \in \mathbb{R}^{d \times d}$ is the weight matrix of the scaling layer. This layer is called the *adaptive scaling layer* since it estimates how the data must be scaled before feeding them to the network. Also, note that this process corresponds to scaling the data according to their variance, as performed with z-score normalization.

Finally, the data are fed to an *adaptive gating layer*, which is capable of suppressing features that are not relevant or useful for the task at hand as

$$\tilde{\tilde{\mathbf{x}}}_j^{(i)} = \tilde{\mathbf{x}}_j^{(i)} \odot \boldsymbol{\gamma}^{(i)} \tag{6}$$

where $\odot$ is the Hadamard (entrywise) multiplication operator and

$$\boldsymbol{\gamma}^{(i)} = \text{sigm}(\mathbf{W}_c \mathbf{c}^{(i)} + \mathbf{d}) \in \mathbb{R}^d \tag{7}$$

$\text{sigm}(x) = 1/(1 + \exp(-x))$ is the sigmoid function, $\mathbf{W}_c \in \mathbb{R}^{d \times d}$ and $\mathbf{d} \in \mathbb{R}^d$ are the parameters of the gating layer, and $\mathbf{c}^{(i)}$ is a third summary representation calculated as

$$\mathbf{c}^{(i)} = \frac{1}{L}\sum_{j=1}^{L} \tilde{\mathbf{x}}_j^{(i)} \in \mathbb{R}^d. \tag{8}$$

Note that in contrast with the previous layers, this layer is nonlinear and it is capable of suppressing the normalized features. In this way, features that are not relevant to the task at hand or can harm the generalization abilities of the network, e.g., features with excessive variance, can be appropriate filtered before being fed to the network. Overall, $\boldsymbol{\alpha}^{(i)}$, $\boldsymbol{\beta}^{(i)}$, and $\boldsymbol{\gamma}^{(i)}$ are dependent on current "local" data on window i and the "global" estimates of $\mathbf{W}_a, \mathbf{W}_b, \mathbf{W}_c$, and $\mathbf{d}$ that are trained using multiple samples on time series, $\{\mathbf{X}^{(i)} \in \mathbb{R}^{d \times L}; i = 1, \ldots, M\}$, where M is the number of samples in the training data.

The output of the proposed normalization layer, which is called deep adaptive input normalization (DAIN), can be obtained simply by feed-forwarding through its three layers, as shown in Fig. 1, while the parameters of the layers are kept fixed during the inference process. Therefore, no additional training is required during inference. All the parameters of the resulting deep model can be directly learned in an end-to-end fashion using gradient descent

$$\Delta(\mathbf{W}_a, \mathbf{W}_b, \mathbf{W}_c, \mathbf{d}, \mathbf{W}) = -\eta \left(\eta_a \frac{\partial \mathcal{L}}{\partial \mathbf{W}_a}, \eta_b \frac{\partial \mathcal{L}}{\partial \mathbf{W}_b}, \eta_c \frac{\partial \mathcal{L}}{\partial \mathbf{W}_c}, \eta_c \frac{\partial \mathcal{L}}{\partial \mathbf{d}}, \frac{\partial \mathcal{L}}{\partial \mathbf{W}} \right) \tag{9}$$

where $\mathcal{L}$ denotes the loss function used for training the network and $\mathbf{W}$ denotes the weights of the neural network that follows the proposed layer. Therefore, the proposed normalization scheme can be used on the top of every DL network, and the resulting architecture can be trained using the regular backpropagation algorithm, as also experimentally demonstrated in Section III. Note that separate learning rates are used for the parameters of each sublayer, i.e., η_a, η_b, and η_c. This was proven essential to ensure the smooth convergence of the proposed method due to the enormous differences in the resulting gradients between the parameters of the various sublayers.

III. Experimental Evaluation

For evaluating the proposed method, a challenging large-scale data set (FI-2010), which contains limit order book data, was employed [17]. The data were collected from five Finnish companies traded in the Helsinki Exchange (operated by Nasdaq Nordic) and the ten highest and ten lowest ask/bid order prices were measured. The data were gathered over a period of 10 business days from June 1, 2010 to June 14, 2010. Then, the preprocessing and feature extraction pipeline proposed in [23] were employed for processing the 4.5 million limit orders that were collected, leading to a total of 453 975 144-D feature vectors that were extracted.

We also followed the anchored evaluation setup that was proposed in [24]. According to this setup, the time series that was extracted from the first day was used to train the model and the data from the second day were used for evaluating the method. Then, the first two days were employed for training the methods, while the data from the next day were used for the evaluation. This process was repeated nine times, i.e., one time for each of the days available in the data set (except from the last one, for which no test data are available). The performance of the evaluated methods was measured using the macroprecision, macrorecall, macro-F1, and Cohen's κ. Let TP_c, FP_c, TN_c, and FN_c be the true positives, false positives, true negatives, and false negatives of class c, respectively. The precision of a class is defined as $prec_c = TP_c/(TP_c + FP_c)$, the recall as $recall_c = TP_c/(TP_c + FN_c)$, while the F1 score for a class c is calculated as the harmonic mean of the precision and the recall: $F1_c = 2 \cdot (prec_c \cdot recall_c)/(prec_c + recall_c)$. These metrics are calculated for each class separately and then averaged (macroaveraging). Finally, using Cohen's κ metric allows for evaluating the agreement between two different sets of annotations while accounting for the possible random agreements. The mean and standard deviation values over the anchored splits are reported. The trained models were used for predicting the direction of the average mid-price (up, stationary, or down) after 10 and 20 time steps, while a stock was considered stationary if the change in the mid-price was less than to 0.01% (or 0.02% for the prediction horizon of 20 time steps).

Three different neural network architectures were used for the evaluation: a multilayer perceptron (MLP) [25], a convolutional neural network (CNN) [26], [27], and a recurrent neural network (RNN) composed of gated recurrent units (GRUs) [28]. All the evaluated models receive as input the 15 most recent measurement (feature) vectors extracted from the time series and predict the future price direction. For the MLP, the measurements are flattened into a constant length vector with $15 \times 144 = 2\,160$ measurements, maintaining, in this way, the temporal information of the time series. The MLP is composed of one fully connected hidden layer with 512 neurons (the rectified linear units (ReLUs) activation function is used [29]) followed by a fully connected layer with 3 output neurons (each one corresponding to one of the predicted categories). Dropout with a rate of 0.5% is used after the hidden layer [30]. The CNN is composed of a 1-D convolution layer with 256 filters and a kernel size of 3, followed by two fully connected layers with the same architectures as in the employed MLP. The RNN is composed of a GRU layer with 256 hidden units, followed by two fully connected layers with the same architectures as in the employed MLP. The networks were trained using the cross-entropy loss.

First, an ablation study was performed to identify the effect of each normalization sublayer on the performance of the proposed method. The results are reported in Table I. The notation "DAIN (1)" is used to refer to applying only (3) for the normalization process, the notation "DAIN (1+2)" refers to using the first two layers for the

TABLE I
ABLATION STUDY USING THE FI-2010 DATA SET

Method	Model	Macro F1	Cohen's κ
No norm.	MLP	12.71 ± 13.22	0.0010 ± 0.0014
z-score norm.	MLP	53.76 ± 0.99	0.3059 ± 0.0157
Sample avg norm.	MLP	41.80 ± 3.58	0.1915 ± 0.0284
Batch Norm.	MLP	52.72 ± 1.94	0.2893 ± 0.0264
Instance Norm.	MLP	59.13 ± 2.94	0.3717 ± 0.0406
DAIN (1)	MLP	57.37 ± 3.16	0.3536 ± 0.0417
DAIN (1+2)	MLP	66.71 ± 2.02	0.4896 ± 0.0289
DAIN (1+2+3)	MLP	$\mathbf{66.92 \pm 1.70}$	$\mathbf{0.4934 \pm 0.0238}$
No norm.	CNN	12.61 ± 12.89	0.0003 ± 0.0006
z-score norm.	CNN	50.94 ± 1.12	0.2570 ± 0.0184
Sample avg norm.	CNN	53.49 ± 3.38	0.2934 ± 0.0458
Batch Norm.	CNN	45.89 ± 3.40	0.1833 ± 0.0517
Instance Norm.	CNN	57.05 ± 1.61	0.3396 ± 0.0219
DAIN (1)	CNN	59.79 ± 1.46	0.3838 ± 0.0199
DAIN (1+2)	CNN	61.91 ± 3.65	0.4136 ± 0.0574
DAIN (1+2+3)	CNN	$\mathbf{63.02 \pm 2.40}$	$\mathbf{0.4327 \pm 0.0358}$
No norm.	RNN	31.61 ± 0.40	0.0075 ± 0.0024
z-score norm.	RNN	52.29 ± 2.10	0.2789 ± 0.0295
Sample avg norm.	RNN	49.47 ± 2.73	0.2277 ± 0.0403
Batch Norm.	RNN	51.42 ± 1.05	0.2668 ± 0.0147
Instance Norm.	RNN	54.01 ± 3.41	0.2979 ± 0.0448
DAIN (1)	RNN	55.34 ± 2.88	0.3164 ± 0.0412
DAIN (1+2)	RNN	$\mathbf{64.21 \pm 1.47}$	$\mathbf{0.4501 \pm 0.0197}$
DAIN (1+2+3)	RNN	63.95 ± 1.31	0.4461 ± 0.0168

normalization process, while the notation "DAIN (1+2+3)" refers to using all the three normalization layers. The optimization ran for 20 epochs over the training data, while for the evaluation, the first 3 days (1, 2, and 3) were employed using the anchored evaluation scheme that was previously described. The proposed method is also compared to: 1) not applying any form of normalization to the data ("No norm."); 2) using z-score normalization; 3) subtracting the average measurement vector from each time series (called "Sample avg norm." in Table I); 4) using the batch normalization [19]; and 5) instance normalization layers [20] directly on the input data. Note that the batch normalization and instance normalization were not originally designed for normalizing the input data. However, they can be used for this task, providing an additional baseline. All the three models (MLP, CNN, and RNN) were used for the evaluation, while the models were trained for 20 training epochs over the data. Furthermore, the data were sampled with probability inversely proportional to their class frequency to ensure that each class is equally represented during the training. Thus, data from the less frequent classes were sampled more frequently and vice versa. For all the conducted experiments of the ablation study, the prediction horizon was set for the next ten time steps.

Several conclusions can be drawn from the results reported in Table I. First, using some form of normalization is essential for ensuring that the models will be successfully trained since using no normalization leads to κ values around 0 (random agreement). Using either z-score normalization or performing sample-based normalization seems to work equally well. Batch normalization yields performance similar to the z-score normalization, as expected, while instance normalization improves the performance over all the other baseline normalization approaches. When the first layer of the proposed DAIN method is applied (adaptive shifting), the performance of the model over the fixed normalization approaches increases (relative improvement) by more than 15% for the MLP, 30% for the CNN, and 13% for the RNN (Cohen's κ), highlighting that *learning* how to adaptively shift each measurement vector, based on the distribution from which the sample was generated, can indeed lead to significant improvements. Note that the adaptive shifting layer directly receives the raw data, without any form of normalization, and yet it manages to learn how they should be normalized in order to successfully train the rest of the network. A key ingredient for this was to: 1) avoid using any nonlinearity in the shifting process (that could possibly lead to saturating the input neurons) and 2) appropriately initializing the shifting layer, as previously described. Using the additional adaptive scaling layer, which also scales each measurement separately, further improves the performance for all the evaluated models. Finally, the adaptive gating layer improves the performance for the MLP and CNN (average relative improvement of about 2.5%). However, it does not further improve the performance of the GRU. This can be explained since GRUs already incorporate various gating mechanisms that can provide the same functionality as the employed third layer of DAIN.

Then, the models were evaluated using the full training data (except from the first day which was used to tune the hyperparameters of the proposed method) and two different prediction horizons (10 and 20 time steps). The experimental results are reported in Table II using the two best performing models (MLP and RNN). Again, no other form of normalization, e.g., z-score, and so on, was employed for the model that uses the proposed (full) DAIN layer and the instance normalization layer. Using the instance normalization leads to better performance over the plain z-score normalization. However, employing the proposed method again significantly improves the obtained results over the rest of the evaluated methods for both models.

Finally, the proposed method was also evaluated on an additional data set, the Household Power Consumption data set [18]. The forecasting task used for this data set was to predict whether the average power consumption of a household will increase or decrease the next 10 min, compared to the previous 20 min (a 90%–10% training/testing split was employed for the evaluation). The same MLP and RNN architectures as before were used for the conducted experiments, while 207-dimensional feature vectors with various measurements (one feature vector for each minute), were fed to the models. The results of the experimental evaluation are reported in Table III. Again, the proposed method leads to significant improvements over the three other evaluated methods. Also, note that even though the GRU model leads to significantly better results when simpler normalization methods are used, e.g., z-score, it achieves almost the same performance with the MLP when the proposed DAIN layer is used.

We also performed one additional experiment to evaluate the ability of the proposed approach to withstand distribution shifts and/or handle heavy-tailed data sets. More specifically, all the measurements fed to the model during the evaluation were shifted (increased) by adding 3 times their average value (except the voltage measurements). This led to a decrease in classification performance from 75.39% to 56.56% for the MLP model trained with plain z-score normalization. On the other hand, the proposed method was only slightly affected: the classification accuracy was reduced less than 0.5% (from 78.59% to 78.21%).

Hyperparameters: The learning hyperparameters were tuned for the FI-2010 data set using a simple line search procedure (the first day of the data set was used for the evaluation). The base learning rate was set to $\eta = 10^{-4}$, while the learning rates for the sublayers were set as follows: $\eta_a = 10^{-6}/10^{-2}/10^{-2}$, $\eta_b = 10^{-3}/10^{-9}/10^{-8}$, and $\eta_c = 10/10/10$ (MLP/CNN/RNN, respectively). For the household power consumption data set, the learning rates were set to $\eta_a = 10^{-5}$, $\eta_b = 10^{-2}$, and $\eta_c = 10$. The weights of the adaptive

TABLE II
EVALUATION RESULTS USING THE FI-2010 DATA SET

Normalization Method	Model	Horizon	Macro Precision	Macro Recall	Macro F1 score	Cohen's κ
z-score	MLP	10	50.50 ± 2.03	65.31 ± 4.29	54.65 ± 2.34	0.3206 ± 0.0351
Instance Normalization	MLP	10	54.89 ± 2.88	70.08 ± 2.90	59.67 ± 2.26	0.3827 ± 0.0316
DAIN	MLP	10	$\mathbf{65.67 \pm 2.26}$	$\mathbf{71.58 \pm 1.21}$	$\mathbf{68.26 \pm 1.67}$	$\mathbf{0.5145 \pm 0.0256}$
z-score	MLP	20	52.08 ± 2.33	64.41 ± 3.58	54.66 ± 2.68	0.3218 ± 0.0361
Instance Normalization	MLP	20	57.34 ± 2.67	$\mathbf{70.77 \pm 2.32}$	61.12 ± 2.33	0.3985 ± 0.0305
DAIN	MLP	20	$\mathbf{62.10 \pm 2.09}$	70.48 ± 1.93	$\mathbf{65.31 \pm 1.62}$	$\mathbf{0.4616 \pm 0.0237}$
z-score	RNN	10	53.73 ± 2.42	54.63 ± 2.88	53.85 ± 2.66	0.3018 ± 0.0412
Instance Normalization	RNN	10	58.68 ± 2.51	57.72 ± 3.90	57.85 ± 2.23	0.3546 ± 0.0346
DAIN	RNN	10	$\mathbf{61.80 \pm 3.19}$	$\mathbf{70.92 \pm 2.53}$	$\mathbf{65.13 \pm 2.37}$	$\mathbf{0.4660 \pm 0.0363}$
z-score	RNN	20	53.05 ± 2.28	55.79 ± 2.43	53.97 ± 2.31	0.2967 ± 0.0353
Instance Normalization	RNN	20	58.13 ± 2.39	60.11 ± 2.24	58.75 ± 1.53	0.3588 ± 0.0234
DAIN	RNN	20	$\mathbf{59.16 \pm 2.21}$	$\mathbf{68.51 \pm 1.54}$	$\mathbf{62.03 \pm 2.20}$	$\mathbf{0.4121 \pm 0.0331}$

TABLE III
EVALUATION RESULTS USING THE HOUSEHOLD POWER CONSUMPTION DATA SET

Normalization Method	Model	Accuracy (%)
None	MLP	71.57
z-score	MLP	75.39
Instance Normalization	MLP	77.93
DAIN	MLP	**78.83**
None	RNN	77.16
z-score	RNN	77.22
Instance Normalization	RNN	77.25
DAIN	RNN	**78.59**

shifting and adaptive scaling layers were initialized to the identity matrix, i.e., $\mathbf{W}_a = \mathbf{W}_b = \mathbf{I}_{d \times d}$, while the rest of the parameters were randomly initialized by drawing the weights from a normal distribution. The RMSProp algorithm was used for optimizing the resulting deep architecture [31].

IV. CONCLUSION

A deep adaptive normalization method, which can be trained in an end-to-end fashion, was proposed in this brief. The proposed method is easy to implement while allowing for directly using the raw time series data. The ability of the proposed method to improve the forecasting performance was evaluated using three different DL models and two time series forecasting data sets. The proposed method consistently outperformed all the other evaluated normalization approaches.

There are several interesting future research directions. First, alternative and potentially stabler learning approaches, e.g., multiplicative weight updates, can be employed for updating the parameters of the DAIN layer, reducing the need for carefully fine-tuning the learning rate for each sublayer separately. Furthermore, more advanced aggregation methods can also be used for extracting the summary representation, such as the bag-of-features representation [32]. Also, in addition to z-score normalization, min-max normalization, mean normalization, and scaling to unit length can be also expressed as special cases in the proposed normalization scheme, providing, among others, different initialization points. Finally, methods that can further enrich the extracted representation with mode information (which is currently discarded) can potentially further improve the performance of the models.

REFERENCES

[1] K. J. Kim, "Financial time series forecasting using support vector machines," *Neurocomputing*, vol. 55, nos. 1–2, pp. 307–319, Sep. 2003.

[2] A. Miranian and M. Abdollahzade, "Developing a local least-squares support vector machines-based neuro-fuzzy model for nonlinear and chaotic time series prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 2, pp. 207–218, Feb. 2013.

[3] W. Yan, "Toward automatic time-series forecasting using neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 7, pp. 1028–1039, Jul. 2012.

[4] R. Ak, O. Fink, and E. Zio, "Two machine learning approaches for short-term wind speed time-series prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 8, pp. 1734–1747, Aug. 2016.

[5] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 653–664, Mar. 2017.

[6] M. Mäkinen, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Forecasting of jump arrivals in stock prices: New attention-based network architecture using limit order book data," 2018, *arXiv:1810.10845*. [Online]. Available: https://arxiv.org/abs/1810.10845

[7] N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Temporal bag-of-features learning for predicting mid price movements using high frequency limit order book data," *IEEE Trans. Emerg. Topics Comput. Intell.*, to be published.

[8] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[9] T. Kuremoto, S. Kimura, K. Kobayashi, and M. Obayashi, "Time series forecasting using a deep belief network with restricted Boltzmann machines," *Neurocomputing*, vol. 137, pp. 47–56, Aug. 2014.

[10] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Using deep learning to detect price change indications in financial markets," in *Proc. Eur. Signal Process. Conf.*, Aug./Sep. 2017, pp. 2511–2515.

[11] A. Gharehbaghi and M. Lindén, "A deep machine learning method for classifying cyclic time series of biological signals using time-growing neural network," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 9, pp. 4102–4115, Sep. 2018.

[12] D. T. Tran, A. Iosifidis, J. Kanniainen, and M. Gabbouj, "Temporal attention-augmented bilinear network for financial time-series data analysis," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 5, pp. 1407–1418, May 2018.

[13] E. Ogasawara, L. C. Martinez, D. de Oliveira, G. Zimbrão, G. L. Pappa, and M. Mattoso, "Adaptive Normalization: A novel data normalization approach for non-stationary time series," in *Proc. Int. Joint Conf. Neural Netw.*, Jul. 2010, pp. 1–8.

[14] S. C. Nayak, B. Misra, and H. Behera, "Impact of data normalization on stock index forecasting," *Int. J. Comput. Inf. Syst. Ind. Manage. Appl.*, vol. 6, no. 1, pp. 357–369, 2014.

[15] X. Shao, "Self-normalization for time series: A review of recent developments," *J. Amer. Stat. Assoc.*, vol. 110, no. 512, pp. 1797–1817, 2015.
[16] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Using Deep Learning for price prediction by exploiting stationary limit order book features," 2018, *arXiv:1810.09965*. [Online]. Available: https://arxiv.org/abs/1810.09965
[17] A. Ntakaris, M. Magris, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Benchmark dataset for mid-price prediction of limit order book data," *J. Forecasting*, vol. 37, no. 8, pp. 852–866, 2018.
[18] G. Hébrail and A. Bérard, "Individual household electric power consumption data set," School Inf. Comput. Sci., Univ. California, Irvine, Irvine, CA, USA, Tech. Rep., 2012. [Online]. Available: https://archive.ics.uci.edu/ml/citation_policy
[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[20] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *Proc. Int. Conf. Comput. Vis.*, 2017, pp. 1510–1519.
[21] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*. [Online]. Available: https://arxiv.org/abs/1607.06450
[22] Y. Wu and K. He, "Group normalization,"2018, *arXiv:1803.08494*. [Online]. Available: https://arxiv.org/abs/1803.08494
[23] A. N. Kercheval and Y. Zhang, "Modelling high-frequency limit order book dynamics with support vector machines," *Quant. Finance*, vol. 15, no. 8, pp. 1315–1329, Jun. 2015.
[24] E. Tomasini and U. Jaekle, *Trading Systems*. Hampshire, U.K.: Harriman House Limited, 2011.
[25] P. Nousi *et al.*, "Machine learning for forecasting mid price movement using limit order book data," 2018, *arXiv:1809.07861*. [Online]. Available: https://arxiv.org/abs/1809.07861
[26] Z. Cui, W. Chen, and Y. Chen, "Multi-scale convolutional neural networks for time series classification," 2016, *arXiv:1603.06995*. [Online]. Available: https://arxiv.org/abs/1603.06995
[27] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Forecasting stock prices from the limit order book using convolutional neural networks," in *Proc. IEEE Conf. Bus. Inform. (CBI)*, Jul. 2017, pp. 7–12.
[28] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*. [Online]. Available: https://arxiv.org/abs/1412.3555
[29] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2011, pp. 315–323.
[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[31] T. Tieleman and G. Hinton, "RMSprop: Divide the gradient by a running average of its recent magnitude," *Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.
[32] N. Passalis and A. Tefas, "Training lightweight deep convolutional neural networks using bag-of-features pooling," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 36, no. 8, pp. 1705–1715, Jun. 2018.