Information Sciences 632 (2023) 815–832

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

PHILNet: A novel efficient approach for time series forecasting using deep learning

M.J. Jiménez-Navarro [a,*], M. Martínez-Ballesteros [a], F. Martínez-Álvarez [b], G. Asencio-Cortés [b]

[a] Department of Computer Science, University of Seville, Seville, ES-41012, Spain
[b] Data Science and Big Data Lab, Pablo de Olavide University, Seville, ES-41013, Spain

ARTICLE INFO

Keywords:
Time series
Forecasting
Deep learning
Efficiency

ABSTRACT

Time series is one of the most common data types in the industry nowadays. Forecasting the future of a time series behavior can be useful in planning ahead, saving time, resources, and helping avoid undesired scenarios. To make the forecasting, historical data is utilized due to the causal nature of the time series. Several deep learning algorithms have been presented in this area, where the input is processed through a series of non-linear functions to produce the output. We present a novel strategy to improve the performance of deep learning models in time series forecasting in terms of efficiency while reaching similar effectiveness. This approach separates the model into levels, starting with the easiest and continuing to the most difficult. The simpler levels deal with smoothed versions of the input, whereas the most sophisticated level deals with the raw data. This strategy seeks to mimic the human learning process, in which basic tasks are completed initially, followed by more precise and sophisticated ones. Our method achieved promising results, obtaining a 35% improvement in mean squared error and a 2.6 time decrease in training time compared with the best models found in a variety of time series.

1. Introduction

Time series is one of the most common data type nowadays. Forecasting future behavior is critical in many fields [6], such as power consumption, weather, and pollution. It is really valuable as it allows us to plan, save time, save resources and help to avoid undesired scenarios [41]. A time series is a collection of ordered data by its time component. Previous known events are used to estimate future behaviors due to the causal nature of time series data. Depending on whether the forecast considers obtaining one future step or multiple future steps, the problem is known as one-step or multi-step forecasting. Additionally, a time series that considers just one variable is named univariate while a time series with several variables is named multivariate.

In this study, we examine multi-step forecasting in univariate time series due to the relevance of various practical scenarios such as traffic management, power usage, and retail sales. In many of these cases, we must consider the evolution over numerous time steps to make judgments and implement solutions that increase performance.

* Corresponding author.
E-mail addresses: mjimenez3@us.es (M.J. Jiménez-Navarro), mariamartinez@us.es (M. Martínez-Ballesteros), fmaralv@upo.es (F. Martínez-Álvarez), guaasecor@upo.es (G. Asencio-Cortés).

https://doi.org/10.1016/j.ins.2023.03.021
Received 11 June 2022; Received in revised form 31 January 2023; Accepted 2 March 2023

Available online 8 March 2023

The research community is adopting deep learning for time series forecasting (TSF) more frequently [6,42,46], reporting competitive results when compared to traditional Box-Jenkins and other machine learning techniques. Deep learning can automatically learn features using raw input without the need for feature engineering. The model learns some non-linear relationship that maps the input to the output. However, as we cannot accurately explain the transformation learned, they are known as black-box algorithms.

In this paper, we define a hierarchical behavior for the model by giving an additional task to each layer in the model and obtaining some insights into the internal model transformations. We assume that, like people, neural nets can learn quicker by starting with easier tasks and gradually increasing the difficulty. For example, if you were to learn to ride a motorcycle, it would be easier if you already knew how to ride a bike, since you would have acquired notions such as balance, braking, circulation, etc. Learning these notions from a more complicated assignment takes longer as there are more factors to consider compared to the easiest activity. Even if the basic notions are learned via another complicated task, they may be biased and unsuitable for that task. Following the motorcycle example, if you can ride both the motorcycle and the bike, you will have a less unbiased sense of balance. This is because you cannot simply identify it with the weight or form of the motorcycle, as you have a more general experience.

We propose the Propagated HIerarchical Learning Network (PHILNet),[1] a novel approach to increase neural network efficiency for univariate multi-step TSF, especially in terms of execution time. PHILNet extends, modifies, and improves the HLNet architecture [22]. The principal modification includes the influence of harder tasks into simpler tasks in the backpropagation algorithm.

The main novelty underlying PHILNet consists in splitting the neural network into levels, which constitute one or several layers. Simpler versions of the input are fed into the first level, while more complex versions are fed into subsequent layers. Smoothing is used to simplify the input window, regulated by a parameter that is different for each layer. The initial layers transfer their information to the subsequent layers. Due to this transfer, the posterior layers can benefit from the knowledge obtained from the previous layers, improving the convergence speed. The convergence speed is especially relevant in scenarios where a large model has to be trained, several models are used to forecast different scenarios, a large number of hyperparameters have to be tuned, and so forth.

Moreover, the main contributions of this paper are as follows:

- The definition of a novel method that improves the efficiency of deep neural networks for multi-step problems with similar efficacy.
- The interpretability improvement of the model by providing insights into the layers in the neural network.
- The comparison with LSTM [20], GRU [11], DeepAR [38], Temporal Fusion Transformers [31], NBEATS [36], Prophet [40], and PHILNet using numerous datasets from different application fields.

The remainder of this paper is structured as follows. We summarize the relevant work related to this research in Section 2, focusing on significant differences. The methodology proposed in this study is described in Section 3. We show the experimentation process carried out describing the models, datasets and metrics employed in Section 4. Section 5 discusses the results obtained after experimentation, assessing the significance of the results using statistical analysis, and investigating the effect of the data size on efficiency. Finally, we discuss the results of the work and future directions in Section 6.

2. State of the art

In this section, the related work is divided into two subsections. In Section 2.1, a general state-of-the-art of time series forecasting area is described, while in Section 2.2 similar approaches to our proposed methodology are analyzed.

2.1. Area of analysis

Time series forecasting needs require large input windows in many application fields. A recent approach in time series forecasting focuses on limiting the number of connections or input records by increasing the sparsity in order to obtain good efficacy with no additional computational resources.

Bai et al. [5] proposed a novel convolutional layer on multiple datasets for TSF. This work proposes the temporal convolution network, which includes the concept of causal and dilated convolution. On the one hand, causal convolution restricts the convolutions so that only past information can influence the future. On the other hand, dilated convolutions introduce a fixed step between every two adjacent filter taps, which increases the receptive field. This model has been extended and used in several applications [17,18,28]. However, even though the receptive field increases, information is lost as a result of the effect of dilation. Our proposal attempts to improve efficiency, speeding up convergence of the neural network, not using more efficient transformations.

Oreshkin et al. [36] develop an explainable neural network architecture. The authors define the backward and forward residual connections which take ideas from the ResNet [16] architecture used in artificial vision. The backward residual connection decomposes the input sequentially, removing complexity, while the forward residual connection is a partial prediction of the neural network. However, our proposal uses a different method to improve the efficiency by introducing a new architectural paradigm in which the learned representations are formed hierarchically.

[1] All the code with the experimentation has been included in the following repository https://github.com/manjimnav/PHILNet.

Salinas et al. [38] proposed a novel architecture for probabilistic forecasting. The model uses an LSTM layer that processes the input obtaining the hidden state. Then, the likelihood model defines the noise of the forecasting by estimating the mean and the standard deviation of the next step distribution.

Several comparisons have been made between recent deep learning approaches applied to TSF. The work developed by Lara-Benítez et al. [29] shows the efficacy and efficiency of some of the most used deep learning approximations for TSF. Recurrent neural networks (RNN) showed competitive results compared to convolutional neural networks (CNN) approaches in a good variety of datasets. Recurrent neural networks are one of the most applied models for time series forecasting in several application fields [1,9,23].

Transformers have become especially relevant due to their application to natural language processing and time series forecasting [30,31,44,48]. The main component of transformers is self-attention, which assigns a score to the different time steps based on their relevance. However, this mechanism has several issues that hinder its application to TSF. Zhou et al. [49] show the main issues and develop an architecture based in the self-attention mechanism. Our methodology is applied to recurrent neural networks due to their proven effectiveness in several TSF problems, but the methodology supports using other layers instead of LSTM.

Finally, we highlight a recent non neural network approach with competitive results. In particular, the model proposed by Taylor et al. [40] called Prophet. This method defines an additive model that includes several exogenous variables such as holidays, future known features, trends, etc.

2.2. Related works

Previous research has investigated the use of several multi-resolution or smoothed time series to improve model efficiency. Typically, these approaches employ some models to learn each representation separately before combining them into a meta-model. Because they must train as many models as representations individually, these techniques are usually very inefficient.

Another comparable method modifies the training algorithm by picking easier examples to learn from before going on to the more challenging ones. The concept is similar to that of PHILNet, but we add interpretability by including the notion of increasing difficulty in the model structure.

Kourentzes and Petropoulos [26] suggest an expansion of MAPA [25] that includes the need for stock holding units. Their technique involves fitting an arbitrary model to each of numerous non-overlapping representations of the data and then averaging the model predictions using a mean or median. On the contrary, PHILNet does not fit a model for each input representation. Instead, each representation is trained together, reducing the training time.

Afolabi et al. [2] develop an approach in which an empirical mode decomposition based noise reduction procedure is used in the original time series to provide meta-information, which is then fed into a model to generate predictions. The authors successfully tested their approach in NARX networks [32] and LSTM. Noise reduction is out of the scope of PHILNet. Furthermore, the simplification is included into the model architecture and the simplified versions are not simply entered as a feature to the input. Instead, we assign the task of predicting a simplified version to the initial layers to save training time.

Wu et al. [43] develop the RESTful framework for improving forecasting efficiency by using multi-resolution patterns into time series. The authors presented a two-step process in which a GRU network is used to extract the hidden state of each time series and resolution, then a convolutional layer is used to extract the pattern of each hidden state and produce predictions. The strength of PHILNet is not only multi-resolution representations, but also smooth time series representations that extract meaningful patterns. PHILNet is also a one-step method, with feature extraction and final predictions completed in the same phase.

Bahrpeyma et al. [4] build a recurrent neural network for each of the different representations of the time series. Multiple horizons are predicted using recurrent neural networks, which are then weighted on the basis of the efficacy of the model. This technique, like other efforts, shows the issue of efficiency and model isolation.

Koenecke and Gajewar [24] use curriculum learning to analyze financial time series, achieving good results by increasing the complexity of the task during training and ordering the data previously. As a complexity ordering criterion, they employed the baby steps technique and seasonal decomposition. We focus on modifying the architecture, rather than on the training data fed to PHILNet. Additionally, we make no assumptions about the complexity of time series segments.

We are interested in optimizing the final prediction job using other tasks that support the main task. For that reason, PHILNet would be equivalent to auxiliary learning [35]. However, to our knowledge, there is no hierarchically multi-step TSF architecture for auxiliary tasks.

3. Methodology

This section introduces the proposed methodology. Hence, Section 3.1 motivates its development, Section 3.2 describes it, and finally Section 3.3 explains how the loss is calculated.

3.1. Motivation

Some studies have shown that CNN learns a hierarchical representation in artificial vision applications [45]. In these cases, the initial layers learn more generic low-level representations, while the subsequent layers use this knowledge to generate more specific higher-level representations. However, to the best of our knowledge, this pattern has not been investigated in TSF.

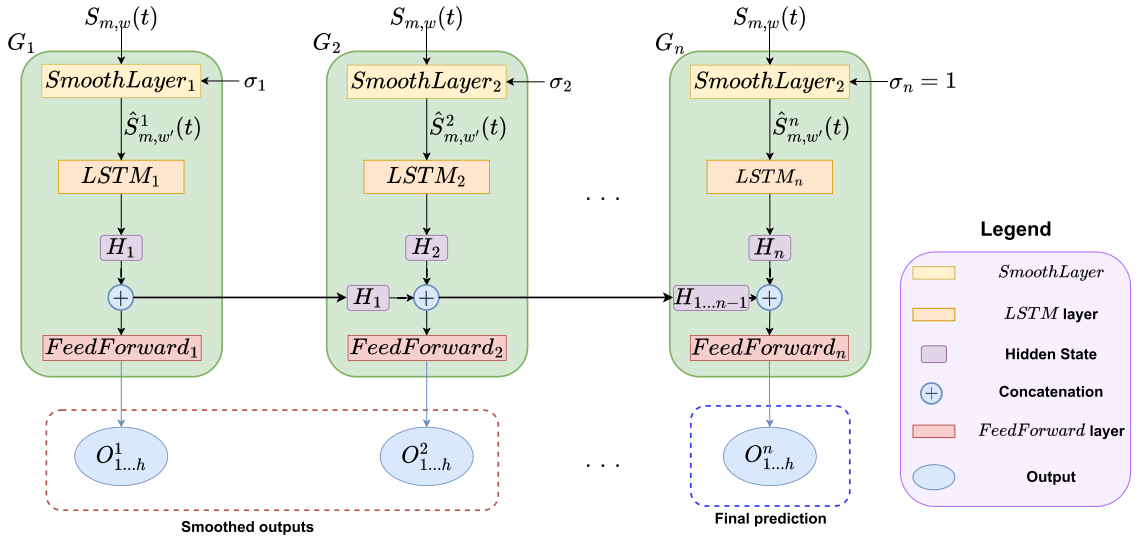


Fig. 1. Detailed version of PHILNet architecture. Each level is represented by a green box that receives the input of the model. Then, each level smooths the input using the smooth layer, encodes the smoothed input using a $LSTM$ layer, and feeds the current hidden state and the previous ones to the $FeedForward$ layer. The final prediction given by the last level represents the output of the model.

Inspired by the behavior learned by convolutional neural networks in image classification [29], we present a novel neural network architecture that explicitly uses a hierarchical representation through different objectives. In the initial levels, simpler representations allow the model to learn more generic features, which are transferred to the next layer.

We hypothesize that recurrent models, such as CNN [14], may try to learn more generic characteristics from the time series in the first layers and, progressively, more specific characteristics in the subsequent layers. According to this assumption, assigning the simpler objectives to the first levels helps the model to speed up the convergence. This is accomplished because simpler objectives are learned faster, learning the general representations that are transferred to the next levels.

3.2. Description

The hierarchical representation of the input window is obtained by using an additional task/objective to each layer or collection of layers, hereinafter called a level. Each task uses a different representation of the input window, decomposing the time series from simpler to harder tasks. The easiest task is assigned to the initial level while, as we progress through the next levels, more complicated tasks/objectives are provided.

We define the input of the network as $S_w(t)$ with length w, which is called the past window. For each level i, the model generates an input $\hat{S}^i_{w'}(t)$, which is simpler than $S_w(t)$. Smoothing allows us to recognize patterns in time series such as trends, acquire a high-frequency representation, and reduce the noise and variance. Smooth representations allow the model to obtain general representations of the input window quickly, as noise reduction facilitates the forecasting task.

The smoothing approach selected is the moving average (MA). The moving sum may be used, but the average keeps the data distribution consistent with the original. The smoothing method receives a parameter σ to control the smoothness of the level, which is a hyperparameter of the model. The parameter $\sigma_i \in \mathbb{N}$, hereinafter called the smooth factor, determines the size of the rolling window in the moving average for each level i. The smoothness and simplification degree increase as the value of σ_i or smooth factor increases. Therefore, the first levels have higher values of σ_i than the following levels. For the last level, a value of 1 is used for the smooth factor (σ) leaving the input unchanged.

Fig. 1 shows the proposed architecture divided in three main steps. First, each level (G_i) starts with a $SmoothLayer$ with their respective smooth factor (σ_i) obtaining $\hat{S}^i_{m,w'}(t)$, where w' denotes the reduced length of the input window after the smoothing process and m denotes the feature size. Then, the LSTM layer receives the simplified input window $\hat{S}^i_{m,w'}(t)$ and outputs the hidden state H_i which encodes the information of the window. The hidden state is concatenated with all the preceding hidden states. This step is crucial in the proposed approach since it is at this point where the previous levels share the information from general representation. Finally, all concatenated hidden states are fed into a $FeedForward$ layer at level i, which generates predictions for each horizon of size h. Each level i generates its own output, $O^i_{1,\dots,h}$ resulting in the same number of outputs as levels. Only the last level has the true unsmoothed forecast, which constitutes the prediction.

In the LSTM layer, the transformations applied to each element S_t in $S_{m,w}(t)$ are:

$$i_t = \sigma\left([S_t, h_{t-1}]W^i\right),$$
$$f_t = \sigma\left([S_t, h_{t-1}]W^f\right),$$
$$o_t = \sigma\left([S_t, h_{t-1}]W^o\right),$$
$$\tilde{C}_t = \tanh\left([S_t, h_{t-1}]W^g\right),$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right),$$
$$h_t = \tanh\left(C_t\right) * o_t,$$

where W^i, W^f, W^o and W^g represent the weights for the input, forget, output and cell gates respectively. $\tilde{C}_t$ represents the candidate cell state, C_t the cell state and h_t the hidden state of the layer.

Therefore, the output for each level is a function of the current level concatenated with all previous hidden states. The function applied during the $FeedForward$ layer is formulated as follows:

$$O^i_{1,\dots,h} = f_i(G_i(S^i_{m,w}(t)) + \dots + G_1(S^1_{m,w}(t))),$$

where G_i represents the transformations that output the hidden state for level i and f_i being the $FeedForward$ layer from level i. The complete process is summarized in Algorithm 3.2.1.

Algorithm 3.2.1 PHILNet inference process.

Given an input window $S_{m,w}$, and smooth factors $\sigma_i, i \in [1...n]$. For each group $G_i, i \in [1...n]$:

1. Calculate $\hat{S}^i_{m,w}(t)$ smoothing the input windows using the $SmoothLayer$ with the corresponding σ_i.
2. Compute the hidden state (H_i) using the $LSTM$ layer.
3. Concatenate the current hidden states with all previous hidden states ($H_{1...i}$).
4. Calculate the output $O^i_{1...h}$ for each horizon.

Return $O^n_{1...h}$ as final output.

3.3. Loss calculation

As a result of the forward step, each level produces a different prediction with an associated loss. The loss for each level is calculated using a smoothed version of the true future values. The loss of all predictions is finally added together to obtain the final loss. Note that the true values for each level must use the appropriate smooth factor σ_i to calculate the loss.

The first layer acquires larger losses proceeding from the simpler tasks and the subsequent tougher tasks at the backpropagation stage. As result, the first levels may tend to stay in areas of the error space where the model performs well in more tasks, obtaining general characteristics to pass on to the layers below. Furthermore, the combination of numerous losses in the initial layers produces a general gradient with greater amplitude, which helps minimize the convergence time.

We predict the gradients of the easier tasks to move to low loss areas quickly since convergence in smaller tasks is expected to be faster than in difficult tasks. Then, the subsequent levels can use that information and focus on the more specific behavior of the difficult tasks using fewer iterations.

4. Experimentation

In this section, we introduce the information about the experimentation performed in PHILNet. First, we describe the datasets used in the experiment and the relevant associated characteristics in Section 4.1. The established experimental environment is detailed in Section 4.2. The main components of the hardware used during all the experimentation are described in Section 4.3. In Section 4.4, the evaluation metrics used in all datasets are presented.

4.1. Datasets

We chose 13 public datasets from various contexts, each of which has several time series. This collection covers a wide range of TSF applications with diverse domains, window sizes, and forecasting horizons. The properties of each dataset are detailed in Table 1, including the number of time series (N), the forecast horizon (FH), and the maximum (M) and minimum (m) window sizes for each dataset used. It can be seen that more than 52000 time series have been analyzed, most of them have been used in TSF contests and other TSF studies [19].

The Demand dataset [37] was taken from the Spanish Electricity Network website and records the power demand from January to December 2015. This is a univariate dataset collected every 10 minutes from the same building. We created 72 time series by dividing the time series into non-overlapping segments of 576 time steps each. The purpose is to forecast the next 24 hours.

The CIF 2016 competition dataset [39] has 72 monthly time series, 15 of which have a 6-month forecast horizon (CIF16o6) and the rest 57 have a 12-month forecast horizon (CIF16o12). Some of these time series are artificially generated, while others are real bank risk analysis indicators.

Table 1
Summary of properties of the datasets used.

Dataset	N	FH	M	m
Demand	72	144	432	144
CIF16o12	57	12	36	15
CIF16o6	15	6	18	7
ExchangeRate (ER)	8	6	18	7
M3	1428	18	36	22
M4	48000	18	36	22
NN5	111	56	168	70
SolarEnergy (SE)	137	6	18	7
Tourism	336	24	48	30
Traffic	862	24	72	30
Traffic-metr-la (TML)	207	12	36	15
Traffic-perms-bay (TPB)	325	12	24	15
WikiWebTraffic (WWT)	997	59	118	73

The ExchangeRate (ER) dataset [27] contains records of the daily exchange rates from 1990 to 2016 in eight countries: Australia, the United Kingdom, Canada, Switzerland, China, Japan, New Zealand, and Singapore. The purpose of this dataset is to forecast the values over the next six days.

The datasets for the M3 [33] and M4 [34] competitions include time series from various domains and observation frequencies. The time series contains monthly samples from various fields. Both contests need an 18-month forecast and include time data from industry, finance, and demographics. The number of cases in each category is evenly distributed based on their occurrence in the actual world, leading to representative results in innovative investigations. The M4 dataset comprises 48000 monthly time series, whereas the M3 dataset has 1428 monthly time series.

The NN5 dataset [13] is a competition composed of 111 time series with a total length of 735 values. This dataset contains daily cash withdrawals from automated teller machines (ATMs) in England over the past two years. The competition created a forecasting horizon of 56 days.

The SolarEnergy (SE) [47] dataset provides solar power production measurements from 137 solar photovoltaic power plants in Alabama State sampled every 10 minutes. The objective is to correctly anticipate one hour.

The Tourism dataset [3] is made up of 336 monthly time series of various lengths that require a 24-month forecast. The information comes from the tourist authorities in Australia, Hong Kong, and New Zealand which represents the overall tourism figures for each nation.

Public traffic speed datasets Traffic-metr-la (TML) and Traffic-perms-bay (TPL) [21]. For the past four months, traffic-metr-la has collected data from 207 sensors on Los Angeles roadways. Similarly, Traffic-perms-bay collects 6 months of traffic speed data from 325 sensors around the Bay region. All data were gathered every 5 minutes, to forecast one hour.

The [10] traffic dataset is a collection of hourly time series obtained by the California Department of Transportation in 2015 and 2016. The information describes the road occupancy rates ranging from 0 to 1, observed by various sensors on the San Francisco Bay region highways every 10 minutes. The objective is to forecast 4 hours in advance.

The WikiWebTraffic (WWT) dataset is part of the [15] Kaggle competition, which attempts to estimate future online traffic for a set of Wikipedia pages. The daily number of hits is used to evaluate on-line traffic, and the forecasting horizon is 59 days.

4.2. Experimental settings

This section describes the experimentation process which is divided in two sub-sections. Section 4.2.1 provides an overview of the models used for comparison, while Section 4.2.2 delves into the specifics of the hyperparameter settings and data division used in the experiments.

4.2.1. Models

We selected four different types of models as baseline: recurrent based models including HLNet, attention based model, residual based model and non neural network model.

Recurrent models were selected as the proposed methodology uses this type of layer as the basic block. For that reason, LSTM [20], GRU [11], DeepAR [38] and HLNet [22] were selected for the experimental study. LSTM and GRU stack a set of layers with the same name and use a fully connected layer at the end to make a direct multi-horizon forecast. We tried to reproduce the same architecture as in the original DeepAR paper using a single LSTM layer, which makes a probabilistic forecast. We use the mean of the normal distribution as the forecasting produced by the model. Unlike the other recurrent models, the forecasting in DeepAR was originally one-step using an iterative approach to adapt to multiple horizons. Finally, HLNet was used during the experimentation with the same configuration and architecture as presented in the original paper.

Attention based model reported great results in recent applications. For that reason, Temporal Fusion Transformers (TFT) [31] was the model selected including some changes compared to the original implementation. The main changes correspond to the output of the model and the loss function. Originally, the output of the model corresponds to the tenth, fiftieth, and ninetieth quantile. As we only consider one output, we selected the fiftieth quantile as the output of the model. Additionally, the quantile loss function was

replaced to the mean square error loss as our implementation does not accept quantiles as output. Finally, exogenous variables were introduced to provide the same information to all models during the experimentation carried out.

Residual based model was also selected due to the results recently reported. In particular, NBEATS [36] is the selected model using the general block proposed in their work. This block does not assume a seasonal-trend decomposition of the time series which provides a better opportunity to adapt to a more variety of datasets.

Finally, we used a non neural network approach in order to compare the neural networks with an efficient approach which has provided competitive results recently. Prophet [40] was selected without including any exogenous variable. As Prophet can fit only one time series, we trained a model for each time series for every dataset and averaged the results obtained.

4.2.2. Experimentation

We created a comparable experimental environment for all models to perform fair comparisons. As a hyperparameter selection approach, a grid search was selected to perform the Bayesian tests in the next section. In PHILNet, the normalization technique, the hidden size, the number of layers, the window size, and the smooth factor comprise the hyperparameter space. The learning rate can be 0.01 or 0.001, the hidden size 32, 64, or 128 neurons, the number of layers 2 or 4, and the window size 125, 200, or 300 percent of the forecast horizon size.

Prophet uses its own hyperparameters, which are not shared with the neural networks. The official documentation indicates that the principal hyperparameters are the changepoint prior scale (CPE) and seasonal prior scale (SPE). The SPE and CPE hyperparameters can take the values: 0.001, 0.01, 0.1, 1, and 10.

The train/test division has been chosen using a fixed origin method [19]. For each segment or separated univariate time series ($m = 1$), the last records are used as the forecasting horizon, while the remaining records in the segment are used as training. A windowing process is used for each segment to obtain the input windows and their respective forecasting horizons.

4.3. Hardware

All experiments have been executed on the same machine with the same hardware configuration. The hardware configuration consists of an NVIDIA Geforce RTX 3070 GPU, an AMD Ryzen 7 5800X 3.8 GHz, and 32 GB RAM.

4.4. Evaluation metrics

We analyzed the outcomes of PHILNet, HLNet and LSTM by comparing three forecasting indicators and one efficiency parameter. The well-known Mean Absolute Error (MAE) and Mean Squared Error (MSE) are the first two metrics, using the following formulas:

$$MAE = \frac{1}{h}\sum_{t=1}^{h}|y_t - \hat{y}_t|$$

$$MSE = \frac{1}{h}\sum_{t=1}^{h}(y_t - \hat{y}_t)^2$$

Furthermore, the Weighted Average Percentage Error (WAPE) was chosen for the variety of datasets with values ranging from 0 to 1, and metrics like Mean Absolute Percentage Error (MAPE) do not accurately reflect reality.

$$WAPE = \frac{\sum_{t=1}^{h}|y_t - \hat{y}_t| * 100}{\sum_{t=1}^{h} y_t}$$

Finally, as an efficiency metric, we measured the training time to show the efficiency of the model.

5. Results discussion

This section discusses the results obtained after the experimentation and evaluation carried out. Section 5.1 compares PHILNet with the previous presented approach focusing on the improvement obtained. Section 5.2 makes a comparison between all the baseline models explained in Section 4.2. Section 5.3 shows the best models found in the baseline and makes a more detailed comparison. Finally, we discuss the influence of the size of the data on the training time in Section 5.4.

5.1. Comparison with HLNet

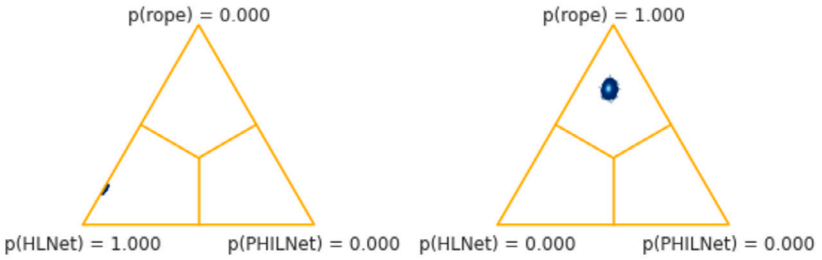
In this section, a comparison with the original proposal is reported. The results analyze the improvement of PHILNet over HLNet in terms of training time. Then, to make conclusions about the statistical significance of the obtained results, a test has been used to support the results.

5.1.1. Results

In PHILNet, the main features can be summarized as follows: use several tasks hierarchically in which the simpler tasks transfer their knowledge to more difficult tasks, and simultaneously, harder tasks influence the simpler tasks via backpropagation. If we do

Table 2
MAE, MSE, WAPE and training time for each dataset and model of our modified version of PHILNet.

	Model							
	PHILNet				HLNet			
	MAE	MSE	WAPE (%)	TIME	MAE	MSE	WAPE (%)	TIME
Demand	899	1.84e06	3.31	**3' 08"**	**893**	**1.77e06**	**3.27**	3' 40"
CIF16o12	**1.22e04**	**8.66e09**	16.48	**0' 05"**	1.40e04	9.06e09	**15.32**	0' 17"
CIF16o6	**1.89e06**	**1.90e13**	**19.54**	0' 05"	2.14e06	2.37e13	19.93	**0' 04"**
ER	**1.80e-4**	**6.25e-6**	**0.28**	**1' 05"**	**1.80e-4**	8.00e-6	**0.28**	2' 02"
M3	**674**	1.52e06	15.43	**1' 34"**	683	**1.50e06**	**15.28**	2' 49"
M4	604	2.00e06	**14.71**	**6' 07"**	**598**	**1.94e06**	14.78	12' 18"
NN5	**3.48**	30.20	18.40	**5' 27"**	3.49	**29.49**	**18.34**	23' 23"
SE	**1.77**	**8.75**	**183.00**	16' 14"	2.16	12.4	255.00	**7' 21"**
Tourism	2224	**8.28e07**	19.49	**1' 28"**	**2217**	9.61e07	**18.95**	1' 40"
Traffic	**1.2e-3**	6.77e-4	34.44	**6' 27"**	**1.2e-3**	**5.77e-4**	**34.40**	21' 12"
TML	**1.99**	**8.60**	**3.33**	**11' 26"**	2.00	8.82	3.34	12' 16"
TPB	**0.92**	2.13	**1.38**	**14' 17"**	**0.92**	**2.12**	**1.38**	33' 41"
WWT	12.00	9171	**47.17**	**9' 28"**	**11.99**	**9009**	47.38	45' 24"



(a) Training time (b) WAPE

Fig. 2. Probability of having greater training time and WAPE between PHILNet and HLNet.

not use several tasks hierarchically or transfer the knowledge, we have an architecture similar to an LSTM. However, if harder tasks cannot influence in the simpler tasks of the architecture, the proposal would be the same as HLNet.

Table 2 shows the comparison between PHILNet and HLNet. In general, if we define the best method as the method with the best results in more metrics, the efficacy obtained in HLNet is better in 8 datasets, while PHILNet is better in 5 datasets. However, the difference is not very significant when HLNet is better.

In terms of training time, PHILNet obtains better results in 11 datasets while HLNet is better than PHILNet in 2 datasets. In addition, the training time for the SE dataset has been reduced considerably. However, there is a notable difference in the rest of the datasets in general. In NN5, Traffic, TPB, and WWT, PHILNet obtains half the training time than HLNet.

5.1.2. Bayesian test

To statically prove the results obtained, we decided to use a Bayesian approximation, since the null hypothesis significance test has several well-known problems [7].

We used the Bayesian correlated t-test [12] and Bayesian signed-rank test [8] to make the comparison for each dataset and all datasets, respectively. These tests are the Bayesian version for Wilcoxon signed rank test and t-test. The input is the paired results of two models A and B, which returns the probability that model A is better than B, model B is better than A, and there are no significant differences (rope).

WAPE and training time were analyzed using a Bayesian approach to show that the results obtained are statistically significant. Bayesian tests need to define a region of practical equivalence, also called rope. In the case of WAPE we decided to set the rope at 1%. For training time, the rope will be 1% of the average difference between PHILNet and HLNet. This means that experiments in which the WAPE difference is less than 1% and experiments in which the training time difference is less than 1% the average difference in the training time are considered equivalent. As the Bayesian tests are paired, it is necessary to have the same number of samples for each model. The samples are the results obtained for each model trained with different hyperparameters in all datasets. We obtained all pairs of results with the same hyperparameters for HLNet and PHILNet.

Fig. 2 illustrates the probabilities for the different scenarios. The probability of PHILNet having higher WAPE than HLNet is represented in p(PHILNet) and the inverse scenario is represented in p(HLNet). Furthermore, p(rope) indicates the probability of similar results. As you can see, the probability of HLNet having greater time than PHILNet and the probability of PHILNet and HLNet having similar results are 1.

Table 3
Best WAPE (in percentage) found for every model in each dataset.

	LSTM	GRU	DeepAR	TFT	Prophet	NBEATS	PHILNet
Demand	3.83	5.27	14.08	4.40	267.99	14.33	**3.31**
CIF16o12	17.10	**12.87**	23.99	14.64	17.02	15.86	16.48
CIF16o6	32.23	25.17	71.82	26.98	20.92	27.20	**19.54**
ER	0.30	0.32	13.02	0.39	6.08	0.36	**0.28**
M3	**15.30**	15.43	30.36	15.53	15.79	17.41	15.43
M4	**14.57**	14.75	41.73	15.07	16.90	15.73	14.71
NN5	18.56	18.89	41.90	18.51	19.37	41.29	**18.40**
SE	222.00	259.61	2255.52	233.96	2357.02	300.56	**183.00**
Tourism	19.69	**19.16**	50.90	19.63	32.94	37.99	19.49
Traffic	**33.92**	34.67	77.61	39.37	69.94	72.84	34.44
TML	**3.33**	3.38	40.39	3.51	9.94	3.40	**3.33**
TPB	1.38	**1.36**	35.16	1.68	–	1.45	1.38
WWT	**46.89**	47.45	175.22	48.14	68.95	48.49	47.17

Table 4
Training time for the best models in each dataset.

	LSTM	GRU	DeepAR	TFT[a]	Prophet	NBEATS	PHILNet
Demand	0’ 49”	8’ 57”	8’ 03”	50’ 32”	0’ 41”	**0’ 2”**	3’ 08”
CIF16o12	0’ 30”	0’ 50”	0’ 07”	1’ 31”	0’ 20”	**0’ 01”**	0’ 05”
CIF16o6	0’ 05”	0’ 02”	0’ 01”	0’ 21”	0’ 05”	0’ 11”	**0’ 05”**
ER	2’ 05”	1’ 36”	0’ 27”	57’ 07”	0’ 27”	**0’ 11”**	0’ 52”
M3	7’ 22”	6’ 01”	1’ 21”	56’ 51”	10’ 29”	**0’ 11”**	1’ 34”
M4	9’ 15”	29’ 34”	2’ 47”	62’ 05”	353’ 42”	**1’ 38”**	6’ 07”
NN5	20’ 09”	16’ 13”	4’ 09”	109’ 29”	0’ 45”	**0’ 38”**	5’ 27”
SE	10’ 14”	10’ 40”	7’ 16”	42’ 35”	61’ 50”	**1’ 57”**	16’ 14”
Tourism	3’ 41”	4’ 58”	1’ 46”	30’ 29”	2’ 30”	**0’ 21”**	1’ 28”
Traffic	41’ 02”	33’ 47”	23’ 54”	73’ 16”	55’ 46”	**1’ 42”**	6’ 27”
TML	14’ 14”	10’ 21”	5’ 57”	81’ 49”	90’ 44”	**2’ 16”**	11’ 26”
TPB	48’ 27”	48’ 10”	2’ 28”	50’ 03”	–	**5’ 23”**	14’ 17”
WWT	46’ 49”	54’ 17”	24’ 32”	81’ 33”	6’ 57”	**3’ 31”**	9’ 28”

[a] Due to incompatibility between the libraries used during the experimentation, the execution was done using the CPU.

With these results, we can conclude that the inclusion of several simpler tasks is enough to improve training time in the datasets. However, we prove that a key concept is a hierarchical optimization, where the harder tasks can influence the simpler tasks. We hypothesize that to reduce the negative transfer impact of the simpler tasks on the harder tasks, we must optimize both tasks jointly.

5.2. Comparison with baseline architectures

An experimental comparison with all the baseline architectures described in Section 4.2 is reported in this section. The results show the best models for each dataset analyzed in terms of efficiency and efficacy.

Table 3 shows the efficacy of each model in all datasets. It can be noted that only the WAPE metric has been used to facilitate the comparison among the tested architectures.

The models based on recurrent layers (PHILNet, LSTM and GRU) except for DeepAR obtained the best results in terms of efficacy. PHILNet obtained the best results in six datasets, LSTM was the best model in five, and GRU was the best in three datasets. We hypothesize that DeepAR obtained poor results although it used recurrent layers as consequence of the iterated forecasting approach.

TFT model obtained generally good results, but they are still under the GRU, LSTM, and PHILNet efficacy.

Prophet had notably worse results compared to the other methods. In datasets like CIF16o6, CIF16o12 or M3, the method obtained remarkable results, but in datasets like WWT or Demand, the results were worse than other models, in general. Furthermore, the efficacy of the TPB dataset was several orders of magnitude higher than the error evaluated in other datasets. For this reason, the results in TPB were removed from the Table 3 and Table 4. To the best of our knowledge, this result may be due to the irregular patterns contained in the time series of some datasets which difficult the training to the set of Prophet models.

NBEATS has the same problem as Prophet. This model obtains competitive results in datasets like CIF16o6, M4 or TML. However, the results are remarkably poorer than models in other datasets, making this model less generalizable to other problems.

In terms of efficiency, Table 4 shows that the best models were NBEATS, DeepAR, and PHILNet. NBEATS and DeepAR were expected to be efficient models, as they are simpler models than the other ones. Despite the fact that PHILNet has almost the same architecture as the usual LSTM model, it presents a good efficiency comparable to simpler models in some datasets.

The poor efficacy obtained in the TFT model was consequence of using the CPU for training instead of the GPU because the libraries and drivers used were incompatible with the official implementation of the model.

Table 5
The best hyperparameters found for every model in each dataset.

	LSTM		GRU		DeepAR		TFT		Prophet		NBEATS		PHILNet		
	Layers	Units	Layers	Units	Layers	Units	Layers	Units	CPE	SPE	Layers	Units	Layers	Units	GF
Demand	2	32	2	64	1	64	2	64	0.1	1	2	128	4	32	2
CIF16o12	4	64	4	128	1	128	2	64	10	0.01	4	64	2	128	4
CIF16o6	4	32	2	128	1	128	2	64	0.1	0.001	2	64	4	128	2
ER	2	64	2	64	1	32	2	128	10	0.001	2	32	2	128	4
M3	2	128	4	32	1	64	2	64	0.5	0.01	2	32	4	32	4
M4	2	32	4	128	1	32	2	32	0.1	0.01	2	32	2	64	8
NN5	2	128	4	32	1	64	2	64	0.5	10	4	32	4	32	8
SE	2	32	2	64	1	128	2	128	10	0.001	4	64	4	32	4
Tourism	2	128	2	128	1	64	2	64	10	0.001	2	128	2	128	8
Traffic	2	128	4	32	1	128	2	128	0.5	10	4	64	2	128	4
TML	2	64	2	32	1	64	2	128	0.1	10	4	32	2	64	6
TPB	4	64	4	64	1	32	2	64	10	0.001	4	128	4	128	4
WWT	4	32	2	128	1	64	2	32	0.5	10	4	128	2	32	2

Prophet seems to have good efficacy in datasets with few time series in general. However, in datasets with a lot of time series, the efficiency decreases as Prophet needs to model each one independently. Table 4 shows that datasets with few instances do not necessarily obtain better efficiency. Additionally, the amount of memory needed to fit a dataset is greater than other models, since every time series in a dataset contains an independent model, which is responsible of the forecasting of the time series.

LSTM and GRU seem to have an efficiency similar to our proposed methodology, but PHILNet is faster even using similar layers. Section 5.3 compares LSTM, GRU, and PHILNet in more detail, as are the models that obtain better results without significantly increasing the training time.

As conclusion, PHILNet seems to be the option which obtain a better efficacy in more datasets preserving a good efficiency.

Table 5 shows the best hyperparameters found for each model in each dataset. In general, there seems to be no pattern in the models nor datasets, showing that more layers or units are required. In Prophet hyperparameters, a very low value of the seasonal prior scale (SPE) and a high value of the changepoint prior scale (CPE) seem to be the most common pattern in most of the datasets.

5.3. Comparison with LSTM and GRU

In this section, a detailed comparison with the best architectures found in Section 5.2 is reported. The following discussion of results analyzes the improvement of PHILNet over LSTM and GRU. Then we apply the Bayesian test between our proposal and the best baseline architecture to demonstrate the statistical significance of the results obtained.

5.3.1. Results

The MAE, MSE and WAPE metrics mentioned in Section 4.4 for the LSTM, GRU and PHILNet architectures are shown in Table 6. In most situations, PHILNet obtains a significant improvement in terms of training time with similar or better results than LSTM or GRU.

In 9 of 13 datasets, the training time for PHILNet was reduced, whereby 8 of 10 datasets obtained more than half the training time compared with LSTM or GRU. The improvement in training time is not related to a decrease in the number of parameters, as seen in Table 5. When comparing the units and layers of LSTM and PHILNet, there does not appear to be any pattern. This fact seems to indicate that PHILNet does not require more or fewer parameters to obtain equivalent results to LSTM or GRU in general. We found that a value of 4 for the smooth factor (sigma) is the most common value for the best models.

If we define the best model of a dataset as the model with the least error in more metrics, there are four datasets in which LSTM outperforms all models, seven datasets in which PHILNet outperforms all models, one dataset in which GRU outperforms all models and one draw. However, in most datasets, there are no significant differences, except for Demand and CIF16o6 datasets, where PHILNet improves LSTM by at least 10% in all metrics.

In general, LSTM seems to be the most competitive architecture, compared to PHILNet. For that reason, the rest of the analysis focuses mainly on the comparison between LSTM and PHILNet.

Table 7 shows the relative average improvement for each metric between the best and second best architecture using the results in Table 6. Additionally, the number of times that an architecture obtains the best results for a metric is reported. The table supports the idea that PHILNet has a better improvement, while the improvement of LSTM and GRU is less notable.

The WAPE of the LSTM, GRU and PHILNet is shown in Fig. 3 for each value of smooth factor (σ). Because the LSTM and GRU do not have the smooth factor (σ), it appears to be 0. We give a boxplot that displays the range of the findings obtained after doing numerous trials with varied settings.

In most situations, when looking at the influence of the smooth factor parameter on PHILNet, we cannot detect any clear relationship between the different smooth factor values and the metrics. When the smooth factor changes its value, the range of the boxplot appears to fluctuate dramatically, but the median in most datasets remains almost the same.

Table 6
MAE, MSE, WAPE (in percentage) and training time for each dataset and model.

	Model											
	LSTM				GRU				PHILNet			
	MAE	MSE	WAPE	TIME	MAE	MSE	WAPE	TIME	MAE	MSE	WAPE	TIME
Demand	1055	2.21e06	3.83	**0' 49"**	1440	4.01e06	5.27	8' 57"	**899**	**1.84e06**	**3.31**	3' 08"
CIF16o12	1.26e04	**7.52e09**	17.10	0' 30"	1.59e04	1.04e10	**12.87**	0' 50"	**1.22e04**	8.66e09	16.48	**0' 05"**
CIF16o6	2.34e06	2.56e13	32.23	0' 05"	3.83e06	1.14e14	25.17	**0' 02"**	**1.89e06**	**1.90e13**	**19.54**	0' 05"
ER	1.9e-4	6.72e-6	0.30	2' 05"	2.06e-4	8.85e-6	0.32	1' 36"	**1.8e-4**	**6.25e-6**	**0.28**	**0' 52"**
M3	691	1.55e06	**15.30**	7' 22"	694	1.58e06	15.43	6' 01"	**674**	**1.52e06**	15.43	**1' 34"**
M4	**592**	**1.94e06**	**14.57**	9' 15"	603	1.99e06	14.75	29' 34"	604	2.00e06	14.71	**6' 07"**
NN5	3.50	30.50	18.56	20' 09"	3.57	30.28	18.89	16' 13"	**3.48**	**30.20**	**18.40**	**5' 27"**
SE	**1.73**	**7.60**	222.00	**10' 14"**	2.03	10.07	259.61	10' 40"	1.77	8.75	**183.00**	16' 14"
Tourism	2349	1.21e08	19.69	3' 41"	2388	1.46e08	**19.16**	4' 58"	**2224**	**8.28e07**	19.49	**1' 28"**
Traffic	**1.23e-3**	**6.53e-4**	**33.92**	41' 02"	1.27	7.07e-4	34.67	33' 47"	**1.23e-3**	6.77e-4	34.44	**6' 27"**
TML	**1.99**	8.77	**3.33**	14' 14"	2.01	9.15	3.38	**10' 21"**	**1.99**	**8.60**	**3.33**	11' 26"
TPB	0.91	2.10	1.38	48' 27"	**0.90**	**2.03**	**1.36**	48' 10"	0.92	2.13	1.38	**14' 17"**
WWT	**11.97**	**8833**	**46.89**	46' 49"	12.11	9430	47.45	54' 17"	12.00	9171	47.17	**9' 28"**

Table 7
Average improvement for the best results of each architecture with the second best architecture over each dataset. The number of instances where the architecture outperforms all the other architectures is in parentheses.

	LSTM	GRU	PHILNet
MAE	1.17% (5)	1.08% (1)	**11.41% (7)**
MSE	5.91% (5)	3.33% (1)	**34.67% (7)**
WAPE	2.60% (4)	10.33% (3)	**29.51% (5)**
Time	63.45% (2)	34.5% (2)	**267.71% (9)**

Table 8
Probability of LSTM having greater WAPE than PHILNet (LSTM), probability of PHILNet having greater WAPE than LSTM (PHILNet) and probability of PHILNet having similar WAPE than LSTM (rope) for each smooth factor.

Dataset	Smooth factor											
	2			4			6			8		
	LSTM	rope	PHILNet	LSTM	rope	PHILNet	LSTM	rope	PHILNet	LSTM	rope	PHILNet
Demand	0.18	0.82	0.00	0.22	0.78	0.00	0.28	0.72	0.00	0.21	0.79	0.00
CIF16o6	0.97	0.03	0.00	0.96	0.03	0.00	0.98	0.02	0.00	–	–	–
CIF16o12	0.56	0.43	0.00	0.54	0.45	0.00	0.60	0.39	0.00	0.54	0.46	0.00
M3	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00
M4	0.00	1.00	0.00	0.10	0.77	0.14	0.00	0.99	0.00	0.05	0.89	0.05
NN5	0.14	0.86	0.00	0.15	0.85	0.00	0.13	0.87	0.00	0.16	0.84	0.00
SE	0.01	0.01	0.99	0.14	0.04	0.82	0.00	0.00	1.00	–	–	–
ER	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	–	–	–
Tourism	0.79	0.15	0.06	0.76	0.17	0.07	0.77	0.16	0.07	0.76	0.17	0.07
Traffic	0.00	0.41	0.59	0.00	0.82	0.18	0.00	0.74	0.26	0.00	0.55	0.45
TML	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00
TPB	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00
WWT	0.70	0.07	0.23	0.70	0.07	0.23	0.70	0.07	0.23	0.69	0.07	0.24

We identified that LSTM and GRU results seem more unstable than PHILNet, with less extreme outliers introduced (most of which then have been removed for visual clarity). This fact leads us to believe that PHILNet is more resilient, displaying fewer results with distribution outliers regardless of the settings.

In general, the results in Fig. 3 are comparable, with no apparent winner in most of datasets. The principal difference between LSTM, GRU and PHILNet is the IQR (interquartile) range obtained in CIF16o6 or ExchangeRate (ER), although the median is nearly the same. The smooth factor seems to have some influence on the results depending on the dataset. The median and IQR range of PHILNet is generally stable for all smooth factors, except in the M4, Solar Energy (SE), and Traffic-perms-bay (TPB) datasets which presents a large variability. In the tourism and M3 data sets, PHILNet has a poorer median and IQR range, although it has superior results in Table 6. GRU seems to outperform LSTM and PHILNet in CIF16012, Demand and Tourism. Finally, even if the metrics in the results table were identical, PHILNet in Traffic-metr-la (TML) provides better results in terms of median and IQR range.

The training time distribution of the experiments is shown in Fig. 4. The boxplots seem to support the notion that PHILNet takes less time to train than LSTM and GRU in general, with less IQR range and median. The median and IQR range of training time was lower in 11 datasets while the training time is similar to, or higher in the five remaining datasets. Interestingly, in SolarEnergy (SE) dataset where PHILNet exhibited a longer training time than LSTM and GRU for the best configuration, the median and IQR range is

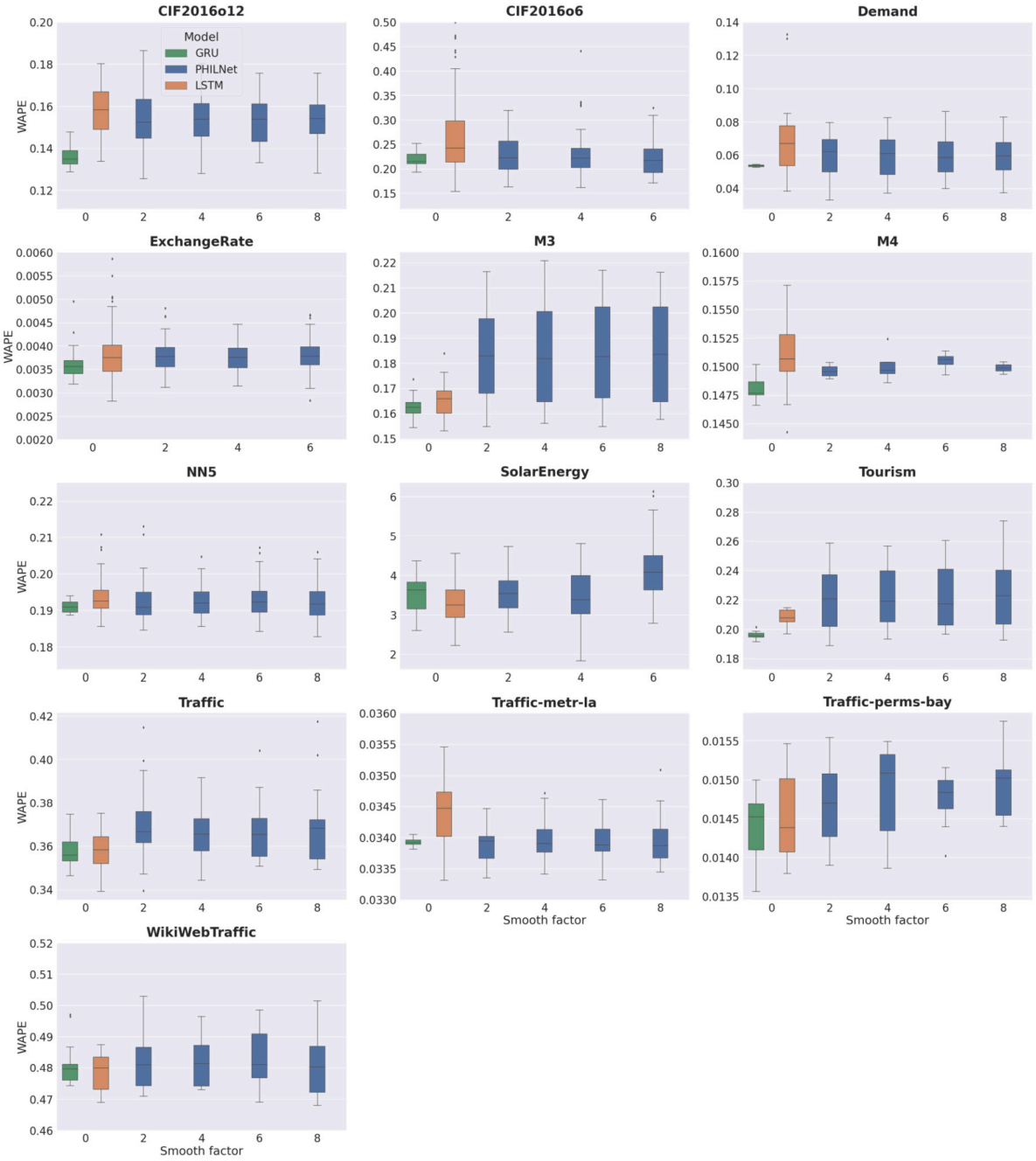


Fig. 3. Boxplot of all experiments indicating the WAPE value for each number of units within PHILNet.

lower. In this case, the training time does not represent a large variation for every smooth factor, indicating that the smooth factor has no great impact on the training time.

Our theory is that PHILNet takes longer to train in the remaining circumstances because the model does not fall into a local minimum and needs more time to find a better minimum. This implies that, when the LSTM models reach a local minimum, the metrics do not improve and early stopping is triggered, reducing the training time in half. Because the PHILNet does not fall as quickly into local minima, the early stopping is delayed, extending the training period.

5.3.2. Bayesian tests

In the previous section, the results seem to indicate that PHILNet has similar results than LSTM in general requiring less training time, while GRU seems to be worst than both models. To obtain a statistical significance in our conclusions, we applied a Bayesian test as in Section 5.1.2. Note that, in contrast to HLNet, all hyperparameters between LSTM and PHILNet are the same except for

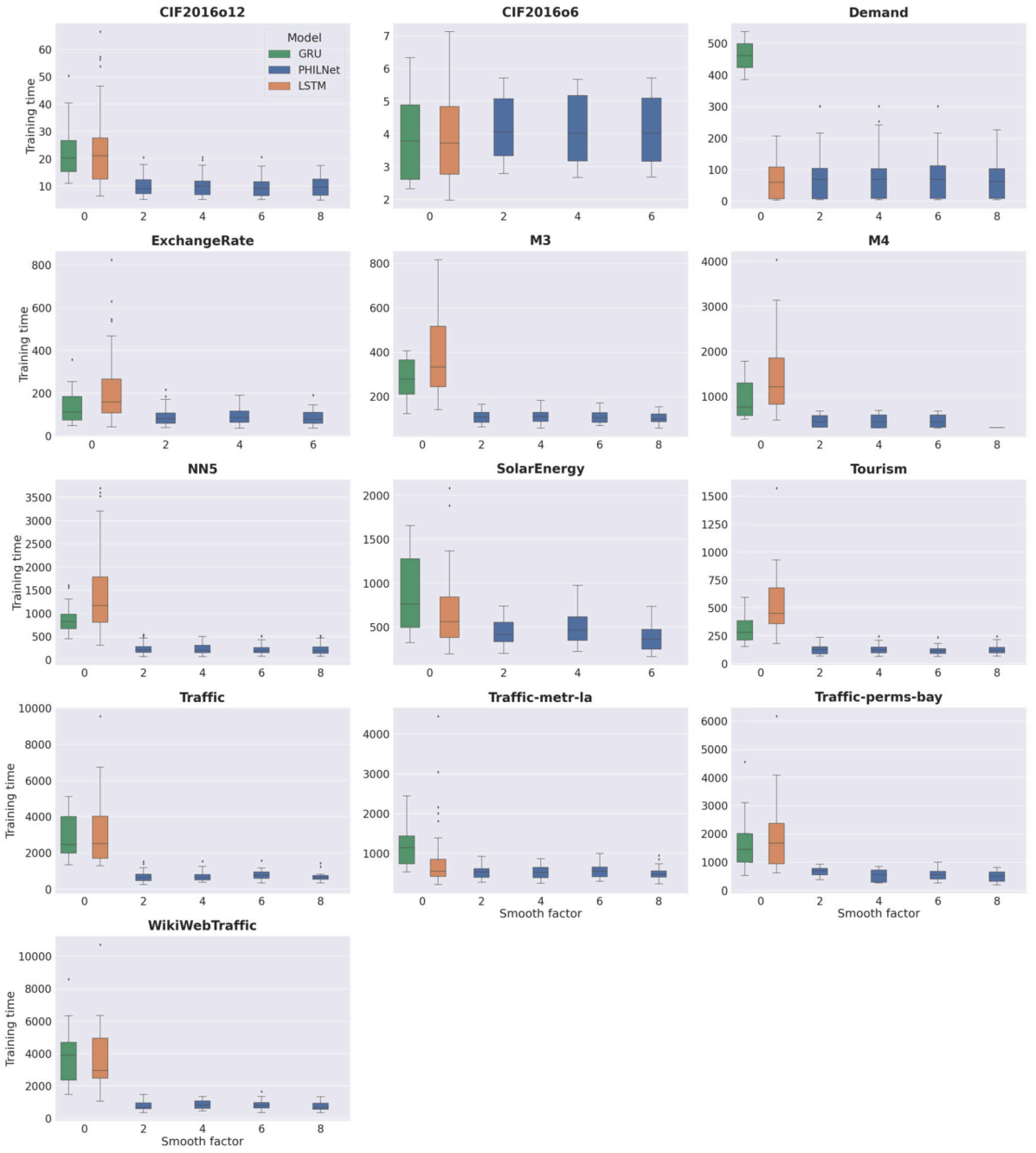
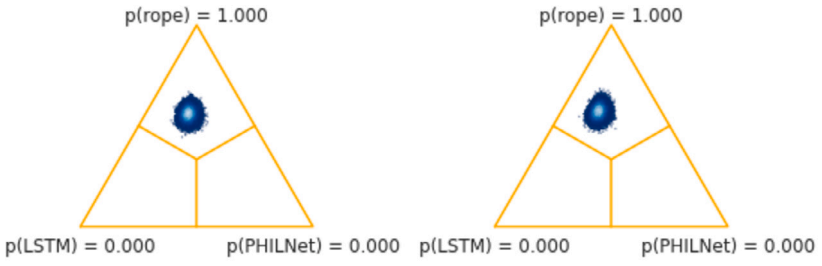


Fig. 4. Box plot of all the experiments training time per smooth factor PHILNet.

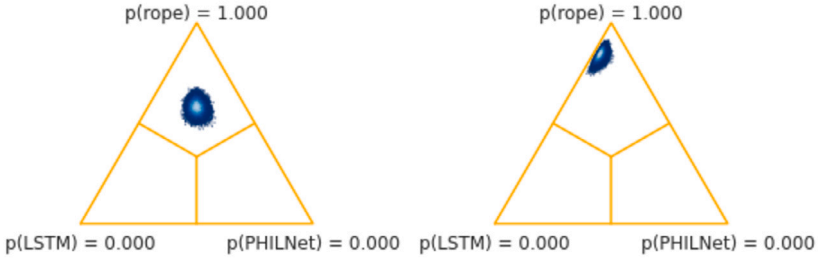
the smooth factor. This means that there are more configurations for PHILNet than for LSTM, so a paired comparison cannot be performed directly. The results for the LSTM model for all hyperparameters are compared with PHILNet for each smooth factor. Furthermore, since GRU is not competitive in terms of training time and WAPE with LSTM and PHILNet, only LSTM is selected in the comparison.

Table 8 shows the probability that LSTM has a higher WAPE than PHILNet (LSTM), PHILNet has a higher WAPE than LSTM (PHILNet) and a similar WAPE (rope). We set a threshold of 75% probability to make assumptions about the results, which means that results that do not exceed the threshold are considered undetermined. The value for each dataset and the smooth factor have been colored green or orange, depending on whether the probability is above or below the threshold, respectively. (For interpretation of the references to color please refer to the web version of this article.)

Most of the probabilities fall into the rope, which means that PHILNet has an error similar to LSTM. The case of the CIF16o12 dataset is considered undetermined, the difference between LSTM, PHILNet, and rope does not exceed the threshold for any smooth



(a) Smooth factor 2. (b) Smooth factor 4.



(c) Smooth factor 6. (d) Smooth factor 8.

Fig. 5. Probability of having a greater WAPE between LSTM and PHILNet for different smooth factors.

Table 9
Probability of LSTM having greater training time than PHILNet (LSTM), probability of PHILNet having greater training time than LSTM (PHILNet) and probability of PHILNet having similar training time than LSTM (rope) for each smooth factor.

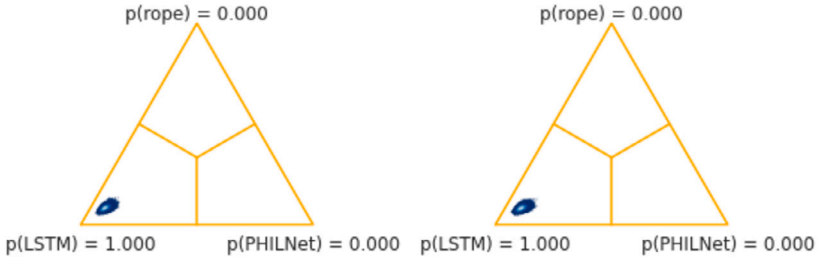
Dataset	Smooth factor											
	2			4			6			8		
	LSTM	rope	PHILNet	LSTM	rope	PHILNet	LSTM	rope	PHILNet	LSTM	rope	PHILNet
Demand	0.15	0.05	0.80	0.22	0.04	0.73	0.12	0.05	0.83	0.12	0.05	0.83
CIF16o6	0.00	0.00	1.00	0.00	0.01	0.99	0.00	0.01	0.99	–	–	–
CIF16o12	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
M3	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
M4	0.91	0.02	0.08	0.90	0.02	0.08	0.95	0.01	0.04	0.91	0.02	0.07
NN5	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
SE	1.00	0.00	0.00	0.98	0.01	0.01	1.00	0.00	0.00	–	–	–
ER	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	–	–	–
Tourism	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
Traffic	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
TML	1.00	0.00	0.00	0.99	0.01	0.00	0.99	0.01	0.00	0.99	0.01	0.00
TPB	0.96	0.02	0.02	0.98	0.01	0.01	0.96	0.02	0.02	0.97	0.01	0.01
WWT	0.99	0.01	0.00	0.99	0.01	0.00	0.99	0.01	0.00	0.99	0.00	0.00

factor. In the Traffic dataset, the results are undetermined for smooth factors 2 and 8 but fall in the rope for 4 and 6. In the Solar Energy (SE) dataset, PHILNet has more WAPE than LSTM. Finally, in the CIF16o6 and Tourism datasets, PHILNet has fewer errors than LSTM.

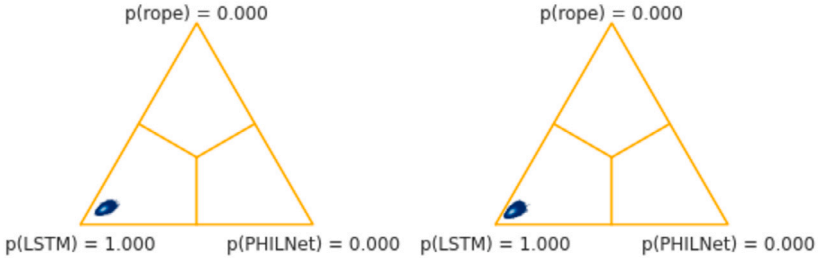
Fig. 5 shows the WAPE comparison between the probabilities LSTM, rope, and PHILNet as p(LSTM), p(rope), and p(PHILNet) respectively for all datasets. We have four equilateral triangles, one for each smooth factor for PHILNet as all the hyperparameters are shared with LSTM except for the SmoothFactor. The figure shows how all points fall into the rope assigning a probability of 1. This supports the first idea of our hypothesis: PHILNet has similar results to LSTM in general.

Table 9 shows the probabilities as the previous table but using the training time metric. In 11 of 13 datasets, the probability showed LSTM has greater training time for each Smooth factor value. In the Demand and CIFo2016o6 datasets, the probability was higher for PHILNet having more training time than LSTM. Note that almost all probabilities are above 90% showing clearly the training time reduction.

Fig. 6 shows the probabilities as in the previous figure but using the training time metric. The figure clearly shows that the training time in LSTM is greater than PHILNet in general.



(a) Smooth factor 2. (b) Smooth factor 4.



(c) Smooth factor 6. (d) Smooth factor 8.

Fig. 6. Probability of having a longer training time between LSTM and PHILNet for different smooth factors.

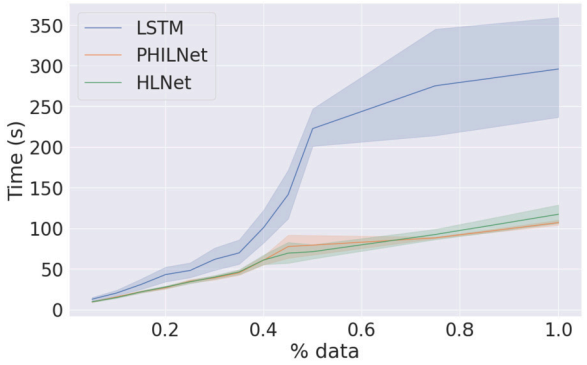


Fig. 7. Training time analysis by data size in M3 dataset.

5.4. Data size analysis

We used the M3 as the computation time is not too expensive and the training time reduction is good enough to see the difference between PHILNet and LSTM. The data has been sampled from 5% to 100% of available data. When we sample the 5% of available data, for example, we take it from more recent instances. As we increase the available data, the oldest data are introduced to the training. We use this sampling method because we consider it the most realistic scenario in practice.

Fig. 7 shows how the training time scales in our method as more data are added to our method. We can observe how the methods start to have the same training time in the beginning, but as we increase the training data the LSTM increases considerably faster than PHILNet.

Fig. 8 shows the evolution of the training time in the M3 dataset by layers and units. The training time in LSTM and PHILNet seems to follow an exponential evolution. However, PHILNet has less training time in general than LSTM maintaining below in almost all cases.

6. Conclusions and future works

In this paper, we introduced PHILNet, a novel method for multi-step forecasting evaluated over 13 different datasets and compared with six important methods in the context of TSF. Thus, we developed a new architecture based on LSTM layers that allows learning

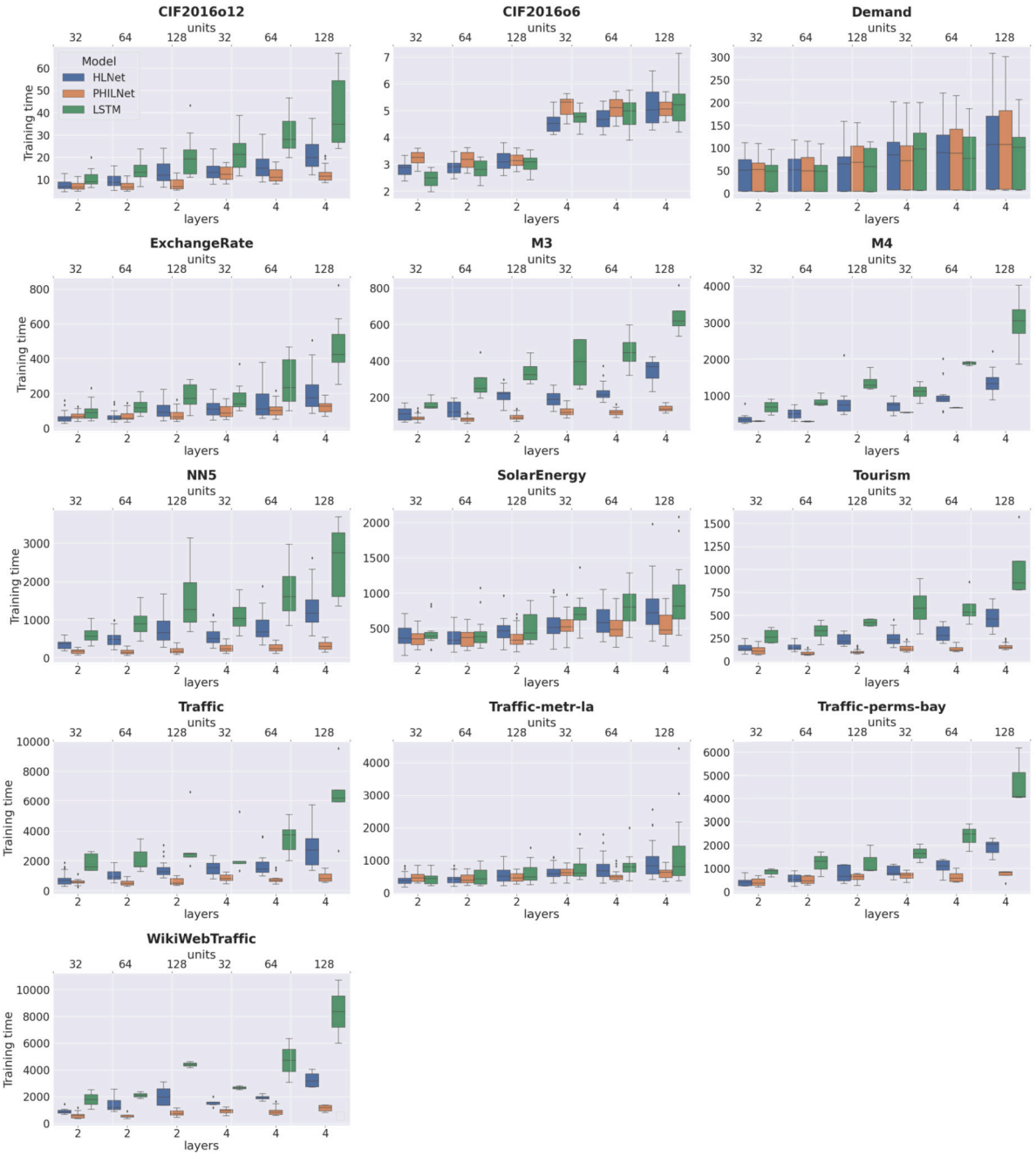


Fig. 8. Training time by layers and units.

from simpler to more complex tasks obtaining a hierarchical representation of the input. The methodology is done by smoothing the original input and transferring the extracted knowledge to subsequent levels. The results have shown a great improvement in terms of efficiency using less training time maintaining a good efficacy, which leads to the use of deeper architectures, optimizing the hyperparameters or training several architectures with less computation. The improvement in efficiency is achieved by assigning specific tasks with increasing difficulty to different levels. This fact accelerates the convergence time, since learning tasks gradually help to provide relevant information to harder tasks quickly. In future work, other smoothing strategies could be followed instead of the moving average. Additionally, the development of a custom loss function with different weights for each level and evolving with the number of epochs is proposed. Furthermore, the inclusion of a technique that reduces the possible effect of negative transfer can be beneficial. Finally, we can use more types of simple tasks that support the main task inspired by self-learning and auxiliary learning.

CRediT authorship contribution statement

M.J. Jiménez-Navarro: Conceptualization, Software, Visualization, Writing – original draft. **M. Martínez-Ballesteros:** Formal analysis, Resources, Writing – review & editing. **F. Martínez-Álvarez:** Data curation, Investigation, Writing – review & editing. **G. Asencio-Cortés:** Project administration, Supervision, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data is available in our article as URLs to the original access. Additionally, a repository with an URL to the datasets used in this work is provided.

Acknowledgements

The authors would like to thank the Spanish Ministry of Science and Innovation for the support under the projects PID2020-117954RB and TED2021-131311B, the European Regional Development Fund and Junta de Andalucía for projects PY20-00870, PYC20 RE 078 USE and UPO-138516. Funding for open access publishing: University of Seville/CBUA.

References

[1] H. Abbasimehr, R. Paki, Improving time series forecasting using LSTM and attention models, J. Ambient Intell. Humaniz. Comput. 13 (2022) 673–691.
[2] D. Afolabi, S. Guan, K. Man, P.W.H. Wong, X. Zhao, Hierarchical meta-learning in time series forecasting for improved interference-less machine learning, Symmetry 9 (2017) 283.
[3] G. Athanasopoulos, R.J. Hyndman, H. Song, D.C. Wu, The tourism forecasting competition, Int. J. Forecast. 27 (2011) 822–844.
[4] F. Bahrpeyma, A. Mccarren, M. Roantree, Multi-resolution forecast aggregation for time series in agri datasets, in: Proceedings of the Irish Conference on Artificial Intelligence and Cognitive, 2017, pp. 1–12.
[5] S. Bai, J.Z. Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, 2018.
[6] A. Belhadi, Y. Djenouri, D. Djenouri, J.C. Lin, A recurrent neural network for urban long-term traffic flow forecasting, Appl. Intell. 50 (2020) 3252–3265.
[7] A. Benavoli, G. Corani, J. Demsar, M. Zaffalon, Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis, J. Mach. Learn. Res. 18 (2017) 2653–2688.
[8] A. Benavoli, F. Mangili, G. Corani, M. Zaffalon, F. Ruggeri, A Bayesian Wilcoxon signed-rank test based on the Dirichlet process, in: Proceedings of the International Conference on Machine Learning, 2014, pp. 2703–2713.
[9] J. Cao, Z. Li, J. Li, Financial time series forecasting model based on ceemdan and lstm, Phys. A, Stat. Mech. Appl. 519 (2019) 127–139.
[10] CDT, California department of transportation, https://pems.dot.ca.gov/, 2015.
[11] K. Cho, B. van Merriënboer, D. Bahdanau, Y. Bengio, On the properties of neural machine translation: encoder–decoder approaches, in: Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Association for Computational Linguistics, 2014, pp. 103–111.
[12] G. Corani, A. Benavoli, A Bayesian approach for comparing cross-validated algorithms on multiple data sets, Mach. Learn. (ISSN 1573-0565) 100 (2015).
[13] S.F. Crone, NN5 time series forecasting competition for neural networks, http://www.neural-forecasting-competition.com, 2008.
[14] Y. Djenouri, A. Belhadi, G. Srivastava, J.C. Lin, Hybrid graph convolution neural network and branch-and-bound optimization for traffic flow forecasting, Future Gener. Comput. Syst. 139 (2023) 100–108.
[15] Google, Web traffic time series forecasting competition, https://www.kaggle.com/c/web-traffic-time-series-forecasting, 2017.
[16] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[17] Y. He, J. Zhao, Temporal convolutional networks for anomaly detection in time series, J. Phys. Conf. Ser. (2019) 042050.
[18] P. Hewage, A. Behera, M. Trovati, E. Pereira, M. Ghahremani, F. Palmieri, Y. Liu, Temporal convolutional neural (TCN) network for an effective weather forecasting using time-series data from the local weather station, Soft Comput. 24 (2020) 16453–16482.
[19] H. Hewamalage, C. Bergmeir, K. Bandara, Recurrent neural networks for time series forecasting: current status and future directions, Int. J. Forecast. 37 (2021) 388–427.
[20] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780.
[21] H. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. Patel, R. Ramakrishnan, C. Shahabi, Big data and its technical challenges, Commun. ACM 57 (2014) 86–94.
[22] M.J. Jiménez-Navarro, G. Asencio-Cortés, A. Troncoso, F. Martínez-Álvarez, HLNet: a novel hierarchical deep neural network for time series forecasting, in: Proceedings of the International Conference on Soft Computing Models in Industrial and Environmental Applications, 2021, pp. 721–727.
[23] H.J. Jo, W.J. Kim, H.K. Goh, C. Jun, An improved time-series forecasting model using time series decomposition and gru architecture, in: International Conference on Neural Information Processing, Springer, 2021, pp. 587–596.
[24] A. Koenecke, A. Gajewar, Curriculum learning in deep neural networks for financial forecasting, in: Lecture Notes in Computer Science, vol. 11985, 2020, pp. 16–31.
[25] N. Kourentze, F. Petropoulos, J.R. Trapero, Improving forecasting by estimating time series structural components across multiple frequencies, Int. J. Forecast. 30 (2014) 291–302.
[26] N. Kourentzes, F. Petropoulos, Forecasting with multivariate temporal aggregation: the case of promotional modelling, Int. J. Prod. Econ. 181 (2016) 145–153.
[27] G. Lai, W. Chang, Y. Yang, H. Liu, Modeling long- and short-term temporal patterns with deep neural networks, in: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, 2018, pp. 95–104.
[28] P. Lara-Benítez, M. Carranza-García, J.M. Luna-Romera, J.C. Riquelme, Temporal convolutional networks applied to energy-related time series forecasting, Appl. Sci. 10 (2020) 2322.
[29] P. Lara-Benítez, M. Carranza-García, J.C. Riquelme, An experimental review on deep learning for time series forecasting, Int. J. Neural Syst. 31 (2021) 2130001.

[30] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y. Wang, X. Yan, Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting, in: International Conference on Neural Information Processing, vol. 32, 2019.
[31] B. Lim, S.Ö. Arık, N. Loeff, T. Pfister, Temporal fusion transformers for interpretable multi-horizon time series forecasting, Int. J. Forecast. 37 (2021) 1748–1764.
[32] T. Lin, B. Horne, P. Tino, C.L. Giles, Learning long-term dependencies in NARX recurrent neural networks, IEEE Trans. Neural Netw. 7 (1996) 1329–1338.
[33] S. Makridakis, M. Hibon, The M3-Competition: results, conclusions and implications, Int. J. Forecast. 16 (2000) 451–476.
[34] S. Makridakis, E. Spiliotis, V. Assimakopoulos, The M4 Competition: 100,000 time series and 61 forecasting methods, Int. J. Forecast. 36 (2020) 54–74.
[35] T. Mordan, N. Thome, G. Henaff, M. Cord, Revisiting multi-task learning with rock: a deep residual auxiliary block for visual detection, in: Proceedings of the Advances in Neural Information Processing Systems, 2018, pp. 1317–1329.
[36] B.N. Oreshkin, D. Carpov, N. Chapados, Y. Bengio, N-BEATS: neural basis expansion analysis for interpretable time series forecasting, arXiv, 2019.
[37] REE, Red Eléctrica de España, www.ree.es, 2015.
[38] D. Salinas, V. Flunkert, J. Gasthaus, T. Januschowski, DeepAR: probabilistic forecasting with autoregressive recurrent networks, Int. J. Forecast. 36 (2020) 1181–1191.
[39] M. Stepnicka, M. Burda, Computational Intelligence in Forecasting (CIF), https://irafm.osu.cz/cif/main.php, 2016.
[40] S.J. Taylor, B. Letham, Forecasting at scale, PeerJ (2017).
[41] J.F. Torres, D. Hadjout, A. Sebaa, F. Martínez-Álvarez, A. Troncoso, Deep learning for time series forecasting: a survey, Big Data 9 (2021) 3–21.
[42] Y. Wei, J. Jang-Jaccard, W. Xu, F. Sabrina, S. Camtepe, M. Boulic, LSTM-autoencoder based anomaly detection for indoor air quality time series data, arXiv, 2022.
[43] X. Wu, B. Shi, Y. Dong, C. Huang, L. Faust, N.V. Chawla, RESTFul: resolution-aware forecasting of behavioral time series data, in: Proceedings of the ACM International Conference on Information and Knowledge Management, 2018, pp. 1073–1082.
[44] X. Wu, D. Zhang, C. Guo, C. He, B. Yang, C.S. Jensen, AutoCTS: automated correlated time series forecasting, Proc. VLDB Endow. 15 (2021) 971–983.
[45] M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: Proceedings of the European Conference on Computer Vision, 2014, pp. 818–833.
[46] X. Zhang, C. Zhong, J. Zhang, T. Wang, W.W. Ng, Robust recurrent neural networks for time series forecasting, Neurocomputing 526 (2023) 143–157.
[47] Y. Zhang, Solar power data for integration studies (NREL), https://www.nrel.gov/grid/solar-integration-data.html, 2007.
[48] C. Zheng, X. Fan, C. Wang, J. Qi, GMAN: a graph multi-attention network for traffic prediction, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2019, pp. 1234–1241, prediction.
[49] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, W. Zhang, Informer: beyond efficient transformer for long sequence time-series forecasting, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2021, pp. 11106–11115.