Journal of Big Data

METHODOLOGY

Open Access

A new deep learning architecture with inductive bias balance for transformer oil temperature forecasting

Manuel J. Jiménez-Navarro[1*], María Martínez-Ballesteros[1], Francisco Martínez-Álvarez[2] and Gualberto Asencio-Cortés[2]

*Correspondence: mjimenez3@us.es

[1] Department of Computer Science, University of Seville, Seville, Spain
[2] Data Science and Big Data Lab, Pablo de Olavide University, Seville, Spain

Abstract

Ensuring the optimal performance of power transformers is a laborious task in which the insulation system plays a vital role in decreasing their deterioration. The insulation system uses insulating oil to control temperature, as high temperatures can reduce the lifetime of the transformers and lead to expensive maintenance. Deep learning architectures have been demonstrated remarkable results in various fields. However, this improvement often comes at the cost of increased computing resources, which, in turn, increases the carbon footprint and hinders the optimization of architectures. In this study, we introduce a novel deep learning architecture that achieves a comparable efficacy to the best existing architectures in transformer oil temperature forecasting while improving efficiency. Effective forecasting can help prevent high temperatures and monitor the future condition of power transformers, thereby reducing unnecessary waste. To balance the inductive bias in our architecture, we propose the Smooth Residual Block, which divides the original problem into multiple subproblems to obtain different representations of the time series, collaboratively achieving the final forecasting. We applied our architecture to the Electricity Transformer datasets, which obtain transformer insulating oil temperature measures from two transformers in China. The results showed a 13% improvement in MSE and a 57% improvement in performance compared to the best current architectures, to the best of our knowledge. Moreover, we analyzed the architecture behavior to gain an intuitive understanding of the achieved solution.

Keywords: Electricity transformer, Insulate oil, Time series, Efficiency, Efficacy, Forecasting, Deep learning

Introduction

Demand forecasting is a challenging field that has been extensively studied in the literature [1, 2]. Long-term demand forecasting remains an open field where a perfect solution has yet to be found. Poor demand prediction can damage power transformers, which is why several studies focus on analyzing the state of power transformers. The insulation oil is a crucial component in the maintenance of power transformers to regulate temperature, as high temperatures can reduce the useful life of transformer

[3, 4]. Forecasting the insulation oil helps to analyze the future state of the power transformer. By setting a defined threshold, it becomes possible to alert when the transformer insulation oil temperature is going to be harmful before it happens, or to evaluate the transformer's safety and avoid unnecessary waste.

However, forecasting the insulation oil presents a challenge in terms of both efficacy and efficiency. Efficacy is necessary to accurately predict future behavior, while efficiency is necessary to scale the problem to real-world scenarios. Different power transformers may have different behaviors, which divides the problem into several subproblems. Each subproblem requires a solution that may require large computational resources to optimize. For this reason, the solution must be as effective and efficient as possible to improve the scalability of the solution.

Time series data is one of the most common data types used in the industry. It is defined as a set of records ordered by time. Time series forecasting has a significant impact on society due to its importance in a variety of real-world applications such as electricity, sales, air pollution, and more. The primary goal of time series forecasting is to predict future records to make plans, mitigate damages, save resources, and so on. Past records are used to predict future records due to the causal nature of time series. The number of features in a time series determines whether it is defined as univariate, with one single feature, or multivariate, with several features [5]. In addition, two types of forecasting can be considered: one-step, where the model predicts one record in the future, and multi-step, where the model predicts several records in the future.

Deep learning is rapidly advancing, with new architectures being developed almost every year that improve the state-of-the-art in supervised fields such as computer vision [6, 7], natural language processing [8, 9], and time series forecasting [10, 11]. However, recent research suggests that achieving state-of-the-art results requires a significant increase in computational resources. This trend can be seen in recent studies that introduce new architectures like the Transformer [12], which suffer from issues such as quadratic time complexity and high memory usage [13]. Typically, in public competitions, the best models are usually large architectures or ensembles of several architectures [14], which require more computational resources. Moreover, experts have been using the carbon footprint metric for a long time to raise awareness about the environmental impact of training deep learning architectures. In addition, training, experimentation, and inference with large and/or multiple architectures can be challenging to scale in resource-limited environments, slowing down academic and industry development. For these reasons, we focus on developing an architecture that can either improve or compete with the current best architectures in terms of efficacy while reducing computational resources.

Since there are not many works that consider the problem of transformer oil temperature prediction in the literature [15] and reproducing/comparing the results of different approaches can be challenging. Therefore, we compared our proposed method using one of the most widely used dataset in this field, namely the Electricity Transformer Temperature (ETT) dataset [13], which has been used by more than 50 works. In this context, the most effective architectures correspond to the works that obtained the best results in terms of efficacy in this specific dataset, some of them are analyzed in the "Related

works" section. The results seem to indicate that our method obtains a remarkable efficacy and efficiency achieving the best results.

We present a novel deep learning architecture for multi-step forecasting, named Smooth Residual Connections Network (SRCNet),[1] which prioritizes efficacy and efficiency. SRCNet employs convolution layers as building blocks and introduces a new mechanism known as smooth residual stacking. This mechanism builds upon the doubly stacking approach developed by Oreshkin et al. [16] and is analyzed in the "Experimentation" section, which has been shown to yield remarkable results when combined with residual connections in convolutional layers, as demonstrated by Liu et al. [17] in the SCINet architecture. By introducing an inductive bias, the proposed mechanism enhances the efficacy of SCINet without imposing any significant restrictions on the architecture. The motivation for leveraging inductive bias will be discussed in the "Methods" section.

Our method is compared to the approach proposed by Liu et al. [17], which, to our knowledge, reported the best results in the ETT dataset. In contrast to their work, we employ a convolutional neural network with the smooth residual stacking mechanism, which has demonstrated remarkable improvements in terms of efficacy and efficiency in several scenarios. Additionally, we include the results of six baseline architectures in the comparison due to their similarity to our approach.

The main contributions of our work are as follows:

- We introduce a new smooth residual stacking mechanism that improves the efficacy and efficiency of the architecture in several scenarios.
- We use a time series decomposition with less inductive bias than previous studies, which maintains both efficacy and efficiency.
- Our application of the proposed architecture to the Electricity Transformer Temperature datasets outperforms the compared architectures in terms of both efficiency and efficacy.

The remaining sections of this work are organized as follows. In the "Related works" section, we provide a review of recent literature on time series forecasting methods with a focus on efficiency. The "Methods" section describes the proposed architecture in detail. The Experiments section presents the datasets used, experimental settings, and evaluation metrics. Next, in the "Results and discussion" section, we present and analyze the results obtained and the behavior learned by the model. Finally, we discuss the conclusions and future directions of this research.

Related works

To address the efficiency issue in making effective forecasting for ETT dataset, several architectures have been developed and applied using deep learning approaches. One of the main challenges in this dataset is the need for a large window size, which results in efficiency issues that considerably increase computational costs. Previous works have

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

analyzed this problem using two approaches: the sparse approach and the inductive bias approach.

Sparse approach

A recently proposed approach is to use the self-attention mechanism [12] for time series forecasting. However, it suffers from quadratic time complexity and high memory usage that make its application difficult. The sparse approach attempts to improve the efficiency of the self-attention mechanism by adding downsampling of the records in the input window using some heuristic. In general, these studies are based on the sparse nature of the data, which may not be fully proven. Sparsity leads to an information bottleneck that limits the amount of information available based on some criteria. In our work, we do not use a self-attention approach to avoid increasing computational resources or information loss by adding sparsity.

The first proposal applied to the ETT dataset was the Informer architecture developed by Zhou et al. in 2020, with the proposed ProbSparse Self-Attention [13]. This mechanism avoids storing and calculating the dot product between all queries and keys based on the hypothesis that few pair query-keys contribute to most self-attention scores. Using this sparsity hypothesis, the queries are truncated based on the top pairs with a greater contribution from a subset of pairs selected by a probabilistic method.

Klimek et al. proposed QuerySelector in 2022 [18] as an improvement to Informer. This architecture uses the same hypothesis as in the previous work, but instead of subsampling the pairs based on a probabilistic method, they subsample the query and keys based on a defined hyperparameter. This hyperparameter allows one to improve the control of the sparsity and the reproducibility of the problem.

Du et al. developed the SCformer architecture in 2022 [19] with the SCAttention mechanism. The architecture uses an encoder-decoder architecture similar to the original transformer replacing the self-attention mechanism with SCAttention. The key idea is to divide the input window into segments of the same size according to the period. Attention scores are computed using the correlation between each segment of queries and keys, which reduces the data precision, but reduces the computation.

Another approach to obtain a sparse representation of the input data would be to apply some dimensionality reduction technique. Žagar et al. [20] applied the principal component analysis to spectral data obtaining a remarkable reduction. Despite being widely used, this process still required storing and processing all the time series data to build the principal components. In addition, mechanism like self-attention which requires sequential data may not be used.

Inductive bias approach

Some architectures introduce an inductive bias based on some hypothesis about the nature of the problem. For example, some solutions divide the problem into trend and seasonality, resulting in competitive results. However, the inductive bias may restrict solutions not inside the solution subspace, degrading the efficacy and generality. The effect of inductive bias is discussed in the "Methods" section. In this work, we use a different inductive bias approach, which does not impose a strong limitation on the problem.

Woo et al. proposed CoST [21], an approach to learning the trend and seasonality of time series through contrastive learning. This approach develops two contrastive loss functions and two components for trend and seasonality. The Trend Feature Disentangler (TFD) uses a mixture of autoregressive experts to represent the trend and a variation of MoCo contrastive loss [22]. The Seasonal Feature Disentangler uses a layer with learnable Fourier transformations and a contrastive loss for the amplitude and phase of each representation.

Wu et al. proposed the Autoformer [23], an architecture that decomposes time series into trend and seasonality using the series decomposition and autocorrelation block. The series decomposition block uses moving averages to extract the trend and seasonality from the input. The autocorrelation block is used on period-based dependencies using autocorrelation to group similar series. This allows the series to focus on specific periods instead of specific records.

Liu et al. developed the Sample Convolution and Interaction Network (SCINet) [17], an architecture that takes advantage of multiple representations obtained at different temporal resolutions. The principal block is the SCI-block which splits the time series into even and odd records and extracts features for both using interaction learning. Then, all the learned features are used to produce the final forecasting.

In some scenarios, the inductive bias is introduced by including information about a physical model that supports the estimations. Kosanić et al. [24] developed a survey where they illustrate different approaches to geophysics forecasting, in which the combination of physical and statistical models is described, using examples. However, in some scenarios, the physical model may not be available, which limits the application of this inductive bias approach.

Methods

In this section, we provide a detailed explanation of the main components and hypotheses of the architecture. Specifically, the "Hypothesis" subsection discusses the principal hypothesis and motivation behind the architecture. The "Nomenclature" subsection defines all symbols used to describe the architecture. The "Embedding" subsection provides a detailed description of the embedding layer. Lastly, the "Smooth Residual Block" section introduces the basic component in the smooth residual stacking mechanism.

Hypothesis

Inductive bias is used by some machine learning algorithms to design a method that can solve a problem based on certain assumptions [25]. These assumptions impose restrictions or regularization that prioritize a solution subspace over others. In the context of deep learning algorithms, these assumptions can be included in the architecture design by the transformations applied by the learned weights [26].

The hypothesis of our proposed architecture relates to the effect of inductive bias on the efficacy of the architecture, which we refer to as the inductive bias-efficacy balance hypothesis.

Hypothesis 1 (H1) Strong restrictions introduced by inductive bias may improve efficiency by reducing the probability of finding a solution outside the prioritized solution space.

The proposed mechanism is motivated by our hypothesis of bias-efficacy balance 1 (H1). The hypothesis suggests that incorporating an inductive bias based on assumptions into the architecture design improves efficiency by prioritizing a solution subspace and penalizing or removing solutions that do not fit the design. This increases the likelihood of converging into a solution within the prioritized subspace. Conversely, an architecture with less bias has to explore a larger solution space, potentially increasing the time needed to converge to a local minimum.

However, limiting or prioritizing the solution space does not guarantee that solutions outside the subspace perform worse than solutions inside it. This may lead to suboptimal solutions, but mitigating the effect of inductive bias may help the architecture find an optimal solution that it could not previously discover. Our proposal aims to find a balance by incorporating enough inductive bias to improve efficiency without excessively prioritizing the suboptimal solution space, thereby preserving efficacy.

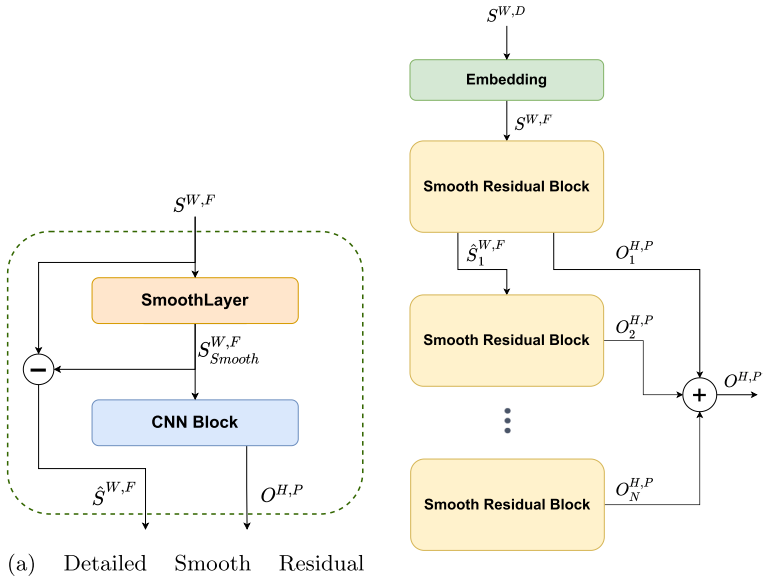
Previous studies have designed architectures for time series forecasting which decompose the input sequence into its constituent parts of trend, seasonality, and noise. Such architectures introduce a strong inductive bias that prioritizes a solution subspace which can be modeled by the combination of these components. In our work, we propose a decomposition of the input sequence that must satisfy the following properties:

1. Each decomposition must remove the complexity of the input window by removing its smoothed version.
2. The decomposition is performed sequentially, such that the next decomposition depends on the previous decomposition.
3. Each Smooth Residual Block focuses on one component without sharing information with other blocks.
4. Smooth Residual Blocks collaborate using the different decompositions to produce the final prediction.

Nomenclature

To understand the architecture, we need to detail the nomenclature used in this section and the following. The elements implied are as follows:

- W denotes the number of past records used to feed the architecture and generate the forecasting, also known as window size.
- D denotes the number of features in each record. Note that in univariate time series $D = 1$.
- F denotes the embedding dimension size used in the embedding layer.
- $S^{W,D}$ denotes the input window, which is a set of W time-ordered records with D features.
- $S^{W,F}$ denotes the encoded input window after embedding with F features.



(a) Detailed Smooth Residual Block. (b) SRCNet architecture.

Fig. 1 Diagram of the architecture developed and the Smooth Residual Block

- $S^{W,F}_{Smooth}$ denotes the smoothed version of the encoded input window.
- $\hat{S}^{W,F}$ denotes the result of the substrate of the original encoded input and the smooth version of the encoded input window.
- H denotes the number of future steps to forecast, also known as the horizon size. Note that in one-step forecasting $H = 1$.
- P denotes the number of predicted features. Note that it may or may not be equal to D.
- N denotes the number of Smooth Residual Blocks used in the architecture.
- $O^{H,P}$ denotes the partial forecast of the Smooth Residual Blocks. All partial forecasts are added to produce the final forecasts.

Embedding

The embedding layer is the first transformation applied to the input sequence, using a causal convolution layer that considers only the previous records to apply the convolution to a record at time t. To achieve causality, the input sequence is padded by replicating the first record $K - 1$ times, where K is the kernel size of the convolution layer. The embedding layer encodes the initial sequence with D features into a set of causal features F, where F is determined by the number of filters used in the convolution layer. Additionally, a positional encoding is added to provide relative information about the input sequence.

Smooth Residual Block

The Smooth Residual Block is a crucial component of the architecture. Its main purpose is to decompose an input sequence $S^{W,F}$ into N components, the sum of which is

$S^{W,H}$. To achieve this, causal convolutions are used as the primary layer, as previous studies have shown that Transformers do not outperform convolutions in time series forecasting [27].

Figure 1 shows the complete architecture developed and details the Smooth Residual Block which constitutes the main component.

The initial Smooth Residual Block produces a partial prediction by generating the $S^{W,F}_{Smooth}$ component for the embedded input window. It is important to note that the decomposition is applied to the encoded version of the input sequence, denoted as $S^{W,F}$. This allows the model to handle both univariate and multivariate time series, as all types are projected onto a latent space. Moreover, encoding the input may aid in identifying a latent space that captures the time series in a more informative feature space.

The component $S^{W,F}_{Smooth}$ is generated by the *SmoothLayer* which takes the previous sequence $S^{W,F}$ as input and outputs the component $S^{W,F}_{Smooth}$. Decomposition is achieved using moving averages, which allows us to obtain a smooth representation of the original sequence. By choosing an appropriate window size or repeating the moving average several times, we can obtain the trend. However, choosing a shorter window size applied once can result in an intermediate function with a combination of trend and seasonality where the noise is reduced. We are interested in this representation because we do not want to restrict the decomposition of the input window into trend and seasonality.

As we use moving averages, we need to define a hyperparameter for the window size in all Smooth Residual Blocks. The hyperparameter is the same for all layers except for the last one, whose window size is one and the sequence remains unchanged. This means that the last layer focuses only on the residuals of all previous layers, mostly obtaining the noise of the original sequence.

After the component $S^{W,F}_{Smooth}$ is generated in the first Smooth Residual Block, it is subtracted from the input sequence $S^{W,F}$. Then, the next block applies the same process using the residual $\hat{S}^{W,F}$ as the input window. This step is essential because it allows the blocks to focus on a different aspect of the sequence. Additionally, it is important to standardize the input to maintain each component within the distribution of the data when we subtract the component from the original sequence.

The component $S^{W,F}_{Smooth}$ is fed into two causal convolution layers that output the prediction $O^{H,P}_i$ for layer i. Finally, the output of each block is added to obtain the final prediction. As the total sum is used for the final prediction, the blocks collaborate using their respective representations of the input. This collaboration, combined with the decomposition explained above, motivates a collaborative division of the problem.

Experimentation

This section provides details on the training and evaluation process of different models for each dataset, along with the results obtained. The "Datasets" section analyzes the datasets used in the experimentation. The "Experimental settings" section lists the different models used for comparison and the hyperparameter space used for grid

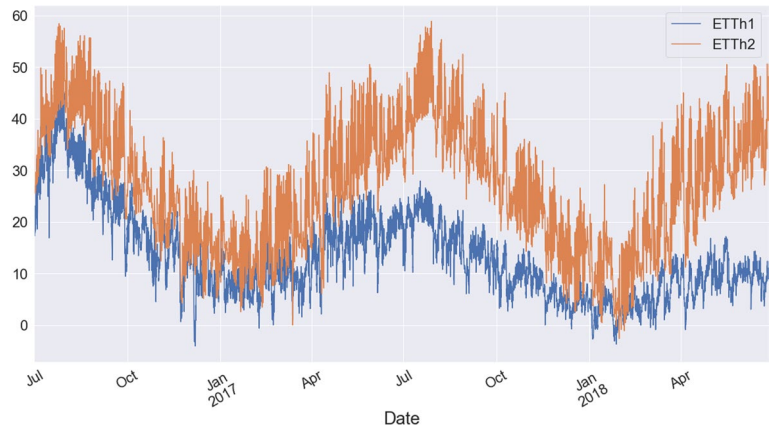


Fig. 2 Transformer oil temperature variable in ETTh1 and ETTh2 datasets

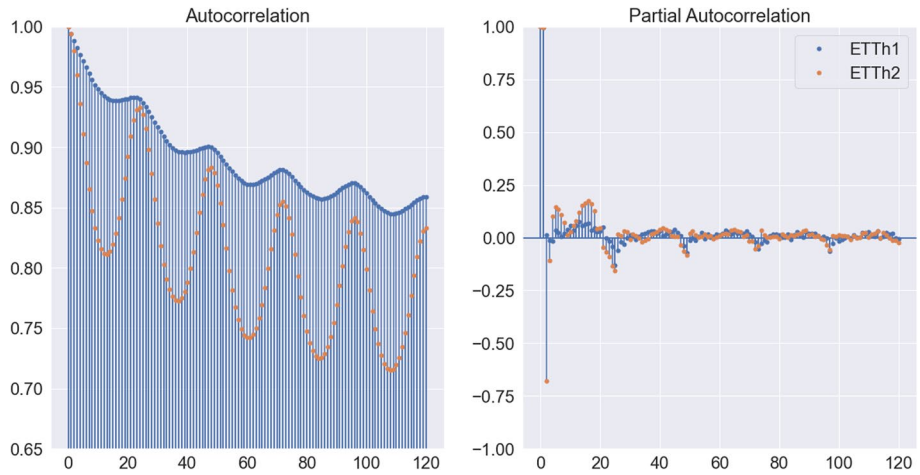


Fig. 3 ETTh1 and ETTh2 autocorrelation plots (left) and partial autocorrelation plots (right)

search. Finally, the "Evaluation metrics" section outlines the metrics proposed for evaluating the efficacy and efficiency of the models.

Datasets

The data studied in this work comes from the Electricity Transformer Datasets [13], which includes data from two electricity transformers at two stations in China. The data was collected on an hourly basis from July 2016 to July 2018. The target of both datasets is to forecast the transformer oil temperature (OT), which reflects the condition of the power transformer.

The data contains several features related to the transformer load. However, recent studies have shown that incorporating these features leads to increased errors compared to the univariate version. Therefore, we have chosen to focus solely on using the transformer oil temperature as input without any additional features.

Figure 2 shows the evolution of the transformer oil temperature in the datasets, while Fig. 3 shows the autocorrelation and partial autocorrelation plots. ETTh2 shows greater variance and stronger decays in the autocorrelation plot than the ETTh1 dataset.

Table 1 Hyperparameter space used during the grid search

Hyperparameter	Values
Window size	24, 48, 72, 96, 120
Embedding size	8, 16
# Neurons for each layer	8, 16, 32, 64
# Layers	1, 2, 3
Kernel size for each layer	3, 5, 7

The plots demonstrate a different behavior for both datasets, which encourages us to use different architectures that model the behavior of each dataset independently. This is one of the reasons for finding a method that is not only effective but also efficient.

Experimental settings

The processing applied to the datasets is the same as proposed in the original work [13], using identical divisions for training, validation, and testing to ensure fair comparisons with other architectures. The data is standardized using the training division to obtain the mean and standard deviation.

To ensure a fair comparison between the architectures explored, a grid search over the hyperparameters has been selected as the optimization method. The hyperparameters optimized during the grid search are defined in Table 1. This approach ensures that all explored architectures use the same hyperparameter space.

Some hyperparameters are not included in the grid search due to computational limitations or because they do not significantly affect the efficacy of the model. A batch size of 32 was chosen, since it is a standard value that generally yields good results. The number of epochs was set to 100, which is more than enough for the selected architectures to converge. The model is considered to have converged if the validation error does not decrease for 10 consecutive epochs. Once the model converges, the training stops, and the weights with the lowest error are restored. A prediction horizon of 24 future steps is considered sufficient. To reduce the influence of randomness, all experiments for each combination of hyperparameters were repeated three times, minimizing the impact of abnormal results.

The models selected to compare with SRCNet are: ARIMA as a classical method, the Informer architecture which originally published the ETT dataset, a convolutional neural network (CNN) used as the baseline, NBEATS to compare our proposed mechanism with theirs, SCINet, known for its excellent efficacy results, and HLNet, the predecessor of SRCNet.

The ARIMA and Informer architecture results are obtained from the work of Liu et al. [17] as the experimentation is reproduced using the same settings.

A CNN consists of a set of convolutional layers with a final fully-connected layer to transform the output into predictions. This simple model serves as a baseline for comparing with the other methods.

The NBEATS [16] model is based on blocks called BEAT, which are made up of a set of fully-connected layers. Each BEAT obtains a decomposition of the input sequence called the backcast and partial forecasting. The backcast is subtracted from

the window, and the result is fed into the next BEAT. The partial components proposed in the method decompose the window sizes between trend and seasonality. The final prediction is the sum of each forecast obtained in each BEAT.

The SCINet [17] model primarily uses convolutional layers to generate the forecast. The main mechanism is interaction learning, which subsamples the input window and divides it between odd and even sample positions. Interaction learning is applied to the subsamples, generating the features that are concatenated to obtain the forecasting.

The HLNet [28] model employs the LSTM [29] as the primary layer. This model divides the problem into difficulty levels, assigning each level to a specific layer with its own output. The final output feeds the representation of each level to the final fully-connected layer.

Evaluation metrics

This document considers two types of metrics: efficacy and efficiency. Efficacy metrics measure the accuracy of our predictions compared to the actual behavior of the time series. Efficiency metrics, on the other hand, measure the resources required to develop and deploy the solution.

The notation used for the metrics is as follows:

- Let t be the prediction moment.
- Let H be the number of future events to forecast (horizon).
- Let y_{t+h} be the predicted value for the moment $t + h$.
- Let $\hat{y}_{t+h}$ be the real value for $t + h$.
- Let N be the number of instances used to evaluate the metric.

The Mean Absolute Error (MAE) was used as the efficacy metric due to its simplicity. This metric obtains the average over the absolute error for all instances.

$$MAE = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{H}\sum_{h=1}^{H}|y_{t+h} - \hat{y}_{t+h}|\,, \tag{1}$$

Here, $|y_{t+h} - \hat{y}_{t+h}|$ denotes the absolute value of the difference between the prediction and the actual value. The error is averaged over all horizons.

As for efficiency metrics, we used the training time and the number of model parameters. Training time is important because it is the stage where the model consumes the most resources, and we need to run several experiments to find the optimal hyperparameters. The time is measured in minutes from the start of training to convergence of the model, which is defined as the point where there is no improvement in the validation dataset after 10 epochs. The number of model parameters is particularly relevant in resource-limited contexts where memory is scarce and large or multiple models are not feasible.

Table 2 MAE metric for the best models for every window size and dataset tested

Dataset	Window	ARIMA	Informer[a]	CNN[a]	HLNet	NBEATS	SCINet	SRCNet
ETTh1	24	0.290	0.247	0.172 (±0.030)	0.197 (±0.014)	0.201 (±0.012)	0.186(±0.006)	**0.138** (±**0.010**)
	48			0.167(±0.015)	0.193 (±0.019)	0.210 (±0.043)	0.166 (±0.021)	**0.137** (±**0.006**)
	72			0.139(±0.013)	0.202 (±0.007)	0.152 (±0.009)	0.149 (±0.013)	**0.131** (±**0.003**)
	96			0.144(±0.007)	0.205 (±0.014)	0.149 (±0.013)	0.146 (±0.010)	***0.129** **(±0.001)**
	120			0.156 (±0.012)	0.192 (±0.007)	0.15 (±0.018)	*0.138 (±0.010)	**0.131** (±**0.004**)
ETTh2	24	0.433	0.240	0.205 (±0.008)	0.333 (±0.016)	**0.197** (±**0.003**)	0.198(±0.006)	0.198(±0.006)
	48			0.198(±0.002)	0.339 (±0.011)	0.201 (±0.003)	0.201 (±0.009)	**0.195** (±**0.005**)
	72			*0.195* (±*0.004*)	0.334 (±0.010)	0.202 (±0.002)	0.196 (±0.009)	**0.189** (±**0.003**)
	96			0.200 (±0.005)	0.338 (±0.003)	0.207 (±0.003)	0.197(±0.004)	***0.193** (±*0.002*)
	120			0.201 (±0.005)	0.339 (±0.004)	0.206 (±0.002)	0.198(±0.004)	**0.194** (±**0.005**)

[a] These results have been obtained from the Liu et al. work. Note that the concept of input window does not exists in ARIMA, which explains why there are repeated values in each window. In addition, randomness does not affect ARIMA which justifies there is no standard deviation

Results and discussion

The "Efficacy and efficiency analysis" section discusses the results obtained in the evaluation process. "Output analysis" section analyzes the behavior learned by our proposal showing the partial predictions. Finally, "Statistical test" section applies a Bayesian statistical test to support the conclusions obtained.

Efficacy and efficiency analysis

This section is divided into two subsections to analyze the performance of our proposed methodology: efficacy and efficiency. In the first section, the efficacy is studied by comparing our proposal with the architectures specified in "Experimental settings" section, while in the second section the efficiency is examined focusing on the models with the best efficacy obtained in previous section.

Efficacy

The results are presented in terms of standardized efficacy and efficiency metrics. Each table displays the metrics for the models with the best efficacy found during the grid search. As experiments for each combination of hyperparameters are repeated three times, the average and standard deviation for the efficacy metric are reported.

Table 2 displays the standardized MAE metrics for the best models for each window size. The model with the best MAE for a given window size is indicated in bold, while the second-best result is underlined. The table also highlights the best MAE for all window sizes in each dataset for the top two models in bold and with an asterisk.

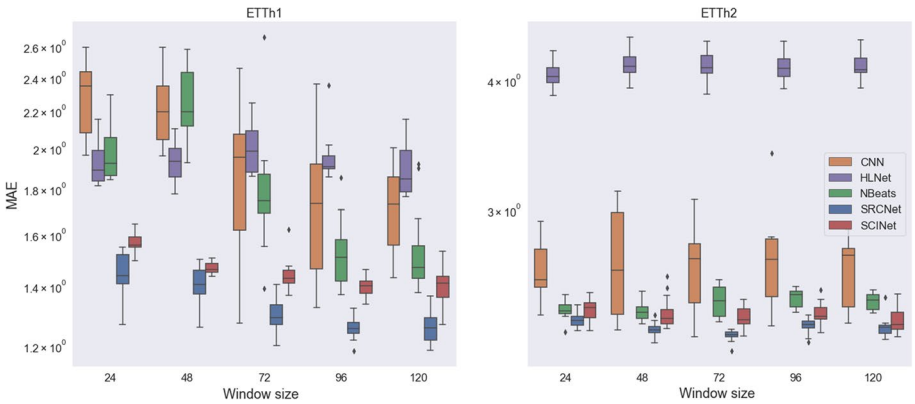


Fig. 4 MAE for each model and window size

As shown, SRCNet consistently delivers the best results across all datasets and window sizes, followed by SCINet. On the other hand, HLNet yields a higher error rate than all other models, indicating that the LSTM used inside is not suitable for achieving accurate results.

In the ETTh1 dataset, using window sizes of 24 and 48 appears to be insufficient to obtain all the information necessary for accurate forecasts, particularly for SCINet, NBEATS, and CNN, which seem to have a great difference in MAE with larger window sizes. In contrast, SRCNet and HLNet exhibit less variation between shorter and longer window sizes. Additionally, there is a clear relationship between window size and MAE for SRCNet and SCINet. Nevertheless, SCINet requires a window size of 120 to achieve the same MAE as SRCNet, which consumes five times more resources during inference. Moreover, SRCNet obtains the lowest standard deviation between experiments, which is ten times lower than that of SCINet in the best-case scenario.

In the ETTh2 dataset, the MAE is higher than that of ETTh1, indicating that this dataset is more challenging to predict. There is no apparent correlation between window size and the error obtained. In general, larger window sizes, such as 96 and 120 do not appear to improve the efficacy of the models. The best performance is achieved in CNN, SRCNet, and SCINet with a window size of 72, while HLNet and NBEATS perform best with a window size of 24. Furthermore, the difference between the best and worst MAE is lower in ETTh2 than in ETTh1. Additionally, the standard deviation between experiments appears to be lower for all models. Specifically, NBEATS and SRCNet have the lowest standard deviation, followed by SCINet. When comparing the configuration with the best efficacy, the standard deviation for NBEATS and SRCNet is three times better than that of SCINet.

Figure 4 shows the MAE for all models trained on the ETTh1 and ETTh2 datasets. The box plots show the distribution of the MAE obtained for each hyperparameter combination during the grid search. The figure seems to indicate that the results obtained by the CNN in Table 2 are outliers, which are not representative of the complete distribution shown in this figure.

For the ETTh1 dataset, CNN exhibits the highest interquartile range with a decreasing median as the window size increases. HLNet has a smaller interquartile range, but the model does not appear to take advantage of more samples in the window;

Table 3 Time required (minutes) to train the two models with the best efficacy

Dataset	Window size	SCINet	SRCNet
ETTh1	24	1.10	**0.50**
	48	2.14	**1.26**
	72	2.81	**1.90**
	96	3.70	**1.01**
	120	1.75	**1.04**
ETTh2	24	3.97	**3.23**
	48	5.23	**1.20**
	72	5.60	**1.57**
	96	5.05	**1.17**
	120	7.24	**1.25**

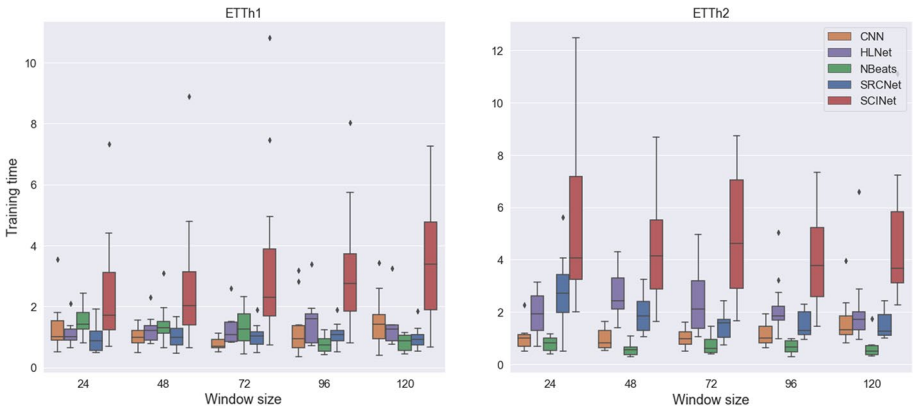


Fig. 5 Training time (in minutes) for each model and window size

the median remains almost constant regardless of the window size. NBEATS seems to improve efficacy as the window size increases, but the results are still worse than those of SRCNet, which performs better overall, particularly for larger window sizes. SCINet appears to have the better median and interquartile range for short window sizes, but SRCNet outperforms SCINet for larger window sizes.

In the ETTh2 dataset, errors are greater than in ETTh1, as shown in Table 2. HLNet achieves the worst efficacy by a large margin, regardless of window size. The best models are NBEATS, SCINet, and SRCNet, but the median for NBEATS decreases as the window size increases, suggesting that larger window sizes introduce noise that harms efficacy. In general, SRCNet obtains the global minimum, followed by SCINet, without a strong relationship between window size and error.

Efficiency

Table 3 displays the training times required for the best models to converge, analyzed for each window size and dataset. The training time with the lowest value for each

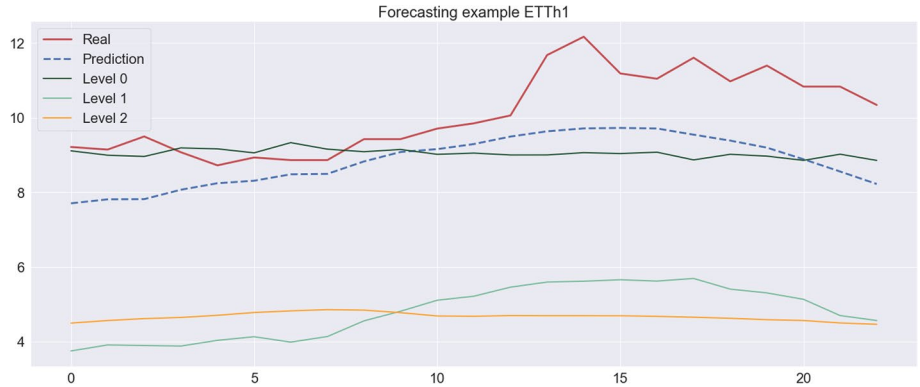


Fig. 6 Output produced by SRCNet for an instance of ETTh1 dataset. The forecast includes the output produced by the levels of SRCNet, which is summed up as the final forecast

window size is highlighted in bold. SRCNet, in general, shows better training times compared to SCINet.

In ETTh1, SCINet exhibits the poorest efficiency for all window sizes, indicating that the mechanism used in the architecture is highly inefficient. However, SRCNet achieves better results than SCINet with a time reduction of over 50.

In ETTh2, SCINet has the highest training times for each window size, which has increased significantly compared to the times obtained in ETTh1. This suggests that convergence in ETTh2 is considerably more challenging than in ETTh1. However, SRCNet does not show any significant increase in training time, except for a window size of 24 h.

Figure 5 illustrates the training time (in minutes) for all trained models.

For the ETTh1 dataset, CNN exhibits the lowest training times for window sizes of 24, 48, and 72. However, for larger window sizes, the training time increases significantly. HLNet displays the same trend as CNN, with increasing training times for larger window sizes. In contrast, NBEATS shows a different behavior, with lower training times for larger window sizes. While SRCNet does not achieve better training times than faster models like CNN or NBEATS, it remains almost constant, regardless of the window size.

For the ETTh2 dataset, NBEATS has the lowest median training time for all window sizes, followed by CNN. HLNet and SRCNet exhibit slightly greater training times for window sizes of 24 and 48, which then slightly decrease.

In general, the size of the training window does not significantly affect the time required for modeling until convergence. SCINet has the highest training times compared to all other models, regardless of the window size.

Output analysis

To gain a deeper understanding of the behavior learned by SRCNet, we analyze the partial predictions generated by the model. We used the best hyperparameters obtained for the ETTh1 dataset, and partial predictions were obtained from the test set.

Figure 6 illustrates the real behavior of the time series (red), the final prediction of SRCNet (dashed blue), and the partial predictions obtained from the three Smooth Residual Blocks (represented as level 0, 1, and 2) in green shades. The first block appears to predict a baseline for the prediction, almost matching the mean of the future records.

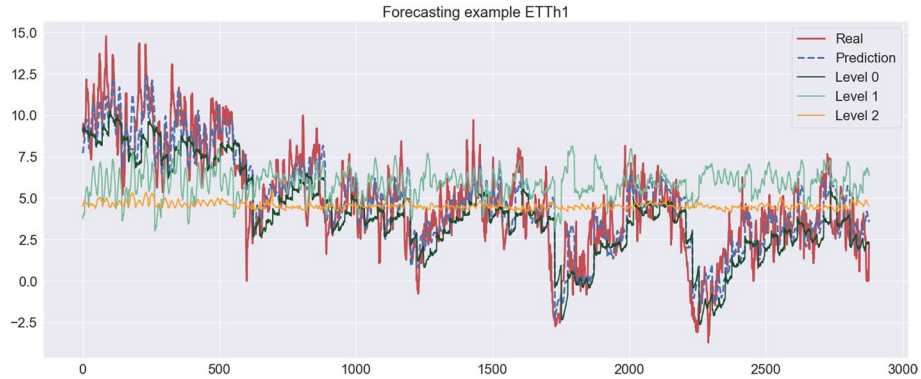


Fig. 7 Output produced by SRCNet for the complete test set of the ETTh1 dataset. The forecast includes the output produced by the SRCNet levels, which summarizes the final forecast

Table 4 Results of Bayesian analysis comparing the efficacy of all reference models with SRCNet

Dataset	Reference	p (reference>SRCNet)	p (Rope)	p (SRCNet>reference)
ETTh1	CNN	1	0	0
	NBEATS	1	0	0
	HLNet	1	0	0
	SCINet	1	0	0
ETTh2	CNN	1	0	0
	NBEATS	1	0	0
	HLNet	1	0	0
	SCINet	1	0	0

The second block focuses on capturing the global shape of the prediction, with a particular emphasis on the ups and spikes, while the last block seems to concentrate on predicting the downs.

In the overall prediction, as observed in Fig. 7, the first block produces an initial estimate of the predictions. The second and third blocks appear to build on this base prediction, focusing on the spikes of the prediction. Specifically, the second block obtains a global representation of the spikes, while the last block appears to capture smaller changes.

Statistical tests

In this section, we apply a statistical test to support our conclusions based on the results obtained during the grid search. We use the Bayesian version of the Wilcoxon signed-rank test proposed by Benavoli et al. [30] due to the well-known issues with p-values in hypothesis testing [30].

For each combination of hyperparameters, the Bayesian test compares each pair of metrics for models A and B, producing three output values. The first value represents the probability that model A is better than model B (p(A>B)), the second value represents the inverse scenario (p(B>A)), and the third value represents the probability that model A and B are similar (p(Rope)). To establish the range of statistical similarity, we

Table 5 Results of Bayesian analysis comparing the efficiency for all reference models with SRCNet

Dataset	Reference	p (reference>SRCNet)	p (Rope)	p (SRCNet>reference)
ETTh1	CNN	0	1	0
	NBEATS	0.31	0.69	0
	HLNet	0.25	0.75	0
	SCINet	1	0	0
ETTh2	CNN	0	1	0
	NBEATS	0	1	0
	HLNet	0.02	0.98	0
	SCINet	1	0	0

need to set a threshold, which will be determined after analyzing the results for each metric.

As the selected method is a non-parametric paired test, we need to obtain each pair of results for each combination of hyperparameters in both datasets. However, each model has a different number of hyperparameters, and some are shared while others are not. Therefore, we only select the shared hyperparameters to obtain the pairs, and then choose the best combination of non-shared hyperparameters.

Table 4 presents the results of the Bayesian test applied to WAPE. The probabilities that the reference model has a greater WAPE are represented by p(reference>SRCNet), while the contrary is represented by p(SRCNet>reference), and the probability of non-significant changes is p(Rope).

The statistical test used a threshold of 0.1% of the relative error to determine the range of statistical similarity. This means that results with an error difference of less than 0.1% are considered similar. The test indicates a 100% probability in all cases, indicating that SRCNet performs better than the reference models and generally achieves lower errors.

Table 5 displays the Bayesian test applied to the efficiency metric. Once again, the probabilities are shown as p(reference>SRCNet) for the probability that the reference model has a longer training time, p(SRCNet>reference) for the opposite scenario, and p(Rope) for the probability of non-significant changes.

In this case, a 60-second threshold was selected, which means that training times with differences of less than 60 seconds are considered similar. The results for CNN, NBEATS, and HLNet indicate that they are generally similar to SRCNet across all datasets. Despite the fact that SCINet has the closest efficacy results to SRCNet, its efficiency is clearly worse in both datasets.

Conclusions and future works

SRCNet has been demonstrated to outperform one of the best methods reported in the literature for this dataset [17] in terms of efficacy, while also improving efficiency. Efficiency was achieved using the approach of decomposing the input window, which differed from conventional trend and seasonality decomposition. This supports the hypothesis that reducing bias in the architecture may improve efficacy, as greater freedom allows for a more optimal data decomposition. The introduction of the Smooth Residual Block also provided the necessary amount of bias to enhance efficacy without

compromising efficiency. In future work, an extended version of the architecture could be explored to assign importance to each block and reduce the impact of noise in the last block, if necessary. Additionally, an investigation into the impact of inductive bias on architectures could be conducted to assess their performance in terms of generalization, data scaling, and efficiency.

Abbreviations

CNN	Convolutional neural network
LSTM	Long short-term memory
HLNet	Hierarchical learning network
MAE	Mean absolute error
SRCNet	Smooth residual connection network
SCINet	Sample convolution and interaction network
ETTh1	Hourly electricity transformer temperature dataset for trasformer 1
ETTh2	Hourly electricity transformer temperature dataset for trasformer 2
OT	Transformer oil temperature

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.

Author contributions
MJJN: conceptualization, methodology, software development, writing. MMB: writing reviewing, statistical test analysis, supervision. FMA: writing reviewing, validation, supervision. GAC: data curation, writing reviewing, supervision. All authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
The dataset has no restrictions that all data can be acquired at the related site.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 20 October 2022 Accepted: 2 May 2023
Published online: 28 May 2023

References

1. Román-Portabales A, López-Nores M, Pazos-Arias JJ. Systematic review of electricity demand forecast using ANN-based machine learning algorithms. Sensors. 2021; 21(13):4544.
2. Martínez-Álvarez F, Troncoso A, Asencio-Cortés G, Riquelme JC. A survey on data mining techniques applied to electricity-related time series forecasting. Energies. 2015; 8(11):13162–93.
3. Thiviyanathan VA, Ker PJ, Leong YS, Abdullah F, Ismail A, Jamaludin ZM. Power transformer insulation system: a review on the reactions, fault detection, challenges and future prospects. Alex Eng J. 2022; 61(10):7697–713.
4. Dursun K. Oil and winding temperature control in power transformers. In: Proceedings of the International Conference Power engineering, energy and electrical drives, 2013; pp. 1631–1639.
5. Beeram S.R, Kuchibhotla S. Time series analysis on univariate and multivariate variables: a comprehensive survey. In: Proceedings of the International Conference on Communication Software and Networks, 2021; pp. 119–126.
6. Tabrizchi H, Razmara J, Mosavi A, Varkonyi-Koczy A.R. Deep learning applications for COVID-19: a brief review. In: Proceedings of the International Conference on Research and education: traditions and innovations, 2022; pp. 117–130.
7. Alzubaidi L, Zhang J, Humaidi AJ, Al-dujaili A, Duan Y, Al-Shamma O, Santamaría J, Fadhel MA, Al-Amidie M, Farhan L. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data, 2021; 8(1):53.
8. Al-Maleh M, Desouki S. Arabic text summarization using deep learning approach. J Big Data. 2021; 7(1):56.

9. Zewdu A, Yitagesu B. Part of speech tagging: a systematic review of deep learning and machine learning approaches. J Big Data. 2022; 9(1):10.
10. Aghdam M, Tabbakh SK, Chabok S, Kheirabadi M. Optimization of air traffic management efficiency based on deep learning enriched by the long short-term memory (LSTM) and extreme learning machine (ELM). J Big Data. 2021; 8(1):54.
11. Shen J, Shafiq M. Short-term stock market price trend prediction using a comprehensive deep learning system. J Big Data. 2020; 7(1):66.
12. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A.N, Kaiser L, Polosukhin I. Attention is all you need. In: Proceedings of the International Conference on Neural Information Processing Systems. 2017; pp. 6000–6010.
13. Zhou H, Zhang S, Peng J, Zhang S, Li J, Xiong H, Zhang W. Informer: beyond efficient transformer for long sequence time-series forecasting. In: Proceedings of the International Conference on Advancement of Artificial Intelligence. 2021; pp. 11106–11115.
14. Makridakis S, Spiliotis E, Assimakopoulos V. M5 accuracy competition: results, findings, and conclusions. Int J Forecast. 2022; pp. 1346–1364.
15. Sui J, Ling X, Xiang X, Zhang G, Zhang X, Transformer oil temperature prediction based on long and short-term memory networks. In: Proceedings of the International Conference on Big Data, 2021; pp. 6029–6031.
16. Oreshkin B.N, Carpov D, Chapados N, Bengio Y. N-beats: Neural basis expansion analysis for interpretable time series forecasting. arXiv. 2019.
17. Liu M, Zeng A, Xu Z, Lai Q, Xu Q. Time Series is a Special Sequence: Forecasting with Sample Convolution and Interaction. arXiv. 2021.
18. Klimek J, Klimek J, Kraśkiewicz W, Topolewski M. Query selector-efficient transformer with sparse attention. Softw Impacts. 2022; 11(1): 100187.
19. Du D, Su B, Wei Z. SCformer: segment correlation transformer for long sequence time series forecasting. In: Proceedings of International Conference on Learning Representations. 2022; pp. 1–12.
20. Žagar J, Mihelič J. Creation of attribute vectors from spectra and time-series data for prediction model development. IPSI Trans Internet Res. 2019; 15(2):1054.
21. Woo G, Liu C, Sahoo D, Kumar A, Hoi S. CoST: Contrastive learning of disentangled seasonal-trend representations for time series forecasting. In: Proceedings of International Conference on Learning Representations. 2022; pp. 1–18.
22. He K, Fan H, Wu Y, Xie S, Girshick R. Momentum contrast for unsupervised visual representation learning. In: Proceedings of International Conference on Computer Vision and Pattern Recognition, 2020; pp. 9726–9735.
23. Wu H, Xu J, Wang J, Long M. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In: Proceedings of International Conference on Neural Information Processing Systems. 2021; pp. 1–12.
24. Kosanić M, Milutinović V. A survey on mathematical aspects of machine learning in geophysics: The cases of weather forecast, wind energy, wave energy, oil and gas exploration. In: Proceedings of the International Conference on Embedded Computing. 2021; pp. 1–6.
25. Goyal A, Bengio Y. Inductive biases for deep learning of higher-level cognition. arXiv. 2020.
26. LI X, Grandvalet Y, Davoine F. Explicit inductive bias for transfer learning with convolutional networks. In: Proceedings of the International Conference on Machine Learning. Machine Learning Research. 2018; pp. 2825–2834.
27. Zeng A, Chen M, Zhang L, Xu Q. Are transformers effective for time series forecasting?. arXiv. 2022.
28. Jiménez-Navarro MJ, Martínez-Álvarez F, Troncoso A, Cortés GA. HLNet: A novel hierarchical deep neural network for time series forecasting. In: Proceedings of International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2021). 2022; pp. 717–727.
29. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997; 9(8):1735–1780.
30. Benavoli A, Corani G, Demsar J, Zaffalon M. Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis. J Mach Learn Res. 2017; 18(77):2653–88.

Publisher's Note