Pattern Recognition 120 (2021) 108148

Contents lists available at ScienceDirect

Pattern Recognition

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

Improving the accuracy of global forecasting models using time series data augmentation

Kasun Bandara[a,*], Hansika Hewamalage[a], Yuan-Hao Liu[a], Yanfei Kang[b], Christoph Bergmeir[a]

[a] Faculty of Information Technology, Monash University, Melbourne, Australia
[b] School of Economics and Management, Beihang University, Beijing 100191, China

ARTICLE INFO

Article history:
Received 6 August 2020
Revised 20 December 2020
Accepted 30 June 2021
Available online 9 July 2021

Keywords:
Time series forecasting
Global forecasting models
Data augmentation
Transfer learning
RNN

ABSTRACT

Forecasting models that are trained across sets of many time series, known as Global Forecasting Models (GFM), have shown recently promising results in forecasting competitions and real-world applications, outperforming many state-of-the-art univariate forecasting techniques. In most cases, GFMs are implemented using deep neural networks, and in particular Recurrent Neural Networks (RNN), which require a sufficient amount of time series to estimate their numerous model parameters. However, many time series databases have only a limited number of time series. In this study, we propose a novel, data augmentation based forecasting framework that is capable of improving the baseline accuracy of the GFM models in less data-abundant settings. We use three time series augmentation techniques: GRATIS, moving block bootstrap (MBB), and dynamic time warping barycentric averaging (DBA) to synthetically generate a collection of time series. The knowledge acquired from these augmented time series is then transferred to the original dataset using two different approaches: the pooled approach and the transfer learning approach. When building GFMs, in the pooled approach, we train a model on the augmented time series alongside the original time series dataset, whereas in the transfer learning approach, we adapt a pre-trained model to the new dataset. In our evaluation on competition and real-world time series datasets, our proposed variants can significantly improve the baseline accuracy of GFM models and outperform state-of-the-art univariate forecasting methods.

1. Introduction

In many industries, such as retail, food, railway, mining, tourism, energy, and cloud-computing, generating accurate forecasts is vital as it provides better grounds for decision-making of organisational short-term, medium-term and long-term goals. Here, industrial application databases often consist of collections of related time series that share key features in common. To generate better forecasts under these circumstances, recently, global forecasting models (GFM) have been introduced as a competitive alternative to the traditional univariate statistical forecasting methods [1], such as exponential smoothing [2] and autoregressive integrated moving average [ARIMA, 3]. Compared to univariate forecasting methods that treat each time series separately and forecast each series in isolation, GFMs are unified forecasting models that are built using all the available time series. And thus the GFMs are able to simultaneously learn the common patterns available across a rich collection of time series, and offer much better scalability to the increasing volumes of time series. In particular, deep learning based GFMs have recently achieved promising results by outperforming many state-of-the-art univariate forecasting methods [4–10].

The GFMs outshine univariate forecasting models, in situations where large quantities of related time series are available from the same domain [1]. Examples include the sales demand of related product assortments in retail, the ride-share services demand in multiple regions, server performance measures in computer centres, household smart meter data, and others. The requirement of having adequate amounts of related time series becomes essential when building accurate GFMs, as the model parameters are estimated jointly using all the available time series. This requirement becomes vital for deep learning based GFMs, as they are inherently data ravenous, and require large numbers of model parameters to be estimated. However, in situations where time series databases are constrained by the amounts of time series available, i.e., small

[*] Corresponding author.
E-mail addresses: Herath.Bandara@monash.edu (K. Bandara), Hansika.Hewamalage@monash.edu (H. Hewamalage), yanfeikang@buaa.edu.cn (Y. Kang), Christoph.Bergmeir@monash.edu (C. Bergmeir).

https://doi.org/10.1016/j.patcog.2021.108148
0031-3203/© 2021 Elsevier Ltd. All rights reserved.

to medium sized datasets, GFMs may not reach their full potential in accuracy.

In the absence of adequate amounts of time series data, GFMs may not be able to learn important characteristics of time series, such as seasonality. In such circumstances, one approach is to supplement the model training procedure by incorporating expert knowledge available about time series. In a situation where such expert knowledge is not readily and explicitly available, data augmentation (DA) techniques can be used to artificially generate new copies of data to increase the sample size in use, and thereby enable the model to learn various aspects of the data better. Here, the DA approach addresses the data sparsity issue by generating synthetic data to increase the number of observations available for model training. The application of the DA strategy has proven successful in various machine learning applications, such as image classification [11], speech recognition [12], text classification [13], general semi-supervised classification [14], and time series related research [15–18]. Another approach to overcome the extensive data requirements of learning algorithms is by transferring the knowledge representations from a background dataset to a target dataset. This process is commonly referred to as transfer learning (TL) in the machine learning literature. In TL, a base model is initially trained using a background or a source dataset that models the source task. The pre-trained model is then transferred to a target dataset with a target task. The TL strategy is particularly useful when the target dataset is significantly smaller than the background dataset. As a result, TL based approaches have been used in a wide range of machine learning applications, such as image classification [19–22], language modelling [23–25], and more recently in imaging based time series forecasting [26]. The success of TL based approaches in these applications can be mainly attributed to the rich data structure inherently available in text, images, and speech related data. As a result, pre-trained models are able to capture the common features of data that can be easily transferable to the target task, while obviating the target task to learn the general characteristics of data from scratch.

When using DA in the time series context, one branch of techniques aims to generate artificial time series that are similar to the data generation process (DGP) of the original dataset [15,16], whereas other techniques generate random sets of time series that may not be similar to the DGP of the original dataset, and they only resemble general characteristics of time series. When building GFMs, the knowledge of the augmented time series can be transferred to a target dataset in two different ways: training a GFM by pooling the augmented time series and the original time series together or pre-training a GFM using the augmented time series, and therewith transferring the knowledge representations of the pre-trained model to the original dataset using the TL strategy. While several studies have investigated the use of these approaches for time series forecasting, a thorough study has not yet been explored in the GFM context. As the recent success of GFMs mostly depends on data-abundant settings [4,5,27], it is crucial to investigate the use of GFMs with limited time series data, and to develop strategies to make GFMs competitive under these circumstances. Motivated by this gap, in this study, we propose a GFM based forecasting framework that can be used to improve the forecast accuracy in data-sparse time series databases. As the primary prediction module of our framework, we use Recurrent Neural Networks (RNN), a promising neural network (NN) architecture that has been heavily used in the recent GFM literature [4–8].

In this study, we demonstrate the use of DA techniques to improve the accuracy of GFMs in data-sparse environments. We use three different DA techniques to synthetically generate time series, namely: 1) GeneRAting TIme Series with diverse and controllable characteristics [GRATIS, 18], 2) moving block bootstrap [MBB, 16], and 3) dynamic time warping barycentric averaging [DBA, 15]. GRATIS [18] is a statistical generative model that artificially generates time series with diverse characteristics, which are not necessarily similar to the DGP of the original dataset. Whereas, the MBB and the DBA methods are aimed to generate time series that are similar to the DGP of the original dataset. As described earlier, we transfer the knowledge representation of the augmented time series to the original dataset using two different approaches. In the first approach, we pool the synthetically generated time series together with the original dataset, and build a GFM across all the available time series (pooled strategy). In our second approach, we pre-train a GFM using the augmented time series, and thereafter transfer the knowledge representations of the pre-trained models to the original dataset using the TL methodology (transfer strategy). Here, we use different TL schemes to import the information from the augmented data to a target dataset. Based on the above strategies, we propose a set of model variants to evaluate against the baseline model, which is built using the original set of time series. Furthermore, we use state-of-the-art statistical forecasting techniques as benchmarks to compare against our proposed methods. The proposed forecasting framework is attested using five time series databases, including two competition datasets and three real-world datasets. As real-world time series are non-negative in many applications, the proposed framework in its current form focuses on non-negative datasets. However, we note that this assumption can be removed with minimal modifications to our framework.

The rest of the paper is organised as follows. In Section 2, we provide a brief review on TL techniques and their recent applications to time series forecasting, and an overview of time series DA approaches. In Section 3, we discuss the architecture of our forecasting engine in detail. The proposed TL schemes are described in Section 4. In Section 5, we explain the time series augmentation techniques used in this study. Our experimental setup is discussed in Section 6. Finally, we conclude our paper in Section 7.

2. Related work

In the following, we discuss the related work in the areas of DA and TL for time series analysis.

2.1. Time series augmentation

The application of DA techniques in machine learning models has seen success in many domains[28]. use a Bayesian generative adversarial network to generate new samples of wind and solar energy input data with different variations. Whereas, [29] use recurrent generative adversarial networks to generate synthetic clinical records in the medical domain, where ethical restrictions often constrain the data collection. Similar to TL, the successful application of DA can be seen in image classification [11], speech recognition [30], and text classification [13].

There exists a large body of literature that discusses various forms of DA techniques available for time series analysis. This includes time series bootstrapping methods [16,31], time series averaging techniques [15], and statistical generative models [16,18,32–34]. use the MBB method to generate multiple versions of a given time series, and build an ensemble of exponential smoothing forecasting models on the augmented time series[15]. use DBA, a data augmentation strategy that averages a set of time series to produce new samples of data for time series classification. Markov Chain Monte Carlo (MCMC) techniques are also frequently used in the literature to generate synthetic time series [32,33]. More recently, [18] propose GRATIS that uses mixture autoregressive models to generate time series with diverse and controllable characteristics.

On the other hand, recent studies also investigate the use of NNs as a time series augmentation technique [35,36]. generate a set of ambient temperature hourly time series using a Multi-Layer Perceptron architecture, and demonstrate the effectiveness of using NNs to generate time series closer to the real-world data. Furthermore, [36] employ a deep Convolutional NN architecture to generate synthetic data for time series classification. More recently, deep NN based Generative Adversarial Network (GAN) architectures [37] have received significant attention in the area of DA. The competition-driven training mechanism employed in GANs, allows the network to generate realistic samples, similar to the DGP of the source dataset. In the literature, though GAN architectures are used mostly for image generation, more recent studies have shown that GANs can also be applied to generate new copies of time series [38,39].

2.2. Transfer learning

The existing TL methods can be distinguished by the type of knowledge representation to be transferred and how these representations are transferred. In the following, T_s denotes the source or background task, D_s the corresponding dataset, T_t the target task, and D_t the corresponding dataset. The TL approaches can be mainly categorised into three types, based on the different transfer methods between T_s and T_t [40], namely: Inductive TL, Transductive TL and Unsupervised TL. The Inductive TL is typically used when T_t is different from T_s, irrelevant of their domains. The Transductive TL is applied when T_t and T_s are the same, while their respective domains are different. According to [40], Transductive TL can be further categorised based on the similarities between the feature spaces of D_s and D_t. Finally, the unsupervised transfer learning is used when the labelled data are not available in D_s and D_t for model training. Furthermore, these approaches can be used to transfer various forms of knowledge representations available in D_s. For example, the instance-transfer approach aims to reuse certain parts of D_s for the learning tasks of T_t by applying instance reweighting. The feature-transfer approach attempts to transfer the knowledge of D_s in the form of feature representation, whereas in the parameter-transfer approach, knowledge is represented by the model parameters or prior distributions that may be shared between the T_s and T_t. The relational knowledge-based transfer is expected to exploit similar relationships among D_s and D_t. For further discussions and definitions of these TL paradigms, we refer to [40].

To overcome the data ravenousness of modern-day deep learning algorithms, the TF based approaches have been introduced in many domains, such as image classification [19,41,42] and language modelling [23–25,43]. With respect to image classification, [42] investigate the preliminary results of using transfer learning on images, and [19] explore the use of shared parameters between the source and target domains to improve the accuracy in image classification tasks. Those authors also argue that the initial layers in an NN model tend to capture the general features of an image, while the last layers aim to embed more specific features. In terms of language modelling, [25] propose a feature-transfer approach that uses a stacked denoising autoencoder to learn the invariant representation between the source and target domains. It allows the sentiment classifiers to be trained and deployed on different domains. Furthermore, [24] compare various TL schemes available for personalised language modelling using RNNs. In the TL process, those authors control the number of trainable parameters of the target model by freezing the initial layers of the RNNs. Also, [23] use variational RNNs to capture underlying temporal latent dependencies in language models, whereas [43] implement a parameter-transfer approach to use pre-trained weights of the base model to initialise the target model.

More recently, the application of TL methods is also gaining popularity in time series forecasting research[44]. introduce Hephaestus, a TL based forecasting framework for cross-building energy prediction, to improve the accuracy of energy estimations for new buildings with limited historical data. There, those authors propose a seasonal and trend adjusted approach that allows Hephaestus to transfer knowledge across similar buildings with different seasonal and trend profiles. The research work in [45] proposes a loss function to reconstruct the input data of the model, and thereby extract time series features using a stack of fully connected LSTM layers. Those authors show that this feature-transfer approach leads to significant accuracy improvements over the traditional TL approaches, in situations where the size of the target dataset is small. To handle time-varying properties in time series data, [46] propose a hybrid algorithm, based on TL that effectively accounts for the observations in the distant past, and leverages the latent knowledge embedded in past data to improve the forecast accuracy. Moreover, [47] implement an RNN autoencoder architecture to extract generic sets of features from multiple clinical time series databases. Those features are then used to build simple linear models on limited labelled data for multivariate clinical time series analysis. Li et al. [26] first transform time series into images and use TL for image feature extraction. The extracted features are used as time series features to obtain the optimal weights of forecast combination [18].

In summary, we have identified feature-transfer learning and parameter-transfer learning approaches as the most commonly used TL paradigms in deep learning based applications. It can be mainly attributed to the capability of NNs to extract non-trivial latent representations of data.

3. Forecasting framework

In this section, we describe in detail the main components of our proposed GFM based forecasting framework, and an illustration gives Fig. 2. The framework consists of three layers, namely: 1) the pre-processing layer, 2) the RNN training layer, and 3) the post-processing layer. In the following, we first discuss the preprocessing techniques used in our forecasting framework. Then, we provide a brief introduction to residual RNNs, which are the primary prediction unit of our forecasting engine. Finally, we explain the functionality of the post-processing layer of the framework.

3.1. Time series pre-processing

As GFM methods are trained across a group of time series, accounting for various scales and variances present in these time series becomes necessary [6,7]. Therefore, as the first step in our pre-processing pipeline, we normalise the collection of time series $\mathcal{X} = \{X_i\}_{i=1}^N$ using a *meanscale* transformation strategy [5,7], which can be defined as follows:

$$X_{i,\text{normalised}} = \frac{X_i}{\frac{1}{k}\sum_{t=1}^{k} X_{i,t}}, \tag{1}$$

where $X_{i,\text{normalised}}$ represents the ith normalised time series, and k is the number of observations in time series X_i, where $i \in \{1, 2, \cdots, N\}$.

We then stabilise the time series' variance by log transformation. It also allows us to convert possible multiplicative seasonal and trend components of a given time series into additive ones, which is necessary for the last step in our pre-processing pipeline, the deseasonalisation process. To avoid problems for zero values, we use the log transformation in the following way:

$$X_{i,\text{normalised \& logscaled}} = \begin{cases} \log(X_{i,\text{normalised}}), & min(\mathcal{X}) > 0; \\ \log(X_{i,\text{normalised}} + 1), & min(\mathcal{X}) = 0, \end{cases} \tag{2}$$

where $\mathcal{X}$ denotes the full set of time series, and $X_{i,\text{normalised \& logscaled}}$ is the corresponding normalised and log transformed time series of X_i. Though our framework currently restricts time series to be non-negative, we note that this constraint can be easily obviated by excluding the above log transformation procedure from the pre-processing phase.

As the next step of our pre-processing pipeline, we introduce a time series deseasonalisation phase to extract the seasonal components from time series. Following [8], when using NNs for forecasting, these extracted seasonal components can be used in two different ways. In the first training paradigm, those authors suggest the Deseasonalised (DS) approach, which removes the extracted seasonal values from a time series, and then uses the remainder, i.e., trend and residual components, to train the NN. Here, as the seasonal components are removed from the time series, an additional reseasonalisation step is introduced in the post-processing phase to predict the future seasonal values of the time series. In the second training paradigm, Seasonal Exogenous (SE) approach, the extracted seasonal components are used as exogenous inputs in addition to the original observations of the time series. As the time series are not seasonally adjusted in this approach, an additional reseasonalisation step is not required in the post-processing phase. The main objective of these two training paradigms is to supplement the subsequent NNs learning process. Those authors suggest that the accuracy of these two variants depends on the seasonal characteristics of the time series. In line with the recommendations by [8], we use these two approaches accordingly in our experiments. In Section 6, we summarise the training paradigms used for each dataset.

Following the recent success of Seasonal-Trend Decomposition [STL, 48] as a pre-processing technique for NNs [6–8], we use it to extract the seasonal components from time series. When we apply STL to a normalised and log scaled time series $X_{i,\text{normalised \& logscaled}}$, its additive decomposition can be formulated as follows:

$$X_{i,\text{normalised \& logscaled}} = \hat{S}_i + \hat{T}_i + \hat{R}_i, \tag{3}$$

where $\hat{S}_i$, $\hat{T}_i$, $\hat{R}_i$ are the corresponding seasonal, trend, and the residual components of the time series $X_{i,\text{normalised \& logscaled}}$, respectively. In this study, we use the R [49] implementation of the STL algorithm, `stl`, from the `forecast` package [50,51].

3.2. Residual recurrent neural networks

Nowadays, the application of deep learning models is gaining popularity among the time series forecasting community [4,8]. Many of these innovations are based on RNN architectures, which were motivated mainly by the continued success of RNNs in modelling sequence related tasks [52,53]. A host of different RNN architectures for time series forecasting exists in the forecasting literature, overviews and discussions [7,54]. Based on the recommendations given by [7], when forecasting with GFMs, we select the Long Short-Term Memory network (LSTM) as our primary RNN architecture and implement the *Stacking Layers* design pattern to train the network. Furthermore, we introduce residual connections to the stacking architecture to address the vanishing gradient problem that may occur in situations with a higher number of hidden layers [55]. This was originally proposed by [56] as the *Residual Net (ResNet)*, where the authors use residual connections to accommodate substantially deeper architectures of Convolutional NNs (CNN) for image classification tasks. They also argue that learning the residual mappings is computationally easier than directly learning to fit the underlying mapping between input and output. More recently, a variant of the ResNet architecture has been applied to time series forecasting using RNNs [4,57]. In fact, the residual architecture proposed by [4] became the winning solution of the M4 forecasting competition [58]. During the transfer learning phase (see Section 4), we expect to add extra stacking layers to the base architecture. Therefore, having residual connections among the stacking layers becomes necessary for a stable learning process in our network.

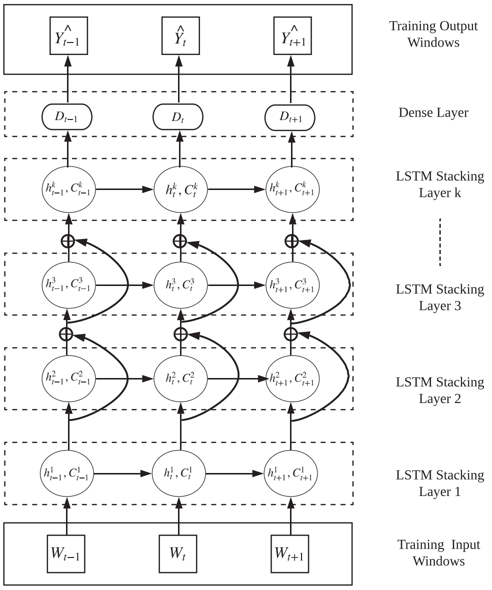


Fig. 1. The unrolled representation of a residual recurrent network architecture with an amount of k stacking layers. Here, the residual connections are represented by curved arrows. According to [56], these residual connections allow the stacking layers to fit a residual mapping between W_t and $\hat{Y}_t$, while avoiding the network to degrade with increasing network depth.

Fig. 1 illustrates the architecture of the forecasting engine used in our experiments, which mainly consists of three components: an input layer, stacking layers with a dense layer, and an output layer. To train this network, we use the pre-processed time series in the form of input and output training windows. Here, the values of the pre-processed time series depend on the training paradigm, i.e., the DS or SE strategy, used in a particular dataset. These training windows are generated by applying the Moving Window (MW) transformation strategy to pre-processed time series. Existing studies often recommend the MW strategy, when training NNs for time series forecasting [4,6,7]. This is mainly due to the Multi-Input Multi-Output (MIMO) principle used in this strategy, where the size of the training output window, m is identical to the size of the intended forecasting horizon M. In this way, the network is trained to directly predict the entire forecasting horizon X_i^M at once, avoiding prediction error accumulation at each forecasting step [59]. Furthermore, on these training windows, we use the local normalisation strategy suggested by [6,7] to avoid possible network saturation effects that occur in NNs. Here, the local normalisation strategy used for the DS approach differs from the SE approach. In the DS approach, we use the trend component of the input window's last value, while the mean value of each input window is used in the SE approach. In Fig. 1, $W_t \in \epsilon R^n$ represents the teacher input window at time step t, whereas $\hat{Y}_t \in \epsilon R^m$ represents the LSTM output at time step t. Here, n denotes the size of the input window. Moreover, h_t refers to the hidden state of LSTM at time step t, while its cell memory at time step t is given by C_t. A fully connected layer D_t (excluding the bias component) is introduced to map each LSTM cell output h_t to the dimension of the output window m, equivalent to M. Given the length of the time series X_i as p, we use an amount of $(p - m)$ data points from the pre-processed X_i to train our network and reserve the last output window of the pre-processed X_i for the network validation. The L1-norm is used as the primary learning objective function of our training archi-

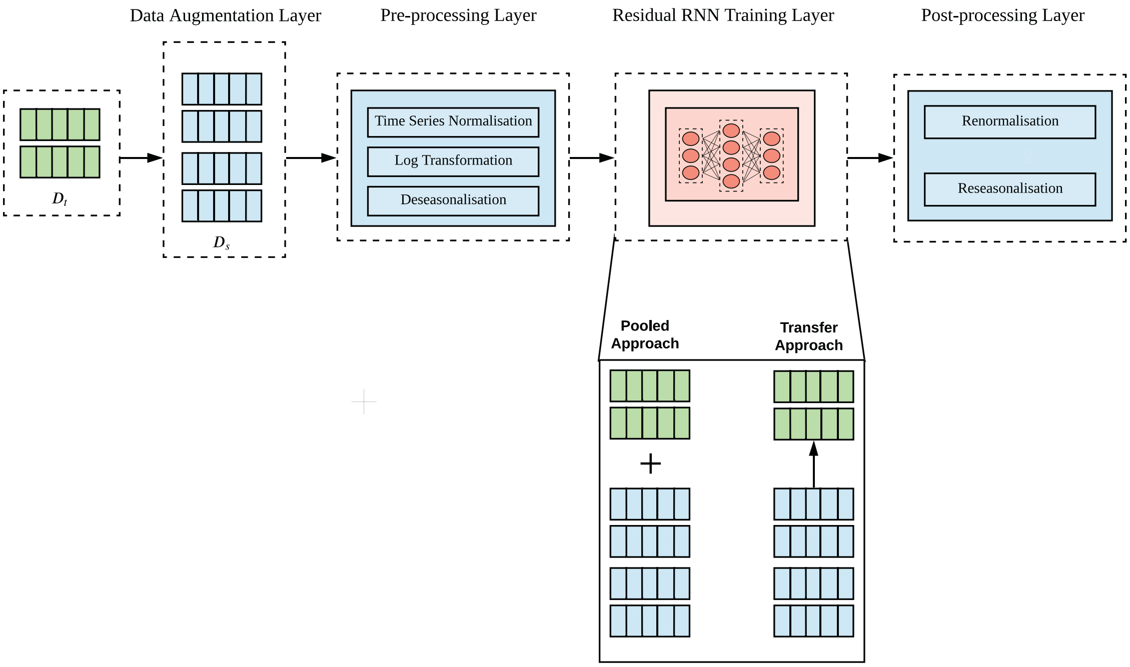


Fig. 2. An overview of the proposed framework, which includes a data augmentation layer, pre-processing layer, residual RNN training layer, and a post-processing layer. Here, D_t denotes the target time series to forecast, and D_s refers to the augmented time series. We generate D_s using the techniques introduced in Section 5. We import the knowledge to D_t from D_s, using two different approaches, namely: 1) pooled approach and 2) transfer learning approach. In the pooled approach, we train D_t alongside D_s. Whereas in the transfer learning approach, we pre-train a model using D_s, and thereafter transfer the learned knowledge to D_t using the TL architectures proposed in Section 4.

tecture, along with an L2-regularisation term to minimise possible overfitting in the network.

Even though we use the LSTM as the primary RNN cell in this study, we note that our forecasting engine can be used with any other RNN variant such as Elman RNN [60], Gated Recurrent Units [GRU, 61], and others.

3.3. Post-processing

We compute the final predictions of our forecasting framework by applying a reseasonalisation process and a denormalisation process to the output given by the LSTM. As outlined earlier in Section 3.1, the reseasonalisation step is only required when the network is trained using the DS strategy. Here, the reseasonalisation process involves adding back the seasonal components, which have been removed during the pre-processing phase. The relevant seasonal components of the time series are computed by repeating the last seasonal components of the time series into the future, up to the intended forecast horizon. For example, if the intended forecast horizon is 8 steps ahead, the seasonal components relevant to those observations are added to the predictions generated by the Residual RNN.

4. Transfer learning architectures

As discussed in Section 2, [24] suggest a host of different TL schemes, when using RNNs for personalised language modelling. In line with these recommendations, we investigate the use of three transfer learning schemes for time series forecasting with the LSTM stacking architecture. Fig. 3 shows the different TL schemes used in this study. Here, we use an abstract view of the proposed three layered residual recurrent architecture (see Fig. 1) to simplify the illustration of the proposed TL schemes. In summary, TL.Dense introduces a dense layer to the pre-trained base model, mapping the output of the base model to the dimension of the output window in D_t. TL.AddDense adds an amount of q dense layers to the pre-trained base model. Finally, in TL.LSTM, an amount of q LSTM residual layers with a dense layer is introduced to the base model. In these TL schemes, we assume there exist k residual layers in the base model. We further introduce variants to these TL schemes by changing the total number of trainable parameters in the architecture. Here, we achieve this by training the network, only using the parameters of newly introduced hidden layers, while freezing the hidden layers of the pre-trained base model. Based on these TL schemes, we define the proposed TL architectures as follows:

TL.Dense.Freeze: The TL architecture that uses the TL.Dense scheme, while freezing the initial layers of the pre-trained model, and training only the newly added layers.

TL.Dense.Retrain: The TL architecture that uses the TL.Dense scheme, while re-training initial layers of the pre-trained model and newly added layers.

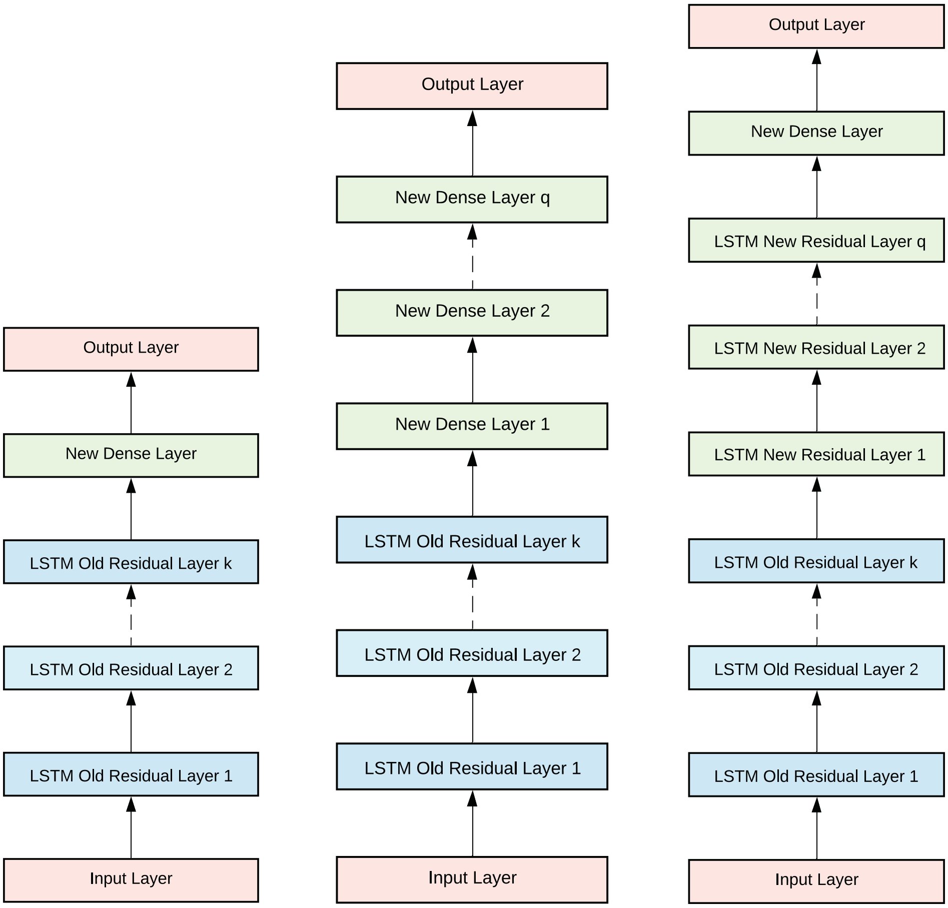
TL.AddDense.Freeze: The TL architecture that uses the TL.AddDense scheme, while freezing the initial layers of the pre-trained model, and training only the newly added layers.

TL.AddDense.Retrain: The TL architecture that uses the TL.AddDense scheme, while re-training initial layers of the pre-trained model and newly added layers.

TL.LSTM.Freeze: The TL architecture that uses the TL.LSTM scheme, while freezing the initial layers of the pre-trained model, and training only the newly added layers.

TL.LSTM.Retrain: The TL architecture that uses the TL.LSTM scheme, while re-training initial layers of the pre-trained model and newly added layers.

Here, TL.Dense.Retrain, TL.AddDense.Retrain, and TL.LSTM.Retrain re-train all the layers of the model, whereas


Output Layer
New Dense Layer
LSTM Old Residual Layer k
LSTM Old Residual Layer 2
LSTM Old Residual Layer 1
Input Layer
Output Layer
New Dense Layer q
New Dense Layer 2
New Dense Layer 1
LSTM Old Residual Layer k
LSTM Old Residual Layer 2
LSTM Old Residual Layer 1
Input Layer
Output Layer
New Dense Layer
LSTM New Residual Layer q
LSTM New Residual Layer 2
LSTM New Residual Layer 1
LSTM Old Residual Layer k
LSTM Old Residual Layer 2
LSTM Old Residual Layer 1
Input Layer

(a) TL.Dense (b) TL.AddDense (c) TL.LSTM

Fig. 3. An overview of the proposed TL schemes used in this study. The layers used to build the base model using the D_s, are represented in blue colour, while the additional layers introduced when building the target model using the D_t, are represented in green colour. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

the TL.Dense.Freeze, TL.AddDense.Freeze, and TL.LSTM.Freeze re-train only newly added layers to the model.

Also, according to the definitions of [40], our approach follows the Transductive TL approach (see Section 2.2). This is because, the T_t and T_s are the same, i.e., time series forecasting, while the feature spaces between the D_t and D_s can be similar or different. In situations where the augmented time series follow a similar DGP to the target dataset, i.e., MBB and DBB, we assume the feature spaces between the D_t and D_s are similar. Whereas, if the augmented time series are generated from an arbitrary DGP, i.e., GRATIS, we assume that the feature spaces between the D_t and D_s are different. We implement the above TL learning schemes and the residual RNN architecture proposed in Section 3.2 using TensorFlow, an open-source deep learning toolkit [62].

5. Time series augmentation

As highlighted in Section 1, DA techniques are useful for synthetically increasing the number of training samples in a dataset. In this study, we use several time series based DA techniques to artificially generate time series, namely GRATIS, MBB, and DBA. The MBB [16] and DBA [15] methods are expected to generate time series from a similar DGP with that of the original series, whereas the GRATIS method generates time series with diverse characteristics from different DGPs, not related to the original dataset. In the following, we briefly describe the above methods, and explain how we exactly use them in our experiments.

5.1. GRATIS

We use GRATIS, a statistical generative model proposed by [18] to create new time series with diverse characteristics. GRATIS employs mixture autoregressive (MAR) models to generate a new set of time series with diverse features. In statistical modelling, MAR models are commonly used to model populations with multiple statistical distributions and diverse characteristics, by using mixtures of models instead of a single autoregressive (AR) model. In the mixture of AR models, each AR process's coefficients are selected from a Gaussian distribution. Then, a mixture weight matrix provides the contribution of each AR model to the generated time series. For more detailed discussions of the GRATIS time series generation methodology, we refer to [18]. In our experiments, we use the implementation available in the `generate_ts` function from the R package `gratis` [63].

In our experiments, we use GRATIS with the second approach (transfer strategy), which pre-trains a GFM model using the time series generated from GRATIS, and then transfers the knowledge to the target dataset. This is because the accuracy of GFMs can degenerate if the two joined time series datasets are too different from each other [6]. Therefore, the augmented time series from the GRATIS method are presumably not suitable to be used with our first approach, the pooled strategy, which trains the original set of time series alongside the augmented time series. Using the second approach, even when using GRATIS, we can still transfer generic time series information from the augmented dataset.

5.2. Moving block bootstrapping

The MBB is a commonly used bootstrapping technique in time series forecasting [16,64]. In our experiments, we use the MBB technique for time series augmentation, following the procedure introduced in [16]. To generate multiple copies of a time series, they first use STL to extract and subsequently remove seasonal and trend components of a time series. Next, the MBB technique is applied to the remainder of the time series, i.e., seasonally and trend adjusted series, to generate multiple versions of the residual components. For more detailed discussions of the MBB technique, we refer to [16]. Finally, the bootstrapped residual components are added back together with the corresponding seasonal and trend to produce new bootstrapped versions of a time series.

As original observations are used in the bootstrapping process, the artificially generated data closely resemble the distribution of the original training dataset, i.e., with similar seasonality and trend. We use the MBB implementation available in the `bld.mbb.bootstrap` function from the R package `forecast` [50].

5.3. Dynamic time warping barycentric averaging (DBA)

Another procedure we use in our work is the Dynamic Time Warping (DTW) based time series augmentation technique proposed by [15]. In contrast to MBB that applies the bootstrapping procedure to each time series separately, the DBA approach averages a set of time series to generate new synthetic samples, thus being able to mix characteristics of different time series when generating new series, and therewith better accounting for the global characteristics in a group of time series. The DBA approach allows weighted averaging in the model when calculating the contribution of each time series towards the final generated time series[15]. develop three methods to determine the weights associated with the time series of the dataset, namely: Average All (AA), Average Selected (AS), and Average Selected with Distance (ASD). For detailed discussions of the theoretical foundations and the methods, we refer to [15]. Having been specifically evaluated against time series classification tasks, DBA is used in this study in a time series forecasting setting. Following the competitive results shown in [15], we use ASD as our primary averaging method. In particular, we use an implementation of the ASD method in Python from [65]. As characteristics of the original dataset are used to generate time series, similar to MBB, DBA can also be classified as a DA technique that generates augmented series similar to the original training dataset.

6. Experimental study

In this section, we evaluate the proposed variants of our framework on five time series datasets. First we describe the datasets, error metrics, statistical tests, hyper-parameter selection method, and benchmarks used in our experimental setup. Then, we provide a detailed analysis of the results obtained.

6.1. Datasets

We use five benchmark time series datasets, which are composed of real-world applications and data from forecasting competitions. To limit the number of observations available and create a situation of relative data-scarcity, which is the main focus of our paper, for datasets with higher sampling rate, i.e., sub-hourly, hourly, and daily, we aggregate the series to weekly time series. We also note that all benchmark datasets contain non-negative observations, which in our experience is representative for most real-world situations in many application areas. We briefly describe the five datasets as follows.

Table 1
Summary of the used datasets.

Dataset	N	$K_{\min}$	$K_{\max}$	T	S	M
NN5	111	105	105	weekly	52	8
NN3	111	51	126	monthly	12	18
AusEnergy-Demand	5	313	313	weekly	52	52
AusGrid-Energy	299	132	132	weekly	52	24
Electricity	321	146	146	weekly	52	10

- NN5 Dataset [66]: Daily dataset from the NN5 forecasting competition, containing daily cash withdrawals at various automatic teller machines (ATMs) located in the UK. We aggregated the original daily time series to weekly time series.
- NN3 Dataset [67]: Monthly dataset from the NN3 forecasting competition.
- AusEnergy-Demand Dataset [68]: The energy demand of different states in Australia. The original time series has a data point every 15 minutes. We aggregate it to weekly energy demand.
- AusGrid-Energy Dataset [69]: A collection of half-hourly time series, representing the energy consumption of households in Australia. We aggregate the original half-hourly time series to weekly time series.
- Electricity Dataset [70]: Electricity consumption records, sampled every 15 minutes from multiple households in Portugal. We aggregate the original data to reflect weekly electricity consumption.

Table 1 summarises statistics of the datasets used in our experiments. Here, N denotes the number of time series, $K_{\min}$ and $K_{\max}$ denote the minimum and maximum available lengths of the time series, respectively, T denotes the sampling rate of the time series, S represents the seasonality present in the time series, and M is the intended forecast horizon. In the NN3 dataset, we see that the lengths of the time series vary considerably, whereas other datasets contain time series with equal lengths ($K_{\min}$ and $K_{\max}$ are the same). Except for the NN3 dataset, we choose the size of the input window n equivalent to $M * 1.25$, following the heuristic proposed by Bandara et al. [6] and Hewamalage et al. [7]. We use $n = 11$ for the NN3 dataset due to the short lengths of its time series.

Furthermore, we choose the SE and DS training paradigms (see Section 3) based on the recommendations of [8]. We use the DS approach for the NN3 dataset, as the time series of those datasets are from disparate data sources, and have unknown starting dates. The SE approach is used for the remaining datasets, which are comprised of homogeneous time series with aligned time stamps.

6.2. Performance measures

To measure the performance of the proposed framework and benchmarks, we use two scale-independent evaluation metrics commonly found in the forecasting literature [71], namely the symmetric Mean Absolute Percentage Error (sMAPE) and the Mean Absolute Scaled Error (MASE). The sMAPE is defined as follows:

$$\text{sMAPE} = \frac{2}{m} \sum_{t=1}^{m} \left(\frac{|F_t - A_t|}{|F_t| + |A_t|} \right), \tag{4}$$

where A_t represents the observation at time t, F_t is the generated forecast, and m indicates the forecast horizon. As the

Table 2
The hyper-parameter ranges used in our experiments.

Model Parameter	Minimum value	Maximum value
LSTM cell dimension	20	50
Mini-batch size	1	100
Epoch size	2	5
Maximum epochs	2	50
Hidden layers	1	5
Gaussian noise injection	10^{-4}	8×10^{-4}
Random-normal initialiser	10^{-4}	8×10^{-4}
L2-regularisation weight	10^{-4}	8×10^{-4}

sMAPE can be unstable around zero values [71], we use the modification proposed by [72] for datasets that have values close to zero, namely the Electricity dataset in our benchmark suite. In this case, we modify the denominator of Equation (4) to $\max(|F_t| + |A_t| + \epsilon, 0.5 + \epsilon)$, where ϵ is a small constant that we set to 0.1, following the recommendations of [72].

Moreover, we use the MASE, a less skewed and more interpretable error measure compared with sMAPE [71]. The MASE error measure is defined as follows:

$$\text{MASE} = \frac{\frac{1}{m}\sum_{t=1}^{m} |F_t - A_t|}{\frac{1}{n-S}\sum_{t=S+1}^{n} |A_t - A_{t-S}|}. \tag{5}$$

Additionally, in Equation (5), n is the number of observations in the training set of a time series, and S refers to the length of the seasonal period in a given time series. The model evaluation of this study is presented using these error measures per series. We then calculate average ranks across all series from the benchmark suite, and also calculate Mean sMAPE, Median sMAPE, Mean MASE, and Median MASE across series within a dataset, to provide a broader overview of the error distributions.

6.3. Statistical tests of the results

We use the non-parametric Friedman rank-sum test to assess the statistically significance of differences among the compared forecasting methods on the benchmark datasets [73][1]. Also, Hochbergs post-hoc procedure is used to further examine these differences with respect to the best performing technique. The statistical testing is done using the error measures specified in Section 6.2, with a significance level of $\alpha = 0.05$.

6.4. Hyper-parameter tuning and data augmentation

The base learner of our forecast engine, LSTM, has various hyper-parameters, including LSTM cell dimension, number of epochs, hidden-layers, mini-batch size, and model regularisation terms. To autonomously determine the optimal values of these hyper-parameters, we use the sequential model-based algorithm configuration (SMAC), a variant of Bayesian Optimisation proposed by [74]. In our experiments, we use the Python implementation of SMAC, which is available as a Python package [75]. To minimise the overall amount of hyper-parameters to be tuned in the learning phase, as the primary learning algorithm, we use COntinuous COin Betting (COCOB) proposed by [76]. Unlike in other gradient-based optimisation algorithms, such as Adam and Adagrad, COCOB does not require tuning of the learning rate. Instead, it attempts to minimise the loss function by self-tuning its learning rate. In this way, we remove the need for fine-tuning the learning rate of our optimisation algorithm. In our experiments, we use the Tensorflow implementation of COCOB [77]. Table 2 summarises the ranges of hyper-parameter values explored in our experiments.

The parameter uncertainty of our proposed models is addressed by training all the models on ten different Tensorflow graph seeds and taking the median of the forecasts generated with those seeds. When generating synthetic time series from the GRATIS, MBB, and DBA approaches (see Section 5), we use three different seeds to address the stochastic nature of these methods. We apply the proposed transfer and pooling strategies to each set of synthetic time series generated from those seeds and compute the average error across them. For each dataset, we generate an equal number of artificial time series from the proposed DA techniques. The number of generated time series (n_A) for each dataset is determined by the amount of bootstraps used per series in the MBB technique. In MBB, we use ten bootstraps per time series, except for the AusEnergy-Demand dataset, where we use 200 bootstraps per time series due to the small size of the dataset. For example, the n_A values of the NN5, NN3, AusEnergy-Demand, AusGrid-Energy, and Electricity datasets are 1110, 1110, 1000, 2990, and 3210, respectively. Furthermore, when running the GRATIS method using the `generate_ts` function, we set the *frequency* parameter equal to the length of the seasonal period (S) of the time series. The *nComp* parameter, which determines the number of mixing components in the MAR model, is set to 4, and n is set to the maximum length of a series in the dataset ($K_{\max}$).

6.5. Benchmarks and variants

We use a host of univariate forecasting techniques to benchmark against our proposed GFM variants, including a forecasting method from the exponential smoothing family, namely the method ES as implemented in the `smooth` package in R by [78], and an ARIMA model from the `forecast` package implemented in R [50]. As our exponential smoothing benchmark, we choose the ES implementation over the more popular ETS method from the `forecast` package [2], as ES is not restricted by the number of seasonal coefficients to be included in the model. In addition to these well-established benchmarks from the time series forecasting literature, we use Prophet, a forecasting technique introduced by [79], in its implementation in the R package `prophet`.

Based on the different DA methods and TA architectures introduced in Sections 4 and 5, we define the following variants of our proposed framework.

LSTM.Baseline: The baseline LSTM model that only uses the original set of time series to train a GFM.

MBB.Pooled: The LSTM model that uses the original set of time series pooled together with the synthetic time series generated from the MBB method to train a GFM.

DBA.Pooled: The LSTM model that uses the original set of time series pooled together with the synthetic time series generated from the DBA method to train a GFM.

MBB.TL.K: The LSTM model that transfers the knowledge using the TL architecture K, from a pre-trained LSTM model, which is trained across the time series generated from the MBB method.

DBA.TL.K: The LSTM model that transfers the knowledge using the TL architecture K, from a pre-trained LSTM model, which is trained across the time series generated from the DBA method.

GRATIS.TL.K: The LSTM model that transfers the knowledge using the TL architecture K, from a pre-trained LSTM model, which is trained across the time series generated from the GRATIS method.

[1] More information can be found on the thematic web site of SCI2S about *Statistical Inference in Computational Intelligence and Data Mining* http://sci2s.ugr.es/sicidm

Table 3
The average ranking of each method across all the time series in the benchmark datasets, ordered by the first column, which is sMAPE. For each column, the results of the best performing method(s) are marked in boldface.

Method	Rank sMAPE	Rank MASE
DBA.TL.Dense.Freeze	**10.60**	**10.63**
MBB.TL.Dense.Retrain	10.94	10.95
DBA.TL.LSTM.Freeze	11.09	11.14
DBA.TL.Dense.Retrain	11.17	11.18
DBA.TL.AddDense.Freeze	11.23	11.24
DBA.Pooled	11.39	11.51
DBA.TL.AddDense.Retrain	11.52	11.58
GRATIS.TL.AddDense.Retrain	11.55	11.49
GRATIS.TL.Dense.Retrain	11.64	11.59
GRATIS.TL.LSTM.Retrain	11.64	11.66
LSTM.Baseline	11.74	11.69
MBB.TL.LSTM.Retrain	11.87	11.88
MBB.Pooled	11.99	11.95
DBA.TL.LSTM.Retrain	12.08	12.15
GRATIS.TL.LSTM.Freeze	12.16	12.19
MBB.TL.AddDense.Retrain	12.37	12.35
MBB.TL.LSTM.Freeze	13.05	13.06
MBB.TL.Dense.Freeze	13.15	13.14
MBB.TL.AddDense.Freeze	13.19	13.10
ARIMA	14.22	14.20
Prophet	14.65	14.49
GRATIS.TL.Dense.Freeze	14.95	15.00
GRATIS.TL.AddDense.Freeze	14.99	14.99
ES	16.79	16.82

Table 4
Results of statistical testing for the sMAPE error measure across all the datasets. Adjusted p-values calculated from the Friedman test with Hochbergs post-hoc procedure are shown. A horizontal line is used to separate the methods that perform significantly worse than the control method from the ones that do not.

Method	p_{Hoch}
DBA.TL.Dense.Freeze	-
MBB.TL.Dense.Retrain	0.334
DBA.TL.LSTM.Freeze	0.317
DBA.TL.Dense.Retrain	0.300
DBA.TL.AddDense.Freeze	0.272
DBA.Pooled	0.110
DBA.TL.AddDense.Retrain	0.047
GRATIS.TL.AddDense.Retrain	0.042
GRATIS.TL.Dense.Retrain	0.020
GRATIS.TL.LSTM.Retrain	0.020
LSTM.Baseline	0.009
MBB.TL.LSTM.Retrain	0.002
MBB.Pooled	6.29×10^{-4}
DBA.TL.LSTM.Retrain	2.18×10^{-4}
GRATIS.TL.LSTM.Freeze	8.17×10^{-5}
MBB.TL.AddDense.Retrain	4.04×10^{-6}
MBB.TL.LSTM.Freeze	1.94×10^{-11}
MBB.TL.Dense.Freeze	2.21×10^{-12}
MBB.TL.AddDense.Freeze	9.24×10^{-13}
ARIMA	1.22×10^{-24}
Prophet	1.11×10^{-30}
GRATIS.TL.Dense.Freeze	2.30×10^{-35}
GRATIS.TL.AddDense.Freeze	6.91×10^{-36}
ES	5.10×10^{-71}

Table 5
Results of statistical testing for the MASE error measure across all the datasets. Adjusted p-values calculated from the Friedman test with Hochbergs post-hoc procedure are shown. A horizontal line is used to separate the methods that perform significantly worse than the control method from the ones that do not.

Method	p_{Hoch}
DBA.TL.Dense.Freeze	-
MBB.TL.Dense.Retrain	0.396
DBA.TL.LSTM.Freeze	0.396
DBA.TL.Dense.Retrain	0.396
DBA.TL.AddDense.Freeze	0.396
DBA.TL.AddDense.Retrain	0.396
GRATIS.TL.Dense.Retrain	0.396
GRATIS.TL.AddDense.Retrain	0.396
LSTM.Baseline	0.191
GRATIS.TL.LSTM.Retrain	0.191
DBA.Pooled	0.161
MBB.TL.LSTM.Retrain	0.036
MBB.Pooled	0.001
GRATIS.TL.LSTM.Freeze	1.11×10^{-4}
DBA.TL.LSTM.Retrain	2.11×10^{-5}
MBB.TL.AddDense.Retrain	1.26×10^{-6}
MBB.TL.LSTM.Freeze	6.36×10^{-11}
MBB.TL.Dense.Freeze	3.30×10^{-13}
MBB.TL.AddDense.Freeze	1.06×10^{-13}
ARIMA	3.45×10^{-26}
GRATIS.TL.Dense.Freeze	1.71×10^{-29}
GRATIS.TL.AddDense.Freeze	4.34×10^{-30}
Prophet	1.27×10^{-30}
ES	1.13×10^{-71}

6.6. Computational performance

We report the computational costs in execution time of our proposed framework and the benchmark models on the NN5 dataset. The results on other datasets are comparable. The experiments are run on an Intel(R) i7 processor (3.2 GHz), with 2 threads per core, 6 cores in total, and 64GB of main memory.

6.7. Results and discussion

Table 3 summarises the overall performance of the proposed variants and the benchmarks, in terms of average ranking across all series in the benchmark suite. According to Table 3, the proposed DBA.TL.Dense.Freeze variant obtains the best Rank sMAPE and Rank MASE. It can be seen that many of the proposed variants outperform the baseline model, LSTM.Baseline on both evaluation metrics. Furthermore, except for the GRATIS.TL.Dense.Freeze and GRATIS.TL.AddDense.Freeze variants, all other proposed methods obtain better accuracies than the benchmarks ES, ARIMA, and Prophet.

Regarding statistical testing, the overall result of the Friedman rank sum test for sMAPE is a p-value of 2.66×10^{-10}, which means the results are highly significant. Table 4 shows the results of the post-hoc test. The DBA.TL.Dense.Freeze method performs best and is chosen as the control method. We can see from the table that the improvements in accuracy over the baseline LSTM.Baseline and the benchmarks are highly significant.

Table 5 shows the results of the statistical testing evaluation for the MASE error measure. The overall result of the Friedman rank sum test for MASE is a p-value of 2.58×10^{-10}, which means the results are highly significant. The DBA.TL.Dense.Freeze variant performs best and is again chosen as the control method. We see that the improvements over the LSTM.Baseline variant are not statistically significant in this instance, but the improvements over the benchmarks ES, Prophet, and ARIMA are highly significant.

After the analysis of the results overall across datasets, where we have been able to establish the statistical significance of the accuracy gains of our methods, we further investigate error distributions in more detail for each dataset. The results of all the proposed variants in terms of the mean sMAPE metric are shown in Table 6. We see that MBB.Pooled, DBA.TL.AddDense.Retrain, GRATIS.TL.Dense.Freeze, GRATIS.TL.AddDense.Freeze, and GRATIS.TL.AddDense.Retrain obtain the best Mean sMAPE for the AusEnergy-Demand, AusGrid-

Table 6
The Mean sMAPE results across all the benchmark datasets. For each dataset, the results of the best performing method(s) are marked in boldface.

Method	AusEnergy-Demand	AusGrid-Energy	Electricity	NN3	NN5
LSTM.Baseline	0.0550	0.2235	0.0900	0.1655	0.1075
MBB.Pooled	**0.0518**	0.2228	0.0903	0.1682	0.1080
DBA.Pooled	0.0607	0.2178	0.0884	0.1650	0.1071
MBB.TL.Dense.Freeze	0.0539	0.2324	0.0890	0.1659	0.1169
MBB.TL.Dense.Retrain	0.0558	0.2195	0.0884	0.1661	0.1068
MBB.TL.AddDense.Freeze	0.0539	0.2325	0.0893	0.1665	0.1166
MBB.TL.AddDense.Retrain	0.0545	0.2214	0.0900	0.1667	0.1090
MBB.TL.LSTM.Freeze	0.0548	0.2275	0.0928	0.1658	0.1154
MBB.TL.LSTM.Retrain	0.0556	0.2248	0.0887	0.1663	0.1086
DBA.TL.Dense.Freeze	0.0561	0.2150	0.0867	0.1654	0.1185
DBA.TL.Dense.Retrain	0.0565	0.2172	0.0881	0.1661	0.1072
DBA.TL.LSTM.Freeze	0.0560	0.2187	0.0869	0.1653	0.1166
DBA.TL.LSTM.Retrain	0.0551	0.2225	0.0896	0.1639	0.1132
DBA.TL.AddDense.Freeze	0.0539	**0.2147**	0.0871	0.1660	0.1242
DBA.TL.AddDense.Retrain	0.0553	0.2175	0.0882	0.1664	0.1073
GRATIS.TL.Dense.Freeze	0.0639	0.2629	**0.0853**	0.1673	0.1245
GRATIS.TL.Dense.Retrain	0.0543	0.2233	0.0870	0.1664	0.1067
GRATIS.TL.AddDense.Freeze	0.0643	0.2607	**0.0853**	0.1675	0.1264
GRATIS.TL.AddDense.Retrain	0.0597	0.2221	0.0867	0.1669	**0.1066**
GRATIS.TL.LSTM.Freeze	0.0566	0.2261	0.0865	0.1659	0.1189
GRATIS.TL.LSTM.Retrain	0.0557	0.2216	0.0879	0.1648	0.1090
Prophet	0.0595	0.2566	0.0996	0.2518	0.1143
ES	0.0642	0.3318	0.1136	**0.1532**	0.1211
ARIMA	0.0739	0.2619	0.0974	0.1564	0.1355

Energy, Electricity, and NN5 datasets. Whereas, the ES method achieves the best Mean sMAPE for the NN3 dataset. For the AusEnergy-Demand dataset, we observe that the majority of the MBB based knowledge transfer variants (both based on TL and pooled), can achieve better accuracy than the LSTM.Baseline. In terms of the AusGrid-Energy dataset, it can be seen that, in most cases, the DBA-based knowledge transfer variants obtain better forecasts than LSTM.Baseline, which is contrary to our previous findings from the AusEnergy-Demand dataset. It is also noteworthy to mention that both pooled variants, i.e., DBA.Pooled and MBB.Pooled, outperform the LSTM.Baseline in this dataset. For the Electricity dataset, we observe that all the GRATIS and DBA based variants outperform the LSTM.Baseline benchmark. Also, among the pooled variants, we see that only the DBA.Pooled method performs better than LSTM.Baseline. Moreover, it can be seen that all the variants that use TL.Dense.Freeze, TL.Dense.Retrain, TL.AddDense.Freeze, TL.AddDense.Retrain, and TL.LSTM.Retrain as the TL architecture, generate better forecasts than the baseline. For the NN3 dataset, even though our proposed variants are unable to outperform the statistical benchmarks, we see that some of the DBA based variants outperform our baseline model in terms of Mean sMAPE. Finally, with respect to the NN5 dataset, on average, we see that the variants that use the TL.Dense.Retrain and TL.AddDense.Retrain architectures achieve better accuracies compared with variants that use other TL architectures. Also, except for the NN3 dataset, we see that many of the proposed variants outperform the state-of-the-art forecasting methods, such as ES, Prophet, and ARIMA variants on Mean sMAPE.

Table 7 shows the results of all the proposed variants in terms of the median sMAPE metric. It can be seen that the proposed MBB.Pooled method, DBA.TL.Dense.Freeze method, MBB.TL.Dense.Freeze and DBA.Pooled method obtain the best Median sMAPE for the AusEnergy-Demand, AusGrid-Energy, Electricity, and NN5 datasets, respectively. The ES method achieves the best Median sMAPE for the NN3 dataset. Similar to the previous findings from Table 6, we see that for the AusEnergy-Demand dataset, the majority of the MBB based knowledge transfer variants generate better forecasts compared with the LSTM.Baseline. Also, for the AusGrid-Energy dataset, the results indicate that the DBA based knowledge transfer variants obtain better accuracy than the MBB based knowledge transfer variants and the LSTM.Baseline. Furthermore, both DBA.Pooled and the MBB.Pooled variants outperform the LSTM.Baseline for this dataset. In terms of the Electricity dataset, it can be seen that except for the DBA.Pooled, MBB.TL.LSTM.Freeze, and DBA.TL.LSTM.Retrain methods, and all other proposed variants manage to outperform the LSTM.Baseline benchmark. Moreover, among the pooled variants, it can be seen that only the MBA.Pooled method performs better than the LSTM.Baseline. For the NN3 dataset, we see that only the DBA.TL.LSTM.Freeze and GRATIS.TL.LSTM.Retrain variants can generate better accuracies compared with our baseline model. Concerning the NN5 dataset, on average, we see that both pooled variants can outperform the LSTM.Baseline benchmark. Overall, in terms of the Mean sMAPE error measure, we notice that the majority of our proposed methods achieve better results than the statistical benchmarks.

The results of the proposed variants in terms of the mean MASE error metric are as shown in Table 8. Apart from the NN3 dataset, we note that our proposed variants obtain the best accuracies for the rest of the datasets. In terms of the AusEnergy-Demand dataset, we observe that the MBB.Pooled variant achieves the best results outperforming the LSTM.Baseline. We note that the DBA.TL.AddDense.Freeze method obtains the best results for the AusGrid-Energy dataset. Here, we see that the DBA.Pooled method and the majority of the DBA based TL architectures achieve better results compared with our baseline variant. In terms of the Electricity dataset, we observe that the proposed GRATIS.TL.LSTM.Freeze achieves the best Mean MASE. Moreover, it can be seen that all the GRATIS based TL architectures outperform the LSTM.Baseline. Similar to the previous observations from Table 6 and Table 7, we see that the ES benchmark obtains the best Mean MASE for the NN3 dataset. However, we note that the DBA.Pooled method, DBA.TL.Dense.Freeze method, DBA.TL.LSTM.Retrain method and GRATIS.TL.LSTM.Retrain method outperform the LSTM.Baseline with respect to the Mean MASE. The GRATIS.TL.AddDense.Retrain variant achieves the best Mean MASE for the NN5 dataset. In addition to the GRATIS.TL.AddDense.Retrain method, and we see that

Table 7
The Median sMAPE results across all the benchmark datasets. For each dataset, the results of the best performing method(s) are marked in boldface.

Method	AusEnergy-Demand	AusGrid-Energy	Electricity	NN3	NN5
LSTM.Baseline	0.0536	0.1949	0.0590	0.1170	0.1028
MBB.Pooled	**0.0504**	0.1933	0.0583	0.1187	0.1014
DBA.Pooled	0.0590	0.1858	0.0618	0.1174	**0.1013**
MBB.TL.Dense.Freeze	0.0532	0.1998	**0.0563**	0.1172	0.1088
MBB.TL.Dense.Retrain	0.0540	0.1880	0.0574	0.1165	0.1034
MBB.TL.AddDense.Freeze	0.0533	0.1966	0.0564	0.1170	0.1073
MBB.TL.AddDense.Retrain	0.0542	0.1896	0.0570	0.1180	0.1040
MBB.TL.LSTM.Freeze	0.0536	0.1934	0.0592	0.1175	0.1090
MBB.TL.LSTM.Retrain	0.0538	0.1930	0.0571	0.1171	0.1018
DBA.TL.Dense.Freeze	0.0545	**0.1836**	0.0571	0.1173	0.1067
DBA.TL.Dense.Retrain	0.0540	0.1859	0.0587	0.1172	0.1022
DBA.TL.AddDense.Freeze	0.0548	0.1853	0.0573	0.1172	0.1100
DBA.TL.AddDense.Retrain	0.0540	0.1870	0.0586	0.1174	0.1052
DBA.TL.LSTM.Freeze	0.0545	0.1865	0.0578	0.1170	0.1065
DBA.TL.LSTM.Retrain	0.0544	0.1950	0.0603	0.1177	0.1059
GRATIS.TL.Dense.Freeze	0.0631	0.2389	0.0572	0.1174	0.1085
GRATIS.TL.Dense.Retrain	0.0534	0.1929	0.0566	0.1185	0.1039
GRATIS.TL.AddDense.Freeze	0.0643	0.2371	0.0576	0.1176	0.1089
GRATIS.TL.AddDense.Retrain	0.0574	0.1892	0.0564	0.1176	0.1045
GRATIS.TL.LSTM.Freeze	0.0578	0.1926	0.0569	0.1172	0.1081
GRATIS.TL.LSTM.Retrain	0.0548	0.1895	0.0577	0.1166	0.1028
Prophet	0.0565	0.2158	0.0621	0.1926	0.1062
ARIMA	0.0580	0.2216	0.0660	0.1182	0.1220
ES	0.0606	0.2852	0.0934	**0.1135**	0.1079

Table 8
The Mean MASE results across all the benchmark datasets. For each dataset, the results of the best performing method(s) are marked in boldface.

Method	AusEnergy-Demand	AusGrid-Energy	Electricity	NN3	NN5
LSTM.Baseline	0.9564	0.8183	0.7512	0.9471	0.7999
MBB.Pooled	**0.9073**	0.8271	0.7497	0.9625	0.8050
DBA.Pooled	1.0504	0.8087	0.7741	0.9459	0.7946
MBB.TL.Dense.Freeze	0.9372	0.8661	0.7475	0.9494	0.8602
MBB.TL.Dense.Retrain	0.9718	0.8117	0.7385	0.9499	0.7942
MBB.TL.AddDense.Freeze	0.9357	0.8686	0.7483	0.9518	0.8576
MBB.TL.AddDense.Retrain	0.9444	0.8233	0.7577	0.9580	0.8097
MBB.TL.LSTM.Freeze	0.9530	0.8483	0.7690	0.9509	0.8485
MBB.TL.LSTM.Retrain	0.9690	0.8396	0.7338	0.9533	0.8094
DBA.TL.Dense.Freeze	0.9669	0.7879	0.7393	0.9463	0.8733
DBA.TL.Dense.Retrain	0.9858	0.7990	0.7523	0.9501	0.7972
DBA.TL.AddDense.Freeze	0.9234	**0.7874**	0.7438	0.9496	0.9142
DBA.TL.AddDense.Retrain	0.9684	0.8008	0.7517	0.9547	0.7981
DBA.TL.LSTM.Freeze	0.9717	0.8036	0.7321	0.9492	0.8581
DBA.TL.LSTM.Retrain	0.9626	0.8229	0.7601	0.9371	0.8396
GRATIS.TL.Dense.Freeze	1.1032	0.9974	0.7378	0.9572	0.9142
GRATIS.TL.Dense.Retrain	0.9402	0.8242	0.7306	0.9550	0.7929
GRATIS.TL.AddDense.Freeze	1.0921	0.9859	0.7375	0.9582	0.9280
GRATIS.TL.AddDense.Retrain	1.0235	0.8200	0.7306	0.9567	**0.7921**
GRATIS.TL.LSTM.Freeze	0.9676	0.8383	**0.7305**	0.9509	0.8732
GRATIS.TL.LSTM.Retrain	0.9635	0.8148	0.7415	0.9443	0.8104
Prophet	1.0627	0.9039	0.8034	1.3233	0.8496
ARIMA	1.2111	0.9603	0.8379	0.9235	1.0021
ES	1.0473	1.2839	1.1696	**0.8942**	0.8918

the proposed DBA.Pooled method, MBB.TL.Dense.Retrain method, DBA.TL.Dense.Retrain method, DBA.TL.AddDense.Retrain method, and GRATIS.TL.Dense.Retrain method obtain better Mean MASE values compared with the LSTM.Baseline. Overall, with respect to the Mean MASE error measure, we see that in the majority of cases, our proposed methods outperform the statistical benchmarks.

Table 9 shows the results of all the proposed variants in terms of the median MASE metric. We see that the proposed MBB.TL.AddDense.Freeze variant obtains the best Median MASE for the AusEnergy-Demand dataset. Whereas, the DBA.TL.Dense.Freeze method obtains the best results for both AusGrid-Energy and Electricity datasets. For the AusGrid-Energy dataset, it can be seen that both the MBB.Pooled and DBA.Pooled methods outperform the LSTM.Baseline. Also, the majority of the DBA based TL architectures achieve better results compared with our baseline variant. For the Electricity dataset, it can be seen that DBA and GRATIS based TL architectures generate accurate forecasts compared with the LSTM.Baseline. The DBA.TL.LSTM.Retrain variant achieves the best Median MASE for the NN3 dataset. This is contrary to our previous findings from Table 6, Table 7, and Table 8, in which the statistical benchmarks outperformed our variants on the NN3 dataset. Also, we notice that the majority of the proposed variants achieve better Median MASE values compared with the Prophet, ARIMA, and ES benchmarks. In terms of the NN5 dataset, we see that the proposed DBA.TL.Dense.Retrain method achieves the best results. Furthermore, we see that the proposed DBA.Pooled method, MBB.TL.Dense.Retrain method, MBB.TL.LSTM.Retrain method, and GRATS.TL.AddDense.Retrain method outperform the LSTM.Baseline.

Table 9
The Median MASE results across all the benchmark datasets. For each dataset, the results of the best performing method(s) are marked in boldface.

Method	AusEnergy-Demand	AusGrid-Energy	Electricity	NN3	NN5
LSTM.Baseline	0.8552	0.7508	0.6982	0.7518	0.7374
MBB.Pooled	0.8575	0.7399	0.6951	0.7730	0.7579
DBA.Pooled	0.9926	0.7210	0.7097	0.7684	0.7340
MBB.TL.Dense.Freeze	0.8638	0.8036	0.6919	0.7806	0.7751
MBB.TL.Dense.Retrain	0.8796	0.7427	0.6731	0.7629	0.7342
MBB.TL.AddDense.Freeze	**0.8499**	0.802	0.6850	0.7629	0.7597
MBB.TL.AddDense.Retrain	0.8671	0.7574	0.7145	0.7735	0.7458
MBB.TL.LSTM.Freeze	0.8693	0.7726	0.7118	0.7519	0.7464
MBB.TL.LSTM.Retrain	0.8710	0.7640	0.6859	0.7655	0.7201
DBA.TL.Dense.Freeze	0.8721	**0.7130**	**0.6558**	0.7735	0.7782
DBA.TL.Dense.Retrain	0.8750	0.7308	0.6938	0.7653	**0.7186**
DBA.TL.AddDense.Freeze	0.8516	0.7140	0.6600	0.7551	0.8284
DBA.TL.AddDense.Retrain	0.8697	0.7287	0.6976	0.7734	0.7451
DBA.TL.LSTM.Freeze	0.8818	0.7464	0.6574	0.7725	0.7534
DBA.TL.LSTM.Retrain	0.8720	0.7601	0.6965	**0.7438**	0.7720
GRATIS.TL.Dense.Freeze	1.1804	0.9192	0.6695	0.7705	0.8415
GRATIS.TL.Dense.Retrain	0.8565	0.7430	0.6821	0.7696	0.7529
GRATIS.TL.AddDense.Freeze	1.0672	0.9122	0.6630	0.7675	0.8548
GRATIS.TL.AddDense.Retrain	0.9173	0.7431	0.6824	0.7590	0.7322
GRATIS.TL.LSTM.Freeze	0.8951	0.7628	0.6779	0.7491	0.8055
GRATIS.TL.LSTM.Retrain	0.8703	0.7516	0.6745	0.7471	0.7398
Prophet	0.9809	0.8394	0.7245	1.2338	0.8625
ARIMA	1.0978	0.8300	0.7884	0.7828	0.9385
ES	1.0652	1.0525	1.0181	0.7793	0.7620

Table 10
The computational time for the NN5 Dataset, ordered by the last column, which is the total computation time in seconds.

Method	Timeseries-generation	Model-Training	Total-time
ARIMA	-	19	19
Prophet	-	40	40
ES	-	120	120
LSTM.Baseline	-	994	994
MBB.Pooled	4	1253	1257
DBA.Pooled	13	1253	1266
DBA.TL.AddDense.Freeze	13	1629	1642
MBB.TL.AddDense.Freeze	4	1675	1679
DBA.TL.Dense.Freeze	13	1848	1861
MBB.TL.Dense.Freeze	4	1970	1974
GRATIS.TL.Dense.Freeze	160	2167	2327
GRATIS.TL.AddDense.Freeze	160	2188	2348
DBA.TL.AddDense.Retrain	13	2747	2760
MBB.TL.Dense.Retrain	4	3057	3061
MBB.TL.AddDense.Retrain	4	3060	3064
DBA.TL.Dense.Retrain	13	3732	3745
GRATIS.TL.AddDense.Retrain	160	4560	4720
DBA.TL.LSTM.Freeze	13	4454	4467
MBB.TL.LSTM.Freeze	4	4623	4627
DBA.TL.LSTM.Retrain	13	4575	4588
GRATIS.TL.Dense.Retrain	160	4575	4735
MBB.TL.LSTM.Retrain	4	5209	5213
GRATIS.TL.LSTM.Freeze	160	5510	5670
LGRATIS.TL.LSTM.Retrain	160	6283	6443

Similar to our previous findings from Table 6, Table 7, and Table 8, we see that the majority of our proposed variants outperform the statistical benchmarks concerning the Median MASE error measure.

Table 10 provides a summary of computational cost of the proposed variants and benchmarks on the NN5 dataset. According to Table 10, we see that the statistical benchmarks, such as ARIMA, Prophet and ES, have a lower execution time compared with the proposed variants. Nonetheless, according to Table 6, Table 7, Table 8, Table 9, we see that these benchmarks do not display competitive results compared with the proposed variants. With respect to computational cost among the proposed methods, we observe that the variants that use pooled approach for knowledge transfer have less computational time than TL-based methods. This is due to their additional model pre-training and model transfer procedure. Furthermore, as the complexity of the TL architecture increases, the total computational time also gradually increases. Here, the TL based variants that freeze the initial layers and do not re-train the layers of the pre-trained model, i.e., TL.Dense.Freeze, TL.AddDense.Freeze, and TL.LSTM.Freeze, consume a smaller amount of time than their counterpart architectures that re-train the layers of the pre-trained model, i.e., TL.Dense.Retrain, TL.AddDense.Retrain, and MBB.TL.LSTM.Retrain (see Section 4). Also, with respect to the time series data augmentation techniques, we see that the GRATIS method takes the highest amount of time to generate the target time series.

To summarise, the proposed DBA based variants achieve competitive results in our experiments. One exception to this is the

AusEnergy-Demand dataset, where MBB based variants outperform the DBA based methods. It can be mainly attributed to the small size of the AusEnergy-Demand dataset, where the DBA technique performs poorly as the number of time series in the source dataset is limited. As the MBB technique generates time series independent of the number of time series available in the source dataset as it augments each time series in isolation, the MBB based variants outperform the DBA based variants in this scenario. The better performance of DBA and MBB based variants over GRATIS based variants also indicates that DA techniques that generate time series similar to the original distribution of the dataset contribute more towards improving the base accuracy of the models. The results also indicate that, in most cases, the proposed variants can outperform state-of-the-art statistical forecasting techniques, such as ES, ARIMA, and Prophet. In situations where our methods are unable to perform better than the statistical benchmarks, we observe that the majority of our proposed variants can improve the accuracy of the baseline model.

7. Conclusions

Generating accurate forecasts with a limited number of time series can be a challenging task for global forecasting models that train across all the available time series. In this study, we have introduced a novel data augmentation based forecasting framework to supplement recurrent neural network based global model architectures, when used in settings with limited amounts of available data. We have used three time series augmentation techniques to produce time series synthetically. They include the Moving Block Bootstrap and the Dynamic Time-Warping Barycentric Averaging techniques that are capable of generating time series that are similar to those in the original dataset, and the GRATIS method that generates time series with diverse characteristics, which can be dissimilar to those in the original dataset.

To transfer knowledge representations from the augmented dataset to the target dataset with less data, we have employed two strategies; the pooled and transfer learning strategies. The pooled strategy trains on the augmented time series together with the original time series database, while the transfer learning strategy initially pre-trains a global model using the augmented time series, and then transfers the pre-trained knowledge representations to the target dataset using various transfer learning architectures.

We have evaluated our methods using five benchmark datasets, including two competition datasets and three real-world datasets. The results have shown that the proposed variants achieve competitive results under small to medium training set size conditions, outperforming the baseline global model and many state-of-the-art univariate forecasting methods with statistical significance. The results also indicate that data augmentation techniques that generate time series with similar characteristics to the target dataset achieve better results than those that generate time series with diverse characteristics. Nonetheless, the results suggest that the subset of GRATIS based variants, which re-trains the pre-trained model and newly added layers, can be a competitive approach among the baseline global model and univariate forecasting methods. This highlights the fact that resembling the general characteristics of time series, and then transferring this information to the target dataset can be useful to improve model accuracy, even if the augmented time series are diverse and different from the target dataset. Furthermore, we observe that the choice of the proposed strategies can be determined by the size of the original dataset, where the pooling strategy is more suitable for situations where the size of the original dataset is small, and the transfer strategy is better if the dataset is more extensive.

As a possible future work, more sophisticated feature extraction techniques, such as Encoder-Decoder architectures could be used, replacing the stacking architecture of our model. The extracted, latent features could then be re-purposed to train on a target dataset for forecasting.

Declaration of Competing Interest

We, all the authors, confirm that

- There are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome
- The manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. We further confirm that the order of authors listed in the manuscript has been approved by all of us.
- We have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property.
- We understand that the Corresponding Author is the sole contact for the Editorial process (including Editorial Manager and direct communications with the office). He is responsible for communicating with the other authors about progress, submissions of revisions and final approval of proofs. We confirm that we have provided a current, correct email address which is accessible by the Corresponding Author and which has been configured to accept email from Herath.Bandara@monash.edu

Acknowledgements

This research was supported by the Australian Research Council under grant DE190100045, by a Facebook Statistics for Improving Insights and Decisions research award, by Monash Institute of Medical Engineering seed funding, by the MASSIVE - High performance computing facility, Australia, and by the National Natural Science Foundation of China (No. 11701022).

References

[1] T. Januschowski, J. Gasthaus, Y. Wang, D. Salinas, V. Flunkert, M. Bohlke-Schneider, L. Callot, Criteria for classifying forecasting methods, Int. J. Forecast. 36 (1) (2020) 167–177.
[2] R.J. Hyndman, A.B. Koehler, J. Keith Ord, R.D. Snyder, Forecasting with exponential smoothing: The state space approach, Springer Science & Business Media, 2008.
[3] G.E.P. Box, G.M. Jenkins, G.C. Reinsel, G.M. Ljung, Time series analysis: Forecasting and control, John Wiley & Sons, 2015.
[4] S. Smyl, A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting, Int J Forecast 36 (1) (2020) 75–85.
[5] D. Salinas, V. Flunkert, J. Gasthaus, T. Januschowski, Deepar: probabilistic forecasting with autoregressive recurrent networks, Int J Forecast 36 (3) (2020) 1181–1191.
[6] K. Bandara, C. Bergmeir, S. Smyl, Forecasting across time series databases using recurrent neural networks on groups of similar series: a clustering approach, Expert Syst. Appl. 140 (2020) 112896.
[7] H. Hewamalage, C. Bergmeir, K. Bandara, Recurrent neural networks for time series forecasting: current status and future directions, International Journal of Forecasting (in press) (2020).
[8] K. Bandara, C. Bergmeir, H. Hewamalage, LSTM-MSNet: Leveraging forecasts on sets of related time series with multiple seasonal patterns, IEEE Trans. Neural Netw. Learn. Syst. (2020) 1–14, doi:10.1109/TNNLS.2020.2985720.
[9] Y. Chen, Y. Kang, Y. Chen, Z. Wang, Probabilistic forecasting with temporal convolutional neural network, Neurocomputing 399 (2020) 491–501.
[10] K. Bandara, C. Bergmeir, S. Campbell, D. Scott, D. Lubman, Towards accurate predictions and causal 'what-if' analyses for planning and policy-making: A case study in emergency medical services demand, 2020.
[11] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.

[12] Z. Wang, Z. Zhou, H. Lu, J. Jiang, Global and local sensitivity guided key salient object re-augmentation for video saliency detection, Pattern Recognit. 103 (2020) 107275.

[13] X. Zhang, J. Zhao, Y. LeCun, Character-level Convolutional Networks for Text Classification, in: C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Systems 28, Curran Associates, Inc., 2015, pp. 649–657.

[14] Z. Donyavi, S. Asadi, Diverse training dataset generation based on a multi-objective optimization for semi-supervised classification, Pattern Recognit. 108 (2020) 107543.

[15] G. Forestier, F. Petitjean, H.A. Dau, G.I. Webb, E. Keogh, Generating synthetic time series to augment sparse datasets, in: 2017 IEEE International Conference on Data Mining (ICDM), 2017, pp. 865–870.

[16] C. Bergmeir, R.J. Hyndman, J.M. Benítez, Bagging exponential smoothing methods using STL decomposition and box–Cox transformation, Int. J. Forecast. 32 (2) (2016) 303–312.

[17] H.I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, P.-A. Muller, Data augmentation using synthetic data for time series classification with deep residual networks (2018). 1808.02455.

[18] Y. Kang, R.J. Hyndman, F. Li, GRATIS: Generating TIme series with diverse and controllable characteristics, Stat. Anal. Data Min. 13 (2020) 354–376.

[19] J. Yosinski, J. Clune, Y. Bengio, H. Lipson, How Transferable Are Features in Deep Neural Networks?, in: Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger (Eds.) Advances in Neural Information Processing Systems 27, Curran Associates, Inc., 2014, pp. 3320–3328.

[20] B. Huang, T. Xu, J. Li, Z. Shen, Y. Chen, Transfer learning-based discriminative correlation filter for visual tracking, Pattern Recognit. 100 (2020) 107157.

[21] N. Zhuang, Y. Yan, S. Chen, H. Wang, C. Shen, Multi-label learning based deep transfer neural network for facial attribute classification, Pattern Recognit. 80 (2018) 225–240.

[22] X. Li, Y. Grandvalet, F. Davoine, A baseline regularization scheme for transfer learning with convolutional neural networks, Pattern Recognit. 98 (2020) 107049.

[23] S. Purushotham, W. Carvalho, T. Nilanon, Y. Liu, Variational recurrent adversarial deep domain adaptation, ICLR, 2017.

[24] S. Yoon, H. Yun, Y. Kim, G.-T. Park, K. Jung, Efficient transfer learning schemes for personalized language modeling using recurrent neural network (2017). 1701.03578.

[25] X. Glorot, A. Bordes, Y. Bengio, Domain adaptation for large-scale sentiment classification: A deep learning approach, in: Proceedings of the 28th International Conference on International Conference on Machine Learning, in: ICML'11, Omnipress, USA, 2011, pp. 513–520.

[26] X. Li, Y. Kang, F. Li, Forecasting with time series imaging, Expert Syst. Appl. 160 (2020) 113680.

[27] K. Bandara, P. Shi, C. Bergmeir, H. Hewamalage, Q. Tran, B. Seaman, Sales demand forecast in e-commerce using a long Short-Term memory neural network methodology, in: Neural Information Processing, Springer International Publishing, 2019, pp. 462–474.

[28] Y. Chen, P. Li, B. Zhang, Bayesian renewables scenario generation via deep generative networks, in: 2018 52nd Annual Conference on Information Sciences and Systems (CISS), 2018, pp. 1–6.

[29] C. Esteban, S.L. Hyland, G. Rätsch, Real-valued (medical) time series generation with recurrent conditional GANs (2017). 1706.02633.

[30] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, A.Y. Ng, Deep speech: Scaling up end-to-end speech recognition (2014). 1412.5567.

[31] N. Iftikhar, X. Liu, S. Danalachi, F.E. Nordbjerg, J.H. Vollesen, A scalable smart meter data generator using spark, in: On the Move to Meaningful Internet Systems. OTM 2017 Conferences, Springer International Publishing, 2017, pp. 21–36.

[32] E.A. Denaxas, R. Bandyopadhyay, D. Patiño-Echeverri, N. Pitsianis, SynTiSe: A modified multi-regime MCMC approach for generation of wind power synthetic time series, in: 2015 Annual IEEE Systems Conference (SysCon) Proceedings, 2015, pp. 668–674.

[33] G. Papaefthymiou, B. Klockl, MCMC For wind power simulation, IEEE Trans. Energy Convers. 23 (1) (2008) 234–240.

[34] L. Kegel, M. Hahmann, W. Lehner, Feature-based comparison and generation of time series, in: Proceedings of the 30th International Conference on Scientific and Statistical Database Management - SSDBM '18, ACM Press, New York, New York, USA, 2018, pp. 1–12.

[35] F. Almonacid, P. Pérez-Higueras, P. Rodrigo, L. Hontoria, Generation of ambient temperature hourly time series for some spanish locations by artificial neural networks, Renew. Energy 51 (2013) 285–291.

[36] A. Le Guennec, S. Malinowski, R. Tavenard, Data augmentation for time series classification using convolutional neural networks, ECML/PKDD workshop on advanced analytics and learning on temporal data, 2016.

[37] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative Adversarial Nets, in: Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 27, Curran Associates, Inc., 2014, pp. 2672–2680.

[38] R. Fu, J. Chen, S. Zeng, Y. Zhuang, A. Sudjianto, Time series simulation by conditional generative adversarial net (2019). 1904.11419.

[39] J. Yoon, D. Jarrett, M. van der Schaar, Time-series Generative Adversarial Networks, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 5508–5518.

[40] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359.

[41] Y. Yu, H. Lin, J. Meng, X. Wei, H. Guo, Z. Zhao, Deep transfer learning for modality classification of medical images, Information 8 (3) (2017) 91.

[42] Y. Bengio, Deep learning of representations for unsupervised and transfer learning, in: I. Guyon, G. Dror, V. Lemaire, G. Taylor, D. Silver (Eds.), Proceedings of ICML Workshop on Unsupervised and Transfer Learning, Proceedings of Machine Learning Research, volume 27, PMLR, Bellevue, Washington, USA, 2012, pp. 17–36.

[43] P. Ramachandran, P.J. Liu, Q. Le, Unsupervised pretraining for sequence to sequence learning, in: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, 2017, pp. 383–391.

[44] M. Ribeiro, K. Grolinger, H.F. ElYamany, W.A. Higashino, M.A.M. Capretz, Transfer learning with seasonal and trend adjustment for cross-building energy forecasting, Energy Build. 165 (2018) 352–363.

[45] N. Laptev, J. Yu, R. Rajagopal, Reconstruction and regression loss for time-series transfer learning, in: Proceedings of the Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) and the 4th Workshop on the Mining and LEarning from Time Series (MiLeTS), London, UK, volume 20, 2018.

[46] R. Ye, Q. Dai, A novel transfer learning framework for time series forecasting, Knowl. Based Syst. 156 (2018) 74–99.

[47] P. Gupta, P. Malhotra, L. Vig, G. Shroff, Transfer learning for clinical time series analysis using recurrent neural networks (2018). 1807.01705.

[48] R.B. Cleveland, W.S. Cleveland, I. Terpenning, STL: A seasonal-trend decomposition procedure based on loess, J. Off. Stat. 6 (1) (1990) 3–73.

[49] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2013.

[50] R.J. Hyndman, G. Athanasopoulos, C. Bergmeir, G. Caceres, L. Chhay, M. O'Hara-Wild, F. Petropoulos, S. Razbash, E. Wang, F. Yasmeen, forecast: Forecasting functions for time series and linear models, 2019. R package version 8.5, http://pkg.robjhyndman.com/forecast.

[51] R.J. Hyndman, Y. Khandakar, Automatic time series forecasting: the forecast package for r, J. Stat. Softw. 27 (03) (2008).

[52] T. Mikolov, M. Karafiát, L. Burget, J. Cernocky, S. Khudanpur, Recurrent neural network based language model, in: Interspeech, volume 2, fit.vutbr.cz, 2010, p. 3.

[53] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to Sequence Learning with Neural Networks, in: Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 27, Curran Associates, Inc., 2014, pp. 3104–3112.

[54] H.-G. Zimmermann, C. Tietz, R. Grothmann, Forecasting with Recurrent Neural Networks: 12 Tricks, in: Neural Networks: Tricks of the Trade, Springer, Berlin, Heidelberg, 2012, pp. 687–707.

[55] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neural Netw. 5 (2) (1994) 157–166.

[56] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[57] S. Smyl, K. Kuber, Data preprocessing and augmentation for multiple short time series forecasting with recurrent neural networks, 36th International Symposium on Forecasting, 2016.

[58] S. Makridakis, E. Spiliotis, V. Assimakopoulos, The M4 competition: results, findings, conclusion and way forward, Int. J. Forecast. 34 (4) (2018) 802–808.

[59] S. Ben Taieb, G. Bontempi, A.F. Atiya, A. Sorjamaa, A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition, Expert Syst. Appl. 39 (8) (2012) 7067–7083.

[60] J.L. Elman, Finding structure in time, Cogn. Sci. 14 (2) (1990) 179–211.

[61] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN Encoder-Decoder for statistical machine translation(2014). 1406.1078.

[62] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, Others, Tensorflow: A system for large-scale machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.

[63] Y. Kang, M. O'Hara-Wild, R.J. Hyndman, F. Li, GRATIS: GeneRAting TIme Series with diverse and controllable characteristics, 2020. Accessed: 2020-2-11, https://github.com/ykang/gratis.

[64] G. Athanasopoulos, H. Song, J.A. Sun, Bagging in tourism demand modeling and forecasting, J. Travel Res. 57 (1) (2018) 52–68.

[65] F. Petitjean, DBA: Averaging for dynamic time warping, 2017, (https://github.com/fpetitjean/DBA),Accessed: 2020-6-17.

[66] S.F. Crone, NN5 competition, 2008, (http://www.neural-forecasting-competition.com/NN5/),Accessed: 2017-8-18.

[67] S.F. Crone, M. Hibon, K. Nikolopoulos, Advances in forecasting with neural networks? empirical evidence from the NN3 competition on time series prediction, Int. J. Forecast. 27 (3) (2011) 635–660.

[68] AEMO, Data dashboard NEM, 2020, (https://www.aemo.com.au/energy-systems/electricity/national-electricity-market-nem/data-nem/data-dashboard-nem)Accessed: 2020-6-30.

[69] AusGrid, Innovation and research - ausgrid, 2019, (https://www.ausgrid.com.au/Industry/Innovation-and-research/),Accessed: 2019-5-16.

[70] G. Lai, Multivariate time series forecasting, 2018, (https://github.com/laiguokun/multivariate-time-series-data),Accessed: 2020-6-30.

[71] R.J. Hyndman, A.B. Koehler, Another look at measures of forecast accuracy, Int. J. Forecast. (2006).

[72] A. Suilin, Kaggle-web-traffic, 2018, (https://github.com/Arturus/kaggle-web-traffic),Accessed: 2020-2-10.
[73] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power, Inf. Sci. 180 (10) (2010) 2044–2064.
[74] F. Hutter, H.H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Proceedings of the 5th International Conference on Learning and Intelligent Optimization, in: LION'05, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 507–523.
[75] AutoML Group, Smac v3: Algorithm configuration in python, 2017, (https://github.com/automl/SMAC3),Accessed: 2020-2-13.
[76] F. Orabona, T. Tommasi, Training deep networks without learning rates through coin betting, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, in: NIPS'17, Curran Associates Inc., USA, 2017, pp. 2157–2167.
[77] F. Orabona, cocob, 2017, (https://github.com/bremen79/cocob),Accessed: 2020-2-13.
[78] I. Svetunkov, smooth: Forecasting Using State Space Models, 2020. R package version 2.6.0, https://CRAN.R-project.org/package=smooth.
[79] S.J. Taylor, B. Letham, Forecasting at scale, Technical Report, PeerJ Preprints, 2017.

Christoph Bergmeir is a Senior Lecturer and a 2019 ARC DECRA Fellow in the Department of Data Science and Artificial Intelligence at Monash University. His fellowship is on the development of “efficient and effective analytics for real-world time series forecasting.” He works as a Data Scientist in a variety of projects with external partners in diverse sectors, e.g. in healthcare or infrastructure maintenance. Christoph holds a PhD in Computer Science from the University of Granada, Spain, and an M.Sc. degree in Computer Science from the University of Ulm, Germany. He has published on time series prediction using Machine Learning methods, recurrent neural networks and long short-term memory neural networks (LSTM), time series predictor evaluation, as well as on medical applications and software packages in the R programming language.

Kasun Bandara received the B.Sc. honours degree in Computer Science from the University of Colombo School of Computing, Sri-Lanka, in 2015. He is currently pursing a Ph.D. degree in Computer Science at the Faculty of Information Technology, Monash University, Melbourne, Australia. His research interests include Big Data, deep neural networks and time series forecasting. He has published in journals such as IEEE Transactions on Neural Networks and Learning Systems, International Journal of Forecasting, and Expert Systems with Applications.

Hansika Hewamalage received the B.Sc. honours degree in Computer Science from the University of Moratuwa, Sri-Lanka, in 2015. She is currently pursing a Ph.D. degree in Computer Science at the Faculty of Information Technology, Monash University, Melbourne, Australia. Her research interests include Big Data, deep neural networks and time series forecasting.

Yanfei Kang is Associate Professor of Statistics in the School of Economics and Management at Beihang University in China. Prior to that, she was Senior R&D Engineer in Big Data Group of Baidu Inc. Yanfei obtained her Ph.D. degree at Monash University in 2014. She worked as a postdoctoral research fellow in feature-based time series forecasting during 2014 and 2015 at Monash University. Her research interests include time series forecasting, time series visualization, statistical computing and machine learning.

Yuan-Hao Liu finished his Master degree at Monash university and works as a data science engineer in Taiwan. His research interest includes areas of deep learning, time series forecasting and transfer learning.