Expert Systems With Applications 140 (2020) 112896

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach

Kasun Bandara[a,*], Christoph Bergmeir[a], Slawek Smyl[b]

[a] Faculty of Information Technology, Monash University, P.O. Box 63, Victoria 3800, Melbourne, Australia
[b] Uber Technologies Inc, 1455 Market St 400, San Francisco, CA 94103, United States

ARTICLE INFO

Article history:
Received 28 April 2019
Revised 18 July 2019
Accepted 21 August 2019
Available online 26 August 2019

Keywords:
Big data forecasting
RNN
LSTM
Time series clustering
Neural networks

ABSTRACT

With the advent of Big Data, nowadays in many applications databases containing large quantities of similar time series are available. Forecasting time series in these domains with traditional univariate forecasting procedures leaves great potentials for producing accurate forecasts untapped. Recurrent neural networks (RNNs), and in particular Long Short Term Memory (LSTM) networks, have proven recently that they are able to outperform state-of-the-art univariate time series forecasting methods in this context, when trained across all available time series. However, if the time series database is heterogeneous, accuracy may degenerate, so that on the way towards fully automatic forecasting methods in this space, a notion of similarity between the time series needs to be built into the methods. To this end, we present a prediction model that can be used with different types of RNN models on subgroups of similar time series, which are identified by time series clustering techniques. We assess our proposed methodology using LSTM networks, a widely popular RNN variant, together with various clustering algorithms, such as kMeans, DBScan, Partition Around Medoids (PAM), and Snob. Our method achieves competitive results on benchmarking datasets under competition evaluation procedures. In particular, in terms of mean sMAPE accuracy it consistently outperforms the baseline LSTM model, and outperforms all other methods on the CIF2016 forecasting competition dataset.

1. Introduction

In the time series forecasting community there has been the long-standing consensus that sophisticated methods do not necessarily produce better forecasts than simpler ones. This was a conclusion of the influential M3 forecasting competition held in 1999 (Makridakis & Hibon, 2000). So, complex methods are often viewed poorly in this field, and this has been especially true for neural networks (NN) and other Machine Learning (ML) techniques. In particular, AutomatANN, the only NN variant that participated in the M3, could not outperform statistical approaches that mostly headlined the rankings. NNs did also not perform well in subsequent competitions, e.g., in the NN3 and NN5 forecasting competitions, which were held specifically for ML methods. In the NN3 competition (Crone, Hibon, & Nikolopoulos, 2011), only one participating ML method was able to outperform damped trend exponential smoothing, and none of the methods was able to outperform the Theta method, which is equivalent to simple exponential smoothing with drift (Hyndman & Billah, 2003). Both these methods are relatively simple standard methods in time series forecasting.

The reasons for this under-performance of sophisticated methods can be attributed to individual series usually being too short to be modelled effectively using ML methods such as NNs. From short individual series the amount of information that can be extracted is limited (Yan, 2012; Zhang, Patuwo, & Hu, 1998). In such a situation, simpler, more rigid models not sensitive to noise and with reasonable prior assumptions about the data will typically perform well. Complex models, such as NNs, in contrast may not have enough data to fit their parameters reliably, and without proper regularization they are in danger of overfitting, i.e., they may fit to the random noise in the training data.

Also, even if large amounts of data are available, the distant past is usually less useful for forecasting, as underlying patterns and relationships will have changed in the meantime. So, a common notion is that unless the underlying time series is very long and from a very stable system, NNs will not be able to substantially outperform simpler models, as they will not have enough data to fit complex models or they will not handle non-stationarity in the data adequately (Hyndman, 2016).

* Corresponding author.
E-mail addresses: herath.bandara@monash.edu (K. Bandara), christoph.bergmeir@monash.edu (C. Bergmeir), slawek@uber.com (S. Smyl).

https://doi.org/10.1016/j.eswa.2019.112896
0957-4174/

These problems preventing NNs from being successful in a univariate time series context do not simply vanish with the advent of "Big Data", where ever increasing quantities of data are collected nowadays by many companies for the routine functioning of their businesses, for example server performance measures in computer centers, sales in retail of thousands of different products, measurements for predictive maintenance, smart meter data, etc. This is because in a time series context, availability of more data does not usually mean that the isolated series change or contain more data, e.g., that they are longer or have a higher sampling rate, as these are determined by the application and not by capturing and storage capabilities. Instead, it means that large quantities of related, similar series are available. Therefore, forecasting time series in these domains with traditional univariate forecasting procedures leaves great potential for producing more accurate forecasts untapped, as a separate model is built for each time series, and no information from other series is taken into account. On the other hand, in this situation now considerable more data is available, and using more complex models becomes feasible, when trained globally across all series.

Given these considerations, a competitive advantage unfolds to forecasting models that can be trained globally across all series, where traditional univariate forecasting techniques such as ETS, ARIMA, Theta, etc., are unable to exercise. For example, Hartmann, Hahmann, Lehner, and Rosenthal (2015) introduce a cross-sectional regression model to sets of related time series observed at the same period of time to alleviate the presence of missing values in a single time series. Also, Trapero, Kourentzes, and Fildes (2015) use a pooled regression model by aggregating sets of related time series to produce reliable promotional forecasts in the absence of historical sales data. However, universal function approximation properties, i.e., capacity to estimate linear and non-linear functions (Cybenko, 1989; Hornik, 1991), along with the large quantities of time series data available with the exposure of Big Data, have positioned NNs as ideal candidates to exploit the information dispersed across many time series.

When building such global models for a time series database, now the problem arises that these global models are potentially trained across disparate series, which may be detrimental to the overall accuracy. We propose to overcome this shortcoming by building separate models for subgroups of time series. The grouping can be based on additional domain knowledge available, or, in the absence of such a natural grouping, we propose a fully automatic mechanism that works on time series databases in general, which accounts for the dissimilarities in a set of time series. To assess our methodology, we use Long Short-term Memory (LSTM) networks, a promising Recurrent Neural Network (RNN) variant, which is heavily used in sequence modelling. Popular application domains are Natural Language Processing (Mikolov, Karafiát, Burget, Cernock\'y, & Khudanpur, 2010), machine translation (Sutskever, Vinyals, & Le, 2014), and speech recognition (Graves, r. Mohamed, & Hinton, 2013), and LSTMs are also gaining popularity in time series research (Lipton, Kale, Elkan, & Wetzel, 2015; Zimmermann, Tietz, & Grothmann, 2012). Moreover, our proposed methodology can be generalised to any RNN variant such as LSTMs, Gated Recurrent Units (GRUs), and others.

Specifically, our proposed method initially discovers clusters of similar series from the overall set of time series, as an augmentation step to exploit the similarity between time series. We propose a feature-based clustering approach using a set of interpretable features of a time series to obtain meaningful clusters. Firstly, we extract the respective features from a time series using the method proposed by Hyndman, Wang, and Laptev (2015). Then, a clustering algorithm is applied to the extracted feature vector, to obtain the clusters. Once we distinguish the time series based on their feature properties, for each cluster of time series, we build a separate RNN predictive model. We stabilize the variance of the series, and then we handle seasonality by a two-staged approach including deterministic deseasonalization of the series and seasonal lags. The trend is handled by a window normalization technique. Our results show that prior subgrouping of time series is able to improve the performance of the baseline RNN model in many situations.

The rest of the paper is organized as follows. In Section 2, we provide a brief review on the evolution of neural networks in time series forecasting and an overview of time series clustering approaches. In Section 3, we discuss the proposed methodology in detail. Section 4 presents the experimental setup and the results, and Section 5 concludes the paper.

2. Related work

In the following, we discuss related work in the areas of forecasting with NNs and clustering methods for time series.

2.1. Forecasting with neural networks

The powerful data-driven self-adaptability and model generalizability enable NNs to uncover complex relationships among samples and perform predictions on new observations of a population, without being constrained by assumptions regarding the underlying data generating process of a dataset. These promising characteristics are further strengthened by the universal function approximation properties that NNs possess (Cybenko, 1989; Hornik, 1991). In pure univariate time series forecasting, over the past two decades, NN architectures have been advocated as a strong alternative to traditional statistical forecasting methods (Zhang et al., 1998).

However, over the past two decades, numerous advances have been developed on the way to uncover the true potential of NNs for time series forecasting. Recent developments have been mainly around preprocessing techniques such as deseasonalization and detrending to supplement the NN's learning process, and novel NN architectures such as recurrent neural networks (RNN), echo state networks (ESN), generalized regression neural networks (GRNN) and ensemble architectures to uplift the constraints of the conventional NN architecture (Ilies, Jaeger, Kosuchinas, Rincon, & others, 2007; Nelson, Hill, Remus, & O'Connor, 1999; Yan, 2012; Zhang & Qi, 2005; Zimmermann et al., 2012). Also, careful selection of network parameters with the right choice of model architectures have proven that now NNs can be a strong alternative to traditional statistical forecasting methods (Crone et al., 2011; Zhang et al., 1998).

Researchers have been increasingly drawing their interest towards developing and applying different NN models for time series forecasting. This includes multi-layer perceptrons (MLP), GRNNs, ensemble architectures, RNNs, ESNs and LSTMs, while MLPs are being the most widely used NN variant for time series forecasting thus far. For a detailed description of the MLP architecture and its widespread applications employed in time series forecasting see Zhang et al. (1998).

More recently, RNN architectures are increasingly gaining interest in the time series forecasting community (Fei & Yeung, 2015; Lipton et al., 2015; Zimmermann et al., 2012), as they have properties that make them suitable for forecasting. We provide a detailed overview of RNN algorithms in Section 3.2 and discuss their superior suitability over conventional MLP architectures to model sequential data such as time series. For example, an ESN (Ilies et al., 2007), a special variant of RNN, was able to perform best among AI contenders in the NN3 competition (Crone et al., 2011), and LSTMs, another popular form of RNNs that are specifically introduced to alleviate the limitations of vanilla RNNs, have been successfully ap-

plied in several time series forecasting applications (Duan, Lv, & Wang, 2016; Lee, Xie, Gallagher, Zhang, & Tu, 2015).

However, despite the architectural suitability and increasing developments in the use of RNN architectures for time series forecasting, many existing studies exhibit design weaknesses, so that the forecasting community remains hesitant. This includes lack of empirical evidence and absence of evaluation metrics and standard benchmarks that are widely accepted in the forecasting community (Armstrong, 2006).

2.2. Time series clustering

We distinguish three main approaches to time series clustering (Warren Liao, 2005), namely algorithms that work directly with distances on raw data points (distance-based), indirectly with features extracted from the raw data (feature-based), or indirectly with models built from the raw data (model-based).

The performance of distance-based clustering approaches depends greatly on the particular distance metric used. Defining an adequate distance measure for raw time series can be a challenging task, and has to consider noise, different length of the series, different dynamics, different scales, etc. Also, many such measures mostly focus on the shape of the respective time series (Aghabozorgi, Seyed Shirkhorshidi, & Ying Wah, 2015), and they may yield unintuitive results and limited interpretability (Wang, Smith, & Hyndman, 2006).

Therefore, we focus on feature-based clustering techniques, which, instead of capturing similarity of point values using a distance metric, use sets of global features obtained from a time series to summarize and describe the salient information of the time series. Feature-based approaches can be more interpretable and more resilient to missing and noisy data (Wang et al., 2006). The feature-based clustering is comprised of two stages, namely a feature extraction phase and the clustering phase, for which standard clustering approaches can be used.

In terms of feature extraction, there is a lot of work present in the literature investigating the use of features of a time series as a data-mining tool for extracting useful patterns (Fulcher & Jones, 2014; Mörchen, 2003; Wang et al., 2006). As a large amount of features seems not practical for our purpose, and limiting the amount of features is desirable, in our proposed framework we use a set of self-describable features proposed by Hyndman et al. (2015) to obtain a meaningful division of clusters. These suggested features are designed to capture the majority of the dynamics that can be observed in time series common in many application cases, such as trends, seasonality, autocorrelation, etc. Table 1 summarizes the respective feature vector that is extracted from an individual time series. In our work, we use the implementation available in R, in the `tsmeasures` function from the `anomalous-acm` package (Hyndman et al., 2015).

The feature extraction phase is then followed by a clustering phase that discovers the optimal grouping between the time series by applying a conventional clustering algorithm to the extracted feature vector. Again, a host of different clustering methods exist, an overview gives, e.g., Berkhin (2006). In this study, we use several clustering techniques to assess the robustness of our framework. This includes K-Means (Lloyd, 1982), Partition Around Medoids (PAM) (Kaufman & Rousseeuw, 1990), DBSCAN (Ester, Kriegel, Sander, & Xu, 1996) and Snob (Wallace & Dowe, 1994; 2000), which we discuss in detail in Section 3.4.

3. Methods

In this section, we first present the basic overview of our proposed architecture, and then discuss each component of our forecasting framework in detail.

Table 1
Summary of features extracted from a time series, following Hyndman et al. (2015).

Feature	Description
Mean	Mean
Var	Variance
ACF1	First order of autocorrelation
Trend	Strength of trend
Linearity	Strength of linearity
Curvature	Strength of curvature
Season	Strength of seasonality
Peak	Strength of peaks
Trough	Strength of trough
Entropy	Spectral entropy
Lumpiness	Changing variance in remainder
Spikiness	Strength of spikiness
Lshift	Level shift using rolling window
Vchange	Variance change
Fspots	Flat spots using disretization
Cpoints	The number of crossing points
KLscore	Kullback-Leibler score
Change.idx	Index of the maximum KL score

3.1. The overall procedure

A summarizing scheme of the forecasting framework is given in Algorithm 1. If a partition of the time series is available in the form of additional knowledge, this partition can be considered.

As stated in Section 2.2, at first we use the "anomalous" feature extraction method from Hyndman et al. (2015). Then, a clustering algorithm is applied to the feature vector, to find the optimal grouping of time series. After discovering the clusters, for each cluster of time series, the following pre-processing steps are applied to generate input data for the RNN training.

First, we stabilize the variance, using a log transformation. Then, the series is decomposed into trend, seasonal part, and remainder using the `stl` function from the `forecast` package, in a deterministic setting. Afterwards, a rolling window approach, along with a local normalization technique, is applied to the sum of trend and reminder, to generate the training data. Thereafter, for each cluster, a separate LSTM model is trained, and used for prediction. In the later sections, we introduce these techniques in detail and justify the appropriateness of the above considerations for our framework.

Fig. 1 gives an illustration of the proposed forecasting framework. The overall model is comprised of three components, namely: 1) The pre-processing layer which consists of a clustering phase and a log-transformation, deseasonalization, and a nor-

Algorithm 1 Generating target input files for the RNN.

```
procedure PREPROCESSING(ts, freq, input.win, output.win)
    ts.features ← anomalous(ts, freq)
    ts.clusters ← clusterAlgo(ts.features)
    for i : len(ts.clusters) do
        ts.log ← log(ts)
        [trend, seasonal, remainder] ← stl(ts.log, freq)
        ts.deseason ← [trend, remainder]
        for i : [tsLength(ts.deseason)-output.win-1] do
            window.frame[i] ← rollWindow(ts.deseason, input.win, output.win)
            normalize.series[i] ← normalize(window.frame[i])
            [input, output] ← get(normalize.series[i])
        end for
    end for
    return ts.series[input, output]
end procedure
```

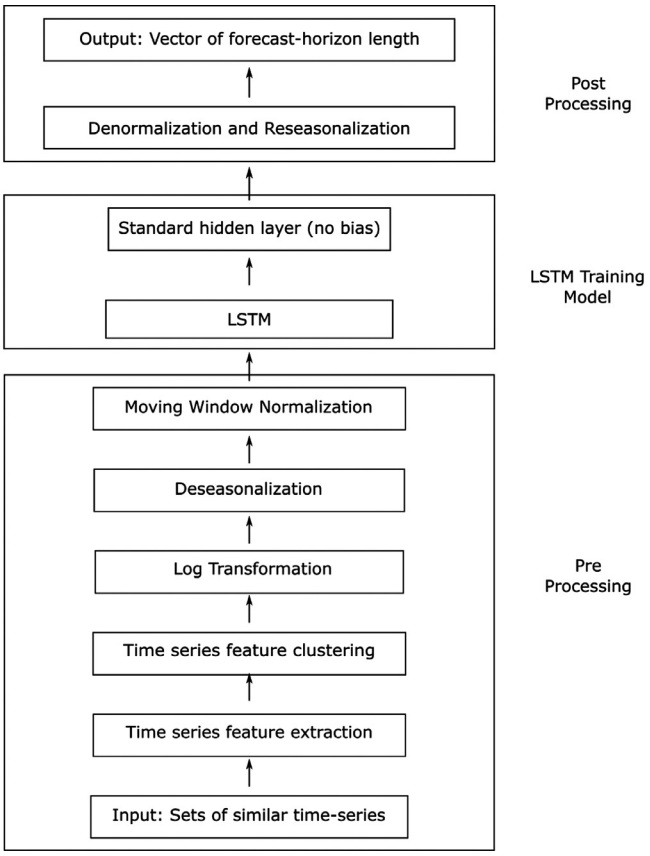

Fig. 1. Proposed network architecture, which includes a pre-processing layer, LSTM training layer and a post-processing layer.

malization phase, 2) the LSTM training layer which consists of an LSTM layer, followed by an affine neural layer (a fully connected layer), excluding the bias component, and 3) a post-processing layer which consists of a denormalization and a reseasonalization phase to ascertain the final forecasts. The reseasonalization process includes introducing the last seasonal component to the generated forecasts. Whereas during the denormalization, the generated forecasts are back-transformed to their original scale, by adding the corresponding trend value obtained from the local normalization process, i.e, adding the last value of the trend inside an input window, and taking the exponent of the values.

In the following sections, we discuss the key design aspects of the components of our framework. This includes a brief introduction to our base algorithm, i.e., to RNNs and LSTMs. Afterwards, we explain the time series clustering method that is utilized to group sets of similar time series in the absence of other groupings. Finally, we discuss the time series preprocessing techniques used in our forecasting framework. This includes variance stabilization, deseasonalization, and a local normalization in a sliding window approach that structures the training data.

3.2. Recurrent neural networks

RNNs are a special type of NNs that are suited to model sequences of variable lengths (Elman, 1990). In addition to the standard input and output layer, RNNs contain special hidden layers that are composed of recurrently connected nodes. These hidden layers are commonly referred to as memory states, as they enable the networks to preserve the sequential information and persist the knowledge acquired from subsequent time steps. The past information is retained through a feedback loop topology, where as a part of the input of the current step, the RNN uses the output of the previous time step during the network training. In effect, this recurrent model enables the network to take the previous values into account. There is a whole family of RNN design patterns available to model sequences, which can be distinguished by the type of recurrent architecture they use, i.e., recurrent connections between hidden units, or recurrent connections only from the output to the hidden units. For more information on RNN design patters, we refer to Goodfellow, Bengio, Courville, and Bengio (2016).

Fig. 2 shows an example of an RNN unfolded in time, which unrolls the feedback loop to expand the complete sequence of the NN in time. The matrices $U \in \mathbb{R}^{m\times n}$, $W \in \mathbb{R}^{n\times n}$, and $V \in \mathbb{R}^{n\times k}$ define the shared weights of the RNN, where the inputs to the hidden connections are parameterized by the weight matrix U, hidden-to-hidden recurrent connections are parameterized by the weight matrix W, and hidden-to-output connections are parameterized by the weight matrix V. Moreover, n, m, and k represent the sizes of state, input, and output vectors, respectively. In this model, $x_t \in \mathbb{R}^m$ denotes the input at time step t, $h_t \in \mathbb{R}^n$ denotes the hidden state at time step t, $h_{t-1} \in \mathbb{R}^n$ denotes the prior state at time step $(t-1)$, and $o_t \in \mathbb{R}^k$ denotes the output at time step t. Note that h_t represents the "memory" of the network at time step t, which is computed based on the current input x_t and the previous hidden state h_{t-1} at time step $(t-1)$. In other words, this is the overall knowledge and reasoning accumulated by the network based on the previous data. The hidden state h_t and output o_t at time step t can be formally defined as follows:

$$h_t = f_\theta(Ux_t + Wh_{t-1})$$
$$o_t = f_\alpha(Vh_t)$$

Generally, f_θ and f_α are non-linear functions such as, e.g., *tanh*, or *rectified linear units* (ReLu; Le, Jaitly, & Hinton, 2015). RNNs often use backpropagation through time (BPTT; Williams & Zipser, 1995) as the learning algorithm. This is an extension of the backpropagation algorithm, which unrolls the network through time to propagate the error sequentially.

Even though RNN architectures are quite capable of capturing short-term dependencies in sequential data, they often have difficulties in learning long-term dependencies from distant past information. This is caused by the *vanishing gradient problem* (Bengio, Simard, & Frasconi, 1994; Hochreiter, 1991), which is a well-known constraint in gradient based learning algorithms. Generally, gradient-based learning techniques determine the influence of a given input, based on the sensitivity of network parameters on the output. For example, as the length of a sequence grows, the corresponding error gradient is propagated through the network many steps. As a result, the gradient decays exponentially as it progresses through the chain, leaving a small impact on the output from the initial elements of the sequence. This decreases the information retention ability of RNNs, while failing to capture the potential impact from initial inputs to the network output.

3.3. Long short-term memory networks

LSTM networks were introduced by Hochreiter and Schmidhuber (1997), to address the long term memory shortage of vanilla RNNs. The LSTM extends the RNN architecture with a standalone memory cell and a gating mechanism that regulates the information flow across the network. The gating mechanism is equipped with three units, namely: input, forget, and output gate. This mechanism cohesively determines which information to be persisted, how long it is to be persisted, and when it is to be read from the memory cell.

As a result, LSTMs are capable of retaining key information of input signals, and ignore less important parts. This memory cell has a recurrently self-connected linear unit called "Constant Error

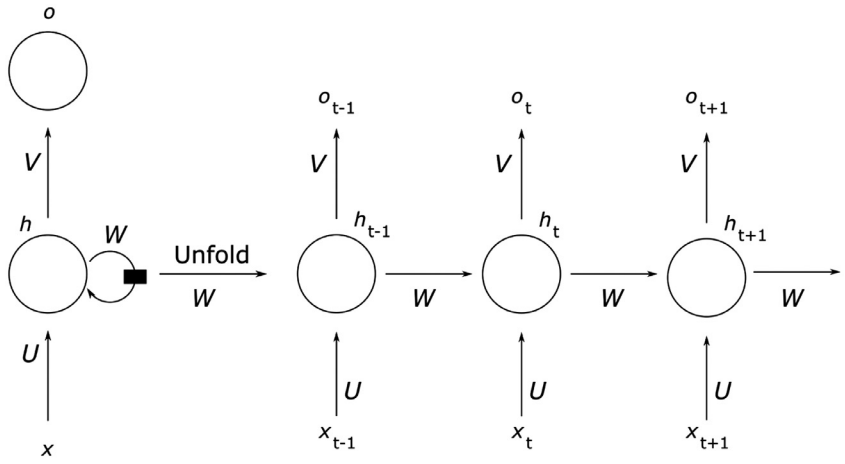

Fig. 2. An unrolled recurrent neural network in time, with the shared weights of *U, V*, and *W*.

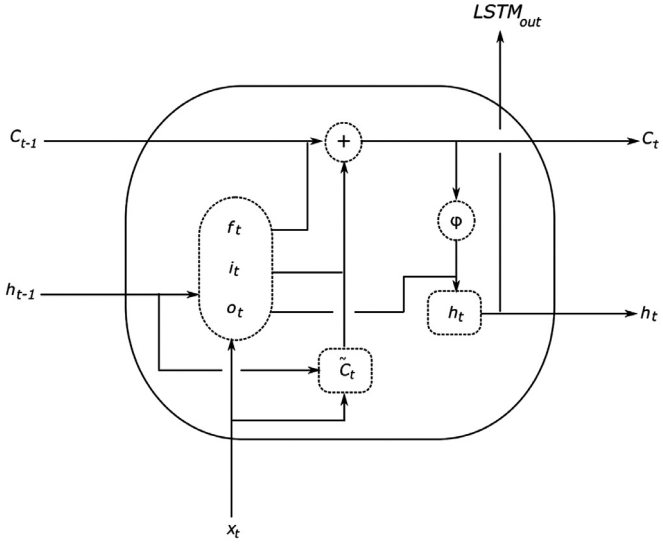

Fig. 3. Basic architecture of an LSTM memory block with three gated layers: forget gate f_t, input gate i_t, and output gate o_t, controlling the activation of cells C_{t-1} and C_t.

Carousel" (CEC), which contains a state vector (C_t) that allows to preserve dependencies for the long-term. Consequently, in contrast to vanilla RNNs, LSTMs preserve information and propagate errors for a much longer chain in the network, and overcome the vanishing gradient problem. In fact, LSTMs possess the ability of remembering over 100 steps of a sequence (Längkvist, Karlsson, & Loutfi, 2014). Fig. 3 illustrates the basic structure of an LSTM memory block with a one cell architecture (following R2RT Blog, 2016). In the figure, $x_t \in \mathbb{R}^m$ denotes the input at time step t, $C_{t-1} \in \mathbb{R}^n$ and $C_t \in \mathbb{R}^n$ denote the cell state at time steps t and $(t-1)$, while $h_t \in \mathbb{R}^n$ and $h_{t-1} \in \mathbb{R}^n$ correspond to the output at time steps t and $(t-1)$, respectively. Here, m represents the size of input vector, while n denotes the size of the memory cell. In general, the short-term memory of LSTMs is provided by the h_t state, while the C_t state enables to retain long-term dependencies.

Also, to distinguish the self-contained memory cell CEC of the LSTM from the conventional state (h_t), we refer to it as C_t. The forget gate f_t takes x_t and h_{t-1} as inputs to determine which information to be retained in C_{t-1}. The gate activation functions i_t, o_t, and f_t are usually sigmoid layers, so that the output is projected to a value between zero and one for each value in C_{t-1}, describing the scale of information retention, i.e., a "zero" output represents the complete expunge of a value from the memory cell, while a "one" represents the complete retention of that value in the memory cell. Meanwhile, the input gate i_t is accompanied with a sigmoid layer that uses x_t and h_{t-1} to ascertain the values to be summed by addition to C_t. Additionally, a non-linear layer ϕ (e.g., *tanh*) is also introduced to generate a vector of candidate values, denoted as $\tilde{C}_t$, to update the state of C_t. The output gate o_t regulates the output values of an LSTM cell, based on the updated state of C_t. Likewise, as in the forget and input gates, the output gate is a sigmoid layer to filter the output. Correspondingly, the updated cell state C_t is fed into a *tanh* layer (ϕ), which scales down the vector to a value between (-1) and $(+1)$. This is then multiplied element-wise by the output of the sigmoid layer to compute the final cell output h_t at time step t. The aforementioned process can be formally defined by the following recursive equations:

$$
\begin{aligned}
i_t &= \sigma\left(W_i \cdot h_{t-1} + U_i \cdot x_t + b_i\right) \\
o_t &= \sigma\left(W_o \cdot h_{t-1} + U_o \cdot x_t + b_o\right) \\
f_t &= \sigma\left(W_f \cdot h_{t-1} + U_f \cdot x_t + b_f\right) \\
\tilde{C}_t &= \phi\left(W_c \cdot h_{t-1} + U_c \cdot x_t + b_c\right) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\
h_t &= o_t \odot \phi(C_t) \\
LSTM_{out} &= h_t
\end{aligned}
$$

Here, $(W_i, W_o, W_f, W_c) \in \mathbb{R}^{n \times n}$ represent the weight matrices of forget gate, input gate, memory cell state, and output gates respectively. Also, $(U_i, U_o, U_f, U_c) \in \mathbb{R}^{m \times n}$ denote the corresponding input weight matrices. Biases of the respective gates are $(b_i, b_o, b_f, b_c) \in \mathbb{R}^n$, while $\odot$ denotes the element-wise multiplication operation. In these equations, σ represents the standard logistic sigmoid activation function, and ϕ stands for the hyperbolic tangent function, i.e., *tanh*. These are defined by the following equations respectively:

$$
\begin{aligned}
\sigma(x) &= \frac{1}{1+e^{-x}} \\
\phi(x) &= \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}
\end{aligned}
$$

Several variants to the originally proposed algorithm can be found in the literature. E.g., LSTM with "peephole connections," introduced by Gers, Schmidhuber, and Cummins (2000), is one of the popular variants that allows LSTM gates to examine the state of their memory cell, before updating their states. Let the peephole weight matrices of input, output and forget gates be defined as $(P_i, P_f, P_o) \in \mathbb{R}^{n \times n}$. Then, the equations of the basic LSTM can

be rewritten as follows:

$$
\begin{aligned}
i_t &= \sigma(W_i \cdot h_{t-1} + U_i \cdot x_t + P_i \cdot C_{t-1} + b_i) \\
f_t &= \sigma(W_f \cdot h_{t-1} + U_f \cdot x_t + P_f \cdot C_{t-1} + b_f) \\
\tilde{C_t} &= \phi(W_c \cdot h_{t-1} + U_c \cdot x_t + b_c) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C_t} \\
o_t &= \sigma(W_o \cdot h_{t-1} + U_o \cdot x_t + P_o \cdot C_t + b_o) \\
h_t &= o_t \odot \phi(C_t) \\
LSTM_{out}^{peephole} &= h_t
\end{aligned}
$$

Since LSTM is an RNN variant, the sequence of the original time series is relevant and needs to be preserved during training. All training patches relevant to a particular time series are read as one sequence. Therefore, the LSTM state needs to be initialized for each series. Typically a vector of zeros is used, but there are other possibilities. As a result of this state transition, the trained LSTM network with a set of fixed weight vectors can still show different predictive behaviour for different time series, i.e., in a trained network, a particular time series is represented by the composition of weight vector and its corresponding internal state (Prokhorov, Feldkarnp, & Tyukin, 2002; Smyl & Kuber, 2016).

In this study, we use the Microsoft Cognitive Toolkit (CNTK), an open-source NN toolkit (Seide & Agarwal, 2016), to implement the LSTM. As our base learning algorithm, we use an LSTM with peephole connections, which is followed by an affine neural layer (excluding the bias component) to project the LSTM cell output to the dimension of the intended forecast horizon, i.e., the dimension of this fully connected neural layer equals the size of the output window. We use L2-norm as our primary loss function to train the LSTM, which essentially minimizes the sum of squared differences between the target values and the estimated values.

3.4. Clustering techniques

As highlighted in Section 2.2, we apply K-Means, PAM, DBSCAN and Snob as clustering techniques to the extracted feature vector to determine the optimal subsets of time series.

K-Means (Lloyd, 1982) and PAM (Kaufman & Rousseeuw, 1990) are partitioning based clustering algorithms that divide the dataset into a set of disjoint partitions, in which each partition represents a cluster. Subsequently, the Elbow method and Silhouette scores are used (Kodinariya & Makwana, 2013) to determine the optimal number of clusters of these methods. We use the `cluster` package (Maechler, Rousseeuw, Struyf, Hubert, & Hornik, 2019) along with the `factoextra` package (Kassambara & Mundt, 2017) in R.

Furthermore, we use DBSCAN (Ester et al., 1996), a density based clustering technique that is capable of discovering clusters of arbitrary shapes. An implementation of DBSCAN from the `dbscan` package (Hahsler & Piekenbrock, 2018) in R is used in our experiments. We also introduce an implementation of the Snob clustering algorithm, which is a mixture model algorithm based on the Minimum Message Length (MML) concept that accounts for the highest posterior probability distribution of each cluster (Wallace & Dowe, 1994; 2000). Unlike in the K-Means and PAM clustering algorithms, DBSCAN and Snob are able to determine an optimal number of clusters in the dataset autonomously.

3.5. Moving window approach

The proposed moving window approach follows the Multi-Input Multi-Output (MIMO) strategy of forecasting that models a multiple input and output mapping, while preserving the stochastic dependencies between predicted values. While RNNs can be operated with one input at a time, the internal state of the network then needs to memorize all relevant information. Using an input window relaxes this requirement and allows the network to also operate directly with lagged values as inputs. Furthermore, Ben Taieb, Bontempi, Atiya, and Sorjamaa (2011) discuss the benefits of applying a MIMO strategy over a single-output forecasting strategy (recursive strategy) in multi-step forecasting. There, those authors highlight that accuracy of the latter approach is affected by its recursive nature, and errors are accumulated at each forecasting step. The MIMO strategy is advocated by recent studies for multi-step forecasting with NNs (Petersen, Rodrigues, & Pereira, 2019).

In this work, we apply the moving window method to a fully preprocesses time series as discussed in the following sections, as follows. At first, a time series of length *tsLength* is converted to patches of length (*outputSize+inputSize*). In total, there are (*tsLength–outputSize–inputSize*) such patches. Here, *outputSize* refers to the length of the output window (i.e., the intended forecasting horizon), while *inputSize* represents the length of the input window used in each frame. Fig. 4 illustrates the procedure with an example of applying the moving window approach to series TS59 of the CIF2016 dataset. The training dataset is generated by iterating the above process until the last point of the input window is positioned at (*tsLength-outputSize-1*), i.e., the last output window of the series is reserved for validation and not used for training. For the validation, forecasts for this last output window are produced. Due to the recursive nature of the process, also for the validation we need to iterate through the whole time series (called the “warm-up” in this case), analogous to the training phase.

As to concrete choices of the size of input and output windows, the output window size is largely determined by the required forecast horizon. As a heuristic, we then currently choose the input window size in relation to the output window size and the seasonality. In particular, $inputSize = 1.25 \cdot \max(outputSize, seasonal\_period)$. That is, we choose the input window larger (with a factor of 1.25 in our experiments) than the output window size or the length of the seasonal period, whichever is larger. This choice is rather empirical, and it seems adequate for situations where a full period can be captured by relatively few data points. In situations where, e.g., a daily series with a yearly seasonality is to be forecasted, the method would likely benefit from more sophisticated feature or lag selection. Also, in situations where time series are very short, a smaller input window size may need to be chosen.

3.6. Modelling seasonality

Early studies suggest that NNs are suitable to effectively model the underlying seasonality and cyclical patterns in time series due to their universal function approximation properties, i.e., their capacity to estimate linear and non-linear functions (Marseguerra, Minoggio, Rossi, & Zio, 1992; Zaiyong Tang, de Almeida, & Fishwick, 1991). However, more recently several studies argue that deseasonalizing data prior to modelling is necessary to produce accurate forecasts. In particular, Nelson et al. (1999) compare the forecasts generated from NNs trained with deseasonalized data and non-deseasonalized data, using 68 monthly time series from the M-competition (Makridakis et al., 1982). The results indicate that the NN trained with prior deseasonalization achieves better forecasting accuracy, in contrast to NNs trained with non-deseasonalized data. Similarly, using the NN5 competition data, Ben Taieb et al. (2011) empirically show that the resulting forecasts benefit from prior deseasonalization of the data. Zhang and Qi (2005) demonstrate that NNs are not capable of effectively modelling trend or seasonality directly, and emphasize that the forecasting errors can be reduced by detrending or deseasonalization of the raw time series. These findings are in line with our discussions in Section 1 of data availability, model complexity and prior assumptions. Though NNs may be able to model a signal in

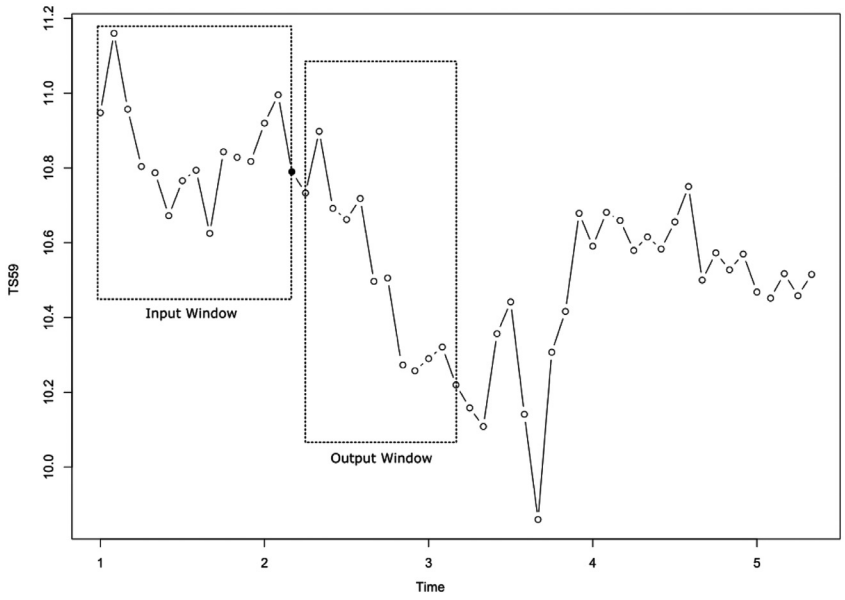

Fig. 4. An example of applying the moving window approach to the series TS59 of the CIF2016 dataset. The filled dot marks the last value of the input window.

principle, this is no guarantee that in practice they will do it accurately.

As our method is intended to run also especially in situations where the overall amount of data is limited, we use deseasonalization techniques to remove deterministic seasonality, which may be unnecessarily difficult to learn for the RNN. Then, inclusion of seasonal lags in our rolling window procedure allows the RNN to capture the remaining stochastic seasonality. This approach is inspired by the well-known "boosting" ensemble technique (Schapire, 2003), where STL deseasonalization can be seen as a weak base learner that is subsequently supplemented by the RNN. We use seasonal and trend decomposition using loess (STL), as proposed by Cleveland, Cleveland, and Terpenning (1990), which is considered a robust method to decompose a time series into trend, seasonal, and remainder components. It consists of a sequence of applications of a loess smoother, making the decomposition computationally efficient, even for longer time series (Cleveland et al., 1990; Hyndman & Athanasopoulos, 2014). We use STL to extract a deterministic seasonality from the series after variance stabilization (see Section 3.7), and pass on the sum of trend and reminder to the next step of data preprocessing. In R, STL is implemented in the `stl` function from the `forecast` package (Hyndman et al., 2015; Khandakar & Hyndman, 2008). To obtain deterministic seasonality, we set the `s.window` parameter to "periodic". This parameter controls the smoothness of the change of seasonal components, and setting it to "periodic" enforces that no change in the components is possible, so that the result is a deterministic seasonality, where all seasonal periods are identical.

We currently apply the deseasonalization procedure to all time series, regardless of whether seasonality is present in the data or not. The only rather basic test we perform for seasonality is that if series contain less than two full seasonal periods, the STL procedure will not be applicable, and then we will assume the series is non-seasonal and consequently we will not deseasonalize the series. In all other cases, we extract a deterministic seasonality. The assumption we currently have is that even if the deseasonalization procedure extracts a seasonal component that is not actually present in the data, this component will be small and can easily be compensated for by the second phase of the seasonality modelling, i.e., the seasonal lags of the RNN. In any way, modelling such a small artificial seasonal component will be easier than modelling of all the deterministic seasonality in the dataset by the RNN, without any preprocessing.

Other possibilities to address this issue would be to determine seasonality with, e.g., a statistical test, and apply deseasonalization only to series where seasonality is detected. However, such tests are usually either based on the Autocorrelation Function (ACF) or they are model-based (Hans Franses, 1992), and therewith they have their own assumptions and shortcomings, and could cause problems in our non-linear, non-parametric setup. Furthermore, following the argumentation of Hyndman (2014), we are not interested in uncovering an underlying data generating process, but we are concerned in improving forecasting accuracy.

Though we restrain from modelling seasonality from only part of the series, depending on the data it is possible to imagine situations where modelling the entire seasonality by the RNN will be superior to the proposed process. Performance of the current process depends on the capability of the deterministic deseasonalization procedure to extract seasonality from a single series. Accordingly, if the dataset is such that seasonality is too noisy in the single series, or the series are too short, the proposed procedure will not be applicable, and it may be beneficial to then omit the seasonality preprocessing alltogether. Also, the deseasonalization is less likely needed when we face a number of time series with calendar features (e.g., day of week) and/or homogeneous seasonality patterns. If time stamps are known and the series are related, e.g. describe electric load of subsections of a regions grid, the RNN will be more likely to learn the seasonality. On the other hand, if there are different and/or unknown seasonalities across the series, it is better to deseasonalize first. In line with the discussions above, best practice would then be to fit both a model with and without deseasonalization, and then choose the better of the two models using a validation set.

3.7. Variance stabilization using power transformations

Some works in the literature argue that, though NNs have universal approximation capabilities (Hornik, 1991), power transformations may make the NN learning procedure difficult by altering the original non-linearities in a time series (Faraway & Chatfield, 2008). However, we stabilize the variance in our data for two reasons. As the STL decomposition method that we employ for sea-

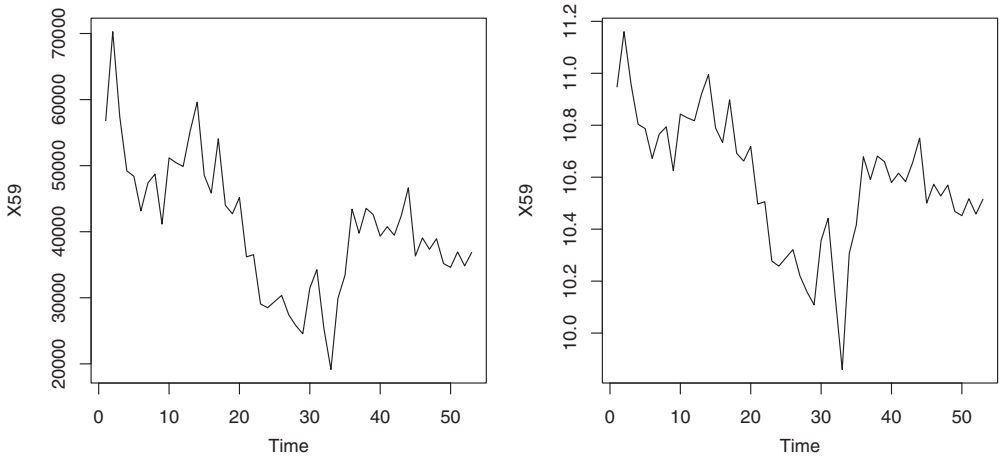

Fig. 5. Series TS59 of the CIF2016 dataset, which is a monthly time series. On left is the original series, right the log transformed version.

sonality extraction is an additive method, we need to ensure that seasonality is additive. Furthermore, we model the trend in a conservative way (see Section 3.8), and stabilizing the variance enables us to model greater dynamics in the trend.

Popular transformations for variance stabilization are, e.g., the Box–Cox transform (Box & Cox, 1964) or the log transform. The log transform is a strongly non-linear transform and is usually used with caution, as small differences in log space may result in large differences in the original space, and therewith model fitting can yield sub-optimal results. The Box-Cox transform is a popular more conservative approach. It is usually defined as follows:

$$w_t = \begin{cases} \log(y_t), & \lambda = 0; \\ (y_t^{\lambda} - 1)/\lambda, & \lambda \neq 0. \end{cases}$$

We see that the transform resembles, depending on its parameter λ, the logarithm or the identity in its most extreme cases ($\lambda = 0$ or $\lambda = 1$, respectively). A difficulty with this transform is the choice of λ. The only procedure we are aware of to choose λ automatically is the procedure of Guerrero (1993). However, in preliminary experiments we found that this procedure has its shortcomings and the parameter λ is difficult to choose in practice, and that in fact the logarithm seems to be the better choice for our proposed method. Therefore, in our experiments, we use the log transformation to transform each time series to a logarithmic scale before it is fed into the STL algorithm. Finally, in the post-processing stage, the corresponding forecasts are back-transformed into the original scale of the time series, by taking the exponent of each generated output value. To avoid problems for zero values, we use the logarithm in the following way:

$$w_t = \begin{cases} \log(y_t), & min(y) > \epsilon; \\ \log(y_t + 1), & min(y) \leq \epsilon; \end{cases}$$

Here, y denotes a time series, and ϵ can be chosen as zero for integer time series, or a small value close to zero for real-valued time series. As an example, Fig. 5 shows the original series and the log transformed version of Series TS59 of the CIF2016 dataset.

3.8. Modelling trend

Within the rolling window processing, the last value of the trend in each input window, provided by STL (illustrated by the filled dot in Fig. 4) is used for local normalization. The trend component extracted by the STL procedure of that last value is subtracted from each data point in the corresponding input and output window. This process is applied to each input and output window combination separately.

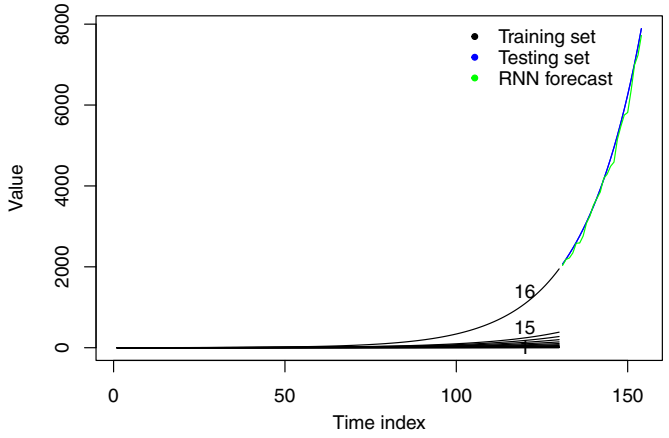

Fig. 6. Set of 16 synthetically generated time series that exhibit exponential trends. Each time series contains 130 data points for training and 24 data points for testing. The steepness of the trend in each time series is elevated gradually. We see that our method is able to predict the exponential trend of the steepest time series accurately, even though the values predicted are outside the range of the training data. We note that the prediction is performed with a single output window.

Modelling the trend in this way has various advantages over extracting the trend using STL and modelling it separately. In contrast to prediction of a deterministic seasonality, predicting forward the extracted trend of a time series is not trivial, so if we extracted the trend and predicted it separately, we would effectively face another non-trivial prediction problem. Instead, we use the RNN directly to predict the trend.

As the RNN will eventually saturate, the predicted trend is limited by the bounds of the transfer function (Ord, Fildes, & Kourentzes, 2017). However, the local normalization step employed makes the network not saturate based on absolute values, but effectively it limits the steepness of the trend to the maximal steepness found in all training windows (after variance stabilization, deseasonalization). This leads to rather conservative forecasts and seems a reasonable assumption in a time series prediction scenario, where predicted exponential trends are often the source of potentially large forecasting errors.

To further illustrate the capability of our procedure to model trends, in Fig. 6 we report the results of a brief experiment with simulated data. We see that in practice the procedure of variance stabilization and local normalization is able to model even fast-growing, exponential trends.

4. Experimental study

In this section, we evaluate the proposed procedure on two benchmark datasets from past forecasting competitions, namely the CIF2016 and NN5 datasets. We describe the forecasting methods and error measures used to perform the experiments, and the results obtained.

4.1. Benchmarking datasets from forecasting competitions

We use the publicly available datasets from the CIF2016 and NN5 forecasting competitions. These competitions were specifically organized to evaluate and compare the potential of ML techniques in handling large scale ex-ante forecasting. In fact, each dataset is comprised of similar time series, related to a certain domain. This is the main basis of using these specific public datasets, as they comply with our original hypothesis of exploiting the advantages of similar time series (unlike, e.g., the M3 competition data).

The CIF2016 competition dataset consists of monthly time series, composed of two different subgroups: series related to the banking industry and artificially generated series (Štěpnička & Burda, 2016). Specifically, contestants were requested to submit 12-months-ahead forecasts for 57 time series, and 6-months-ahead forecasts for 15 time series, so for a total of 72 series. The CIF2016 competition attracted participants from numerous fields of Computational Intelligence (CI), such as artificial NNs, fuzzy methods, support vector machines, decision and regression trees, etc.

The NN5 competition dataset contains 2 years of daily cash withdrawals at various automatic teller machines (ATMs) located in the UK (Crone, 2008). In detail, 111 time series of ATMs were made available during the competition, and the participants were asked to submit the forecasts for a prediction horizon of 56 days ahead. Moreover, the NN5 competition includes various challenges of a real-world forecasting task, such as multi-step ahead forecasting, outliers, missing values. A variety of time series imputations techniques are available to replace the missing observations in a time series (Andiojaya & Demirhan, 2019; Moritz, Sardá, Bartz-Beielstein, Zaefferer, & Stork, 2015). For the simplicity, we impute the missing values of the NN5 dataset by using the median cash withdrawals on each day of the week, assuming time series follow a weekly seasonality, i.e., frequency =7. Similar to the CIF2016, a variety of CI solutions were presented at the competition.

However, while primary means of ranking forecasting approaches of these competitions was among CI techniques, statistical and hybrid techniques were also permitted to submit their forecasting solutions. Therefore, we can straightforwardly evaluate our proposed approach against state-of-the-art statistical benchmarks such as ETS, ARIMA, Theta, etc.

4.2. Error measures

To compare our proposed approach against the participants' forecasts of CIF2016 and NN5, we calculate the symmetric Mean Absolute Percentage Error (sMAPE) for every series, and then calculate an average over all the series to obtain a mean sMAPE, which is consistent with the evaluations in the competitions. To get a better understanding of the overall error distribution of the forecasts, and positioning of the forecast methods across all the time series, we furthermore calculate the median and rank sMAPE over all series.

Furthermore, a host of different choices to evaluate forecasts exists in the forecasting literature, overviews and good discussions give Hyndman and Koehler (2006) and Davydenko and Fildes (2013). Many popular measures such as the sMAPE have known shortcomings, such that they are skewed, lack of robustness, lack of interpretability, or are unstable with values close to zero in the original data and/or the forecasts. To address some of these issues, as proposed in Hyndman and Koehler (2006), we use the Mean Absolute Scaled Error (MASE) as our second evaluation metric. Unlike sMAPE, MASE is a scale independent error measure that calculates the forecast errors independent of the scale of the time series. Moreover, MASE offers interpretability, as it measures the forecasting accuracy relative to the average one-step naïve forecast error, or to the naïve seasonal forecast error, if the time series is seasonal. That is, assuming the time series are seasonal, MASE $<$ 1 means that on average the method performs better than the naïve seasonal forecast computed on the training data, while MASE $>$ 1 indicates that the method performs worse. We use the sMAPE and the MASE in the following definitions:

$$\text{sMAPE} = \frac{200}{h}\sum_{t=1}^{h}\left(\frac{|F_t - Y_t|}{|F_t| + |Y_t|}\right)$$

$$\text{MASE} = \frac{1}{h}\frac{\sum_{t=1}^{h}|F_t - Y_t|}{\frac{1}{n-M}\sum_{t=M+1}^{n}|Y_t - Y_{t-M}|}$$

Here, Y_t denotes the observation at time t, F_t is the respective forecast. Furthermore, h denotes the number of data points in the test set and n denotes the number of data points in the training set of a time series. The seasonal period of a time series is represented by M. In particular, we provide the following evaluation measures based on these primary error measures: Mean of the sMAPEs (Mean sMAPE), Median of the sMAPEs (Median sMAPE), Mean of the sMAPE ranks of each series (Rank sMAPE), Mean of the MASE (Mean MASE), Median of the MASE (Median MASE), and Mean of the MASE ranks (Rank MASE) of each series.

4.3. Hyperparameter selection and compared methods

The LSTM has various hyper-parameters. We tune these with a Bayesian global optimization methodology (Snoek, Larochelle, & Adams, 2012), which uses Bayesian inference and a Gaussian process to autonomously optimize the hyper-parameters of an unknown function using a validation set. In particular, we use the `BayesianOptimization` function from the `bayesian-optimization` package implemented in Python (Fernando, 2017). Table 2 shows the parameter ranges that are used throughout the experiments.[1]

Table 2
Parameter value ranges used throughout the LSTM learning process, represented by the respective minimum and maximum values. Here, n_{tr} denotes the number of training examples in a training file.

Model parameter	Minimum value	Maximum value
LSTM-cell-dimension	10	80
Epoch-size	(n_{tr})	(n_{tr}*3)
Mini-batch-size	2	40
Learning-rates-per-sample	0.001	0.04
Maximum-epochs	10	40
Gaussian-noise-injection	0.0005	0.005
L2-regularization-weight	0.0005	0.0008

Furthermore, we choose input and output window sizes throughout the experiments according to the discussions in Section 3.5. In particular, for the NN5 dataset, as the forecasting horizon is 56, we use an *inputSize* of 70 and an *outputSize* of 56. As the CIF2016 dataset has two different target horizons and some very short series, choosing window sizes is more complicated here and differs among different models, as outlined in the following.

In our experiments, we use the following variants of our proposed methodology and LSTM baseline:

[1] More information about the respective hyper-parameters can be found on the CNTK web site *Python API for CNTK* https://cntk.ai/pythondocs/cntk.html.

Table 3
Results for the 72 monthly series of CIF2016, ordered by the second column, which is the Mean sMAPE. For each column, the results of the best performing method(s) are marked in boldface.

Method	Mean sMAPE	Median sMAPE	Rank sMAPE	Mean MASE	Median MASE	Rank MASE
LSTM.Cluster.kMeans.Silhouette	**10.52**	7.30	12.34	0.88	0.59	12.17
LSTM.Cluster.Snob	10.53	7.34	13.21	0.89	0.59	13.17
LSTM.Cluster.Dbscan	10.57	7.30	13.13	0.91	0.59	13.08
LSTM.Cluster.kMeans.Elbow	10.60	7.20	12.09	0.89	0.56	12.03
LSTM.Horizon	10.61	7.92	13.19	0.90	0.60	12.60
LSTM.All	10.69	6.72	12.61	0.83	0.59	12.49
LSTMs and ETS	10.83	6.60	**10.88**	**0.79**	0.56	**10.82**
LSTM.Cluster.PAM.Silhouette	10.98	7.41	12.98	0.91	0.57	12.83
ETS	11.87	6.67	12.21	0.84	**0.53**	12.21
MLP	12.13	6.92	12.99	0.84	0.54	13.13
REST	12.45	7.57	14.61	0.90	0.59	14.90
ES	12.73	6.51	13.13	0.87	**0.53**	13.05
FRBE	12.90	6.77	12.78	0.89	0.57	12.78
HEM	13.04	7.32	14.44	0.90	0.59	14.49
Avg	13.05	8.02	16.49	0.99	0.67	16.48
BaggedETS	13.13	**5.98**	11.61	0.83	0.54	11.38
LSTM	13.33	8.20	17.07	0.95	0.68	17.05
Fuzzy c-regression	13.73	10.04	17.46	1.13	0.72	17.46
PB-GRNN	14.50	7.86	16.00	1.01	0.65	16.18
PB-RF	14.50	7.86	16.00	1.01	0.65	16.18
ARIMA	14.56	7.03	14.62	0.92	0.56	14.62
Theta	14.76	11.01	20.57	1.25	0.74	20.73
PB-MLP	14.94	8.05	16.57	0.99	0.68	16.58
TSFIS	15.11	10.18	19.74	1.27	0.91	19.71
Boot.EXPOS	15.25	6.92	14.60	0.93	0.61	14.51
MTSFA	16.51	9.69	18.24	1.13	0.71	18.19
FCDNN	16.62	8.71	20.01	1.14	0.82	20.26
Naïve Seasonal	19.05	12.72	23.12	1.29	0.95	23.46
MSAKAF	20.39	14.24	21.89	1.57	1.31	21.85
HFM	22.39	11.89	22.97	3.27	1.14	23.14
CORN	28.76	19.86	28.44	2.24	1.83	28.42

Table 4
Cluster results for the CIF2016 dataset, columns representing the number of clusters, number of time series for each cluster and the variance associated with each cluster, respectively.

Cluster variant	ClusterID	No. time series	Variance
LSTM.Cluster.kMeans.Silhouette	1	30	0.189
	2	27	0.221
LSTM.Cluster.kMeans.Elbow	1	30	0.189
	2	5	0.166
	3	22	0.185
LSTM.Cluster.Dbscan	1	13	0.126
	2	12	0.164
	3	12	0.155
	4	4	0.102
	5	16	0.287
LSTM.Cluster.Snob	1	12	0.197
	2	12	0.126
	3	18	0.176
	4	6	0.208
	5	9	0.177
LSTM.Cluster.PAM.Silhouette	1	8	0.133
	2	6	0.112
	3	6	0.09
	4	2	0.06
	5	9	0.168
	6	6	0.126
	7	3	0.160
	8	6	0.127
	9	7	0.153
	10	4	0.145

LSTM.Horizon. In the CIF2016 competition, additional knowledge is available in the form of 2 different required forecasting horizons. We use this additional knowledge to group the time series accordingly. I.e., separate prediction models are generated for each group of time horizons, following the steps 5:16 of Algorithm 1. Also, some of the series with a required horizon of 6 are very short and consist only of 23 data points. Following our heuristic, we use an *inputSize* of 7 when the required *outputSize* is 6, and an *inputSize* of 15 when the required *outputSize* is 12.

LSTM.Cluster. Our proposed methodology as illustrated in Algorithm 1. An individual prediction model is produced for each cluster obtained. Due to the peculiarities and short series within the CIF2016 dataset, we start with the same partition as for the LSTM.Horizon model in this case, and then apply the methodology only for the series with a target

Table 5
Summary of computation times of the proposed LSTM variants for the CIF2016 dataset (in minutes).

LSTM variant	Pre-processing	Model training	Post-processing	Total time
LSTM.All	4.4	26.2	0.9	31.5
LSTM.Horizon	4.0	22.2	0.8	27.0
LSTM.Cluster.kMeans.Silhouette	3.6	20.4	0.8	24.8
LSTM.Cluster.kMeans.Elbow	3.4	15.2	0.7	19.3
LSTM.Cluster.Dbscan	3.0	16.2	0.5	19.7
LSTM.Cluster.Snob	3.1	16.4	0.6	20.1
LSTM.Cluster.PAM.Silhouette	2.0	12.2	0.4	14.6

Table 6
Results of statistical testing for the CIF2016 dataset, including original participants' results and our results (printed in boldface). Adjusted p-values calculated from the Friedman test with Hochberg's post-hoc procedure are shown. A horizontal line is used to separate the methods that perform significantly worse than the best method from the ones that do not. We see that the LSTM.All and the proposed LSTM.Cluster variants do not perform significantly worse compared to the best method LSTMs and ETS.

Method	p_{Hoch}
LSTMs and ETS	–
BaggedETS	0.627
ES	0.627
ETS	0.627
MLP	0.627
FRBE	0.627
LSTM.ALL	0.627
LSTM.Cluster.Snob	0.627
LSTM.Cluster.kMeans.Silhouette	0.627
LSTM.Cluster.Dbscan	0.627
LSTM.Cluster.kMeans.Elbow	0.627
LSTM.Cluster.PAM.Silhouette	0.627
LSTM.Horizon	0.627
HEM	0.240
Boot.EXPOS	0.196
REST	0.196
ARIMA	0.196
PB.RF	0.012
PB.GRNN	0.012
Avg	0.004
PB.MLP	0.003
Fuzzy.c.regression	3.07×10^{-4}
LSTM	9.14×10^{-4}
MTSFA	2.73×10^{-5}
TSFIS	1.19×10^{-7}
FCDNN	4.08×10^{-8}
Theta	4.11×10^{-9}
MSAKAF	9.84×10^{-12}
HFM	3.99×10^{-14}
Naïve Seasonal	1.81×10^{-14}
CORN	1.32×10^{-23}

horizon of 12. This is mainly because the `anomalous-acm` package that we use to extract features uses internally an STL decomposition, and therefore needs 2 full periods of data, i.e., 24 data points in our case. So, as the clustering method is not applicable for some of the CIF2016 series, the LSTM.Cluster variant is only performed on 57 time series with 12 months of forecasting horizon. As a result, in addition to the LSTM models that are generated for each cluster, a separate LSTM model is built for the remaining 15 time series with forecasting horizon of 6 months. We introduce several variants to this LSTM.Cluster methodology by employing different clustering algorithms, namely: LSTM.Cluster.kMeans.Elbow, LSTM.Cluster.kMeans.Silhouette, LSTM.Cluster.Dbscan, LSTM.Cluster.PAM.Silhouette and LSTM.Cluster.Snob. Here, Elbow and Silhouette suffixes represent the approach used to determine the optimal number of clusters in the corresponding clustering algorithm.

LSTM.All. The baseline LSTM algorithm, where no subgrouping is performed but one model is generated globally across all time series in the dataset. Note that, as some series from the CIF2016 dataset are very short, we use here an *inputSize* of 7 throughout, and an *outputSize* of 12, both for target horizons 6 and 12.

4.4. Statistical tests of the results

We perform a non-parametric Friedman rank-sum test with Hochberg's post-hoc procedure to assess the presence of statistical significance of differences within multiple forecasting methods, and then to further characterize these differences (García, Fernández, Luengo, & Herrera, 2010).[2] The statistical testing is done using the sMAPE measure.

We use the statistical testing framework in two steps. First, we determine whether the differences among the proposed and the base model are statistically significant. Then, we compare our approaches to competitive benchmarks, in particular to the original participants of the competitions where possible. A significance level of $\alpha = 0.05$ is used.

4.5. Performed experiments

To provide a thorough empirical study that allows us to assess accurately the performance of our approach in different situations, we perform experiments and evaluations with three different setups as follows:

Competition setup (CO): We run our methods under the original competitions' setup. We use a fixed origin with a withheld test set, and evaluate using the mean sMAPE, which is the primary evaluation metric used in both competitions. In this scenario, we are able to compare the proposed methods against all participants of the original competitions.

Fixed origin setup (FO): In the CIF2016 competition, the forecasts from the originally participating methods are publicly available. This enables us to perform our two-stepped statistical testing procedure and compare against all participating methods as benchmarks. From the NN5, to the best of our knowledge the actual forecasts for each method on each time series are not publicly available, and we are unable to perform testing for statistical significance on all originally participating methods. Alternatively, we use controlled benchmarks to measure the statistical significance against our approach. In particular, we use ARIMA, seasonal naïve, and ETS methods from the `forecast` package. We also use the ES method from the `smooth` package (Svetunkov, 2017), which is an alternative implementation of exponential smoothing. Compared to its counterpart ETS from the `forecast` package, ES is not restricted by

[2] 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 7
Results of the rolling origin evaluation on selected benchmarks, using the Mean sMAPE measure, for the 72 monthly series of CIF2016, in ascending order. For each column, the results of the best performing method(s) are marked in boldface.

Method	Mean sMAPE	Median sMAPE	Rank sMAPE	Mean MASE	Median MASE	Rank MASE
LSTM.Horizon	**9.80**	7.08	5.50	0.79	0.61	5.38
LSTM.Cluster.Snob	9.95	7.01	3.75	0.81	0.61	3.88
ETS	10.76	6.59	3.25	0.78	0.52	3.06
BaggedETS	10.77	**5.97**	**2.12**	**0.76**	**0.48**	**2.31**
ES	11.04	6.56	3.00	0.78	0.49	3.06
LSTM.All	11.10	6.97	4.25	0.78	0.63	4.25
ARIMA	12.76	7.28	6.38	0.83	0.55	6.31
Naïve Seasonal	19.82	12.09	7.75	1.38	0.99	7.75

Table 8
Original Mean sMAPE results for the 111 daily series of the NN5, together with the results from our methods, in ascending order.

Contender name	Mean sMAPE
Wildi	19.9
Andrawis	20.4
Vogel	20.5
D'yakonov	20.6
Noncheva	21.1
LSTM.Cluster.Snob	**21.6**
Rauch	21.7
Luna	21.8
Lagoo	21.9
Wichard	22.1
Gao	22.3
LSTM.Cluster.kMeans.Elbow	**22.6**
LSTM.Cluster.PAM.Silhouette	**22.7**
LSTM.Cluster.kMeans.Silhouette	**22.7**
LSTM.Cluster.kMeans.Dbscan	**22.8**
LSTM.All	**23.4**
Puma-Villanueva	23.7
Autobox(Reilly)	24.1
Lewicke	24.5
Brentnall	24.8
Dang	25.3
Pasero	25.3
Adeodato	25.3
undisclosed	26.8
undisclosed	27.3
Tung	28.1
Naïve Seasonal	28.8
undisclosed	33.1
undisclosed	36.3
undisclosed	41.3
undisclosed	45.4
Naïve Level	48.4
undisclosed	53.5

the number of seasonal coefficients to be included in the model. Furthermore, these controlled benchmarks are univariate forecasting techniques that forecast time series in isolation. Consequently, these models are unable to exploit any advantage of prior time series sub grouping, so that we do not combine them with clustering approaches.

Rolling origin setup (RO): We also perform rolling origin evaluation using the test sets by averaging the accuracy over different forecasting origins (Tashman, 2000). This enables us to obtain stronger evidence as results depend less on particular forecast origins and the evaluation is performed on more data points. Again, as this setup differs from the original competition setup, we are unable to compare against the original participants, and evaluate against the benchmark methods.

Following the nomenclature of Tashman (2000), we use the benchmark models in a recalibration mode. Here, for each forecast origin, as iteratively new data points are included in the training set while the forecast origin moves forward, we re-fit the respective benchmark forecasting model. For the LSTM variants, re-fitting the entire model for each origin is costly, and we do not re-fit the model. Instead, we use an updating mode (Tashman, 2000), where we use the initial model which is build from the original forecasting origin. New data points are used to update the model but no re-fitting of parameters is performed. For simplicity, among the LSTM.Cluster variants, we employ only LSTM.Cluster.Snob, which is an autonomous clustering algorithm to experiment under this setting.

We also provide a detailed breakdown of the time series groups identified by each clustering algorithm. Specifically, we report number of clusters, number of time series assigned to each cluster, and the variance within each cluster. We determine the cluster variance by obtaining the mean of the variances of the normalised extracted feature associated with each cluster.

Moreover, we summarise the execution times of the proposed LSTM variants under the FO setup. A breakdown of pre-processing, model training and post-processing times is reported to provide better insights of the execution times relevant to each layer of our forecasting framework.

4.6. Results on the CIF2016 dataset

Table 3 shows the results of the CO/FO evaluation setup for the CIF2016 dataset. The table reports results of our methods (marked in boldface in the table), together with the original results of the competition; those methods are described in Štěpnička and Burda (2016). Furthermore, we added the ES method (Svetunkov, 2017) as a benchmark to complete our benchmarking suite in line with the other experiments.

We see that regarding the mean sMAPE, which is the primary measure used in the original competition, the LSTM.Cluster.kMeans.Silhouette variant from our proposed methods outperforms all other methods from the competition. In particular, it also outperforms the baseline LSTM.All variant, and all LSTM variants outperform the ETS, BaggedETS, and Theta methods, which can be seen as the state of the art for forecasting monthly data. Moreover, except for LSTM.Cluster.PAM.Silhouette, all the other LSTM.Cluster variants outperform the baseline LSTM.All variant. Considering the other measures, the results change quite drastically, and traditional univariate methods such as ES, ETS, BaggedETS and LSTMs and ETS, which uses ETS model parameters as external features in LSTM, outperform our methods. This can be attributed to the peculiarities of the CIF2016 dataset. The dataset contains 48 artificially generated time series that may not contain useful cross-series information. This claim is strengthened by the fact that already the BaggedETS submission to the competition outperformed all other methods in this subset, in particular the LSTM methods (Štěpnička & Burda, 2016).

We also note that the under-performance of LSTM.Cluster.PAM.Silhouette compared to other LSTM.Cluster

Table 9
Results of the fixed origin evaluation for the 111 daily series of NN5, ordered by the first column, which is the Mean sMAPE. For each column, the results of the best performing method(s) are marked in boldface.

Method	Mean sMAPE	Median sMAPE	Rank sMAPE	Mean MASE	Median MASE	Rank MASE
ES	**21.44**	**20.29**	5.13	**0.86**	**0.80**	4.66
ETS	21.46	20.57	5.07	**0.86**	0.81	4.58
BaggedETS	21.46	20.57	5.07	0.87	0.84	5.37
LSTM.Cluster.Snob	21.66	20.71	6.00	0.94	0.89	7.29
LSTM.Cluster.kMeans.Elbow	22.57	20.89	5.50	0.90	0.84	5.13
LSTM.Cluster.PAM.Silhouette	22.67	20.54	**4.87**	0.90	0.83	**4.41**
LSTM.Cluster.kMeans.Silhouette	22.72	21.02	5.39	0.90	0.83	5.04
LSTM.Cluster.Dbscan	22.79	21.30	5.35	0.90	0.83	4.55
LSTM.All	23.46	21.75	7.46	0.96	0.93	7.84
ARIMA	25.29	21.74	7.29	0.97	0.90	6.78
Naïve Seasonal	26.49	23.31	8.86	1.01	0.94	10.34

Table 10
Cluster results for the NN5 dataset, columns representing the number of clusters, number of time series for each cluster and the variance associated with each cluster, respectively.

Cluster variant	ClusterID	No. time series	Variance
LSTM.Cluster.kMeans.Silhouette	1	41	0.187
	2	70	0.187
LSTM.Cluster.kMeans.Elbow	1	31	0.166
	2	39	0.146
	3	41	0.187
LSTM.Cluster.Dbscan	1	8	0.286
	2	67	0.179
	3	36	0.159
LSTM.Cluster.Snob	1	27	0.180
	2	25	0.154
	3	9	0.263
	4	6	0.212
	5	25	0.178
	6	19	0.178
LSTM.Cluster.PAM.Silhouette	1	69	0.183
	2	42	0.200

variants can be attributed to the large number of clusters identified in the clustering phase. According to Table 4, the LSTM.Cluster.PAM.Silhouette variant has the highest number of clusters among others. This may have affected the LSTM training procedure by loosing potential cross series information available across the original set of time series, i.e., now the time series are dispersed across many groups, thus exploitation of the information becomes difficult for our LSTM modelling framework. Even though LSTM.Cluster.PAM.Silhouette returns time series groups with less variance to other cluster variants, it does not necessarily provide better forecasts.

Also, with respect to computational cost of the proposed variants, according to Table 5 we see that the clustering approaches have a better model training time compared to the baseline LSTM.All model. This is mainly because, the clustering procedure enables us to train multiple LSTM models (for each cluster) in parallel. The LSTM.Cluster.PAM.Silhouette has the best model training time as here model parallelism is higher due to the number of clusters in the model.

For the first step of our statistical testing evaluation, where we compare the LSTM variants among themselves, the corresponding Friedman test has an overall p-value of 0.3063, which highlights that these differences are insignificant and all LSTM methods should be considered to have comparable performance on this dataset.

As the second step of our statistical testing evaluation, we compare the LSTM variants against the other participant methods from the CIF2016 competition. The Friedman test for multiple comparisons results in an overall p-value of 1.96×10^{-10}. Hence, these differences are highly significant, and Table 6 shows the associated post-hoc testing. The LSTMs and ETS method achieves the best ranking and is used as the control method. We see that the LSTM.All and LSTM.Cluster variants do not perform significantly worse than this control method.

Table 7 shows the evaluation summary of the RO setup, where the proposed LSTM.Horiz-on variant obtains the best Mean sMAPE, outperforming the baseline LSTM.All variant and all other state-of-the-art univariate forecasting methods such as ETS, BaggedETS, ES, ARIMA, and Naïve Seasonal. On all the other error measures, BaggedETS performs best, which is consistent with the particularities of this dataset as discussed earlier.

4.7. Results on the NN5 dataset

Table 8 shows the original results of the CO setup for the NN5 forecasting competition data, together with results of our methods. It can be seen that all the proposed LSTM.Cluster variants perform better than the baseline LSTM.All variant. In particular, LSTM.Cluster.Snob reaches a 6^{th} overall rank. Note that the proposed LSTM.Horizon variant is not benchmarked against the NN5 dataset, as no additional information is available in this case.

Tables 9 and 12 provide results of the FO and RO setup evaluations on the NN5 dataset. As for the NN5 the forecasts from the original participants are not available, both FO and RO evaluations, as well as the statistical testing, are performed using the benchmark methods.

We see that all the LSTM.Cluster variants consistently outperform the LSTM.ALL variant across all error measures for the fixed origin setup. Though the exponential smoothing benchmarks ES and ETS are able to outperform the proposed LSTM variants in terms of the Median sMAPE, Mean MASE and Median MASE, we see that the LSTM.Cluster.PAM.Silhouette variant achieves better average ranking of sMAPE and MASE error measures compared to other methods. Furthermore, the LSTM.Cluster.Snob variant also outperforms the LSTM.ALL variant across all error measures for fixed origin setup.

Also, among the LSTM.Cluster variants, except for Mean sMAPE, the LSTM.Cluster.PAM.Silhouette variant outperforms the rest of the LSTM.Cluster variants in every other error measure. Though the LSTM.Cluster.Snob variant achieves better results in terms of Mean sMAPE, it performs poorly for the rest of the error measures. This performance difference can be explained by the number of clusters identified by the Snob clustering algorithm. As summarised in Table 10, the Snob clustering variant returns the highest number of clusters for the NN5 dataset, whereas the overall best performing LSTM.Cluster variant, LSTM.Cluster.PAM.Silhouette gives only two clusters.

Consistent with our previous findings, Table 11 shows that the clustering approaches have uplifted the LSTM model training time by enabling model parallelism.

Table 11
Summary of computation times of the proposed LSTM variants for the NN5 dataset (in minutes).

LSTM Variant	Pre-processing	Model training	Post-processing	Total time
LSTM.All	47.6	90.5	0.6	138.2
LSTM.Cluster.kMeans.Silhouette	34.3	57.0	0.8	92.1
LSTM.Cluster.kMeans.Elbow	24.2	47.2	0.7	72.1
LSTM.Cluster.Dbscan	24.3	50.5	0.7	75.5
LSTM.Cluster.Snob	20.4	35.4	0.4	56.2
LSTM.Cluster.PAM.Silhouette	36.6	60.6	0.8	98.0

Table 12
Results of the rolling origin evaluation, ordered by the Mean sMAPE measure, for the 111 daily series of NN5, in ascending order. For each column, the results of the best performing method(s) are marked in boldface.

Method	Mean sMAPE	Median sMAPE	Rank sMAPE	Mean MASE	Median MASE	Rank MASE
ES	**22.18**	20.75	2.66	**0.87**	**0.83**	2.50
ETS	22.24	**20.56**	**2.63**	**0.87**	**0.83**	**2.47**
BaggedETS	22.39	20.80	3.00	0.88	0.86	3.00
LSTM.Cluster.Snob	23.38	22.27	4.18	0.93	0.89	4.30
LSTM.All	23.89	22.44	4.87	0.95	0.93	4.83
ARIMA	23.96	22.61	3.85	0.96	0.94	4.11
Naïve Seasonal	28.23	26.10	6.81	1.10	1.04	6.78

Table 13
Results of statistical testing for NN5 data, using the results of the selected benchmarks and the LSTM variants. LSTM.Cluster.PAM.Silhouette performs best overall, ES, BaggedETS, and other cluster variants do not perform significantly worse.

Method	p_{Hoch}
LSTM.Cluster.PAM.Silhouette	–
LSTM.Cluster.Dbscan	0.663
LSTM.Cluster.kMeans.Silhouette	0.663
LSTM.Cluster.kMeans.Elbow	0.663
BaggedETS	0.663
ETS	0.663
ES	0.663
LSTM.Cluster.kMeans.Snob	0.075
ARIMA	4.68×10^{-7}
LSTM.All	5.70×10^{-8}
Naïve Seasonal	3.74×10^{-18}

For the first step of our statistical testing evaluation, where we compare the LSTM variants among themselves, the corresponding Friedman test has an overall p-value of 0.3063, which highlights that these differences are insignificant and all LSTM methods should be considered to have comparable performance on this dataset.

The Friedman test gives an overall p-value of 1.80×10^{-9} within the LSTM variants. Therefore, the differences among the LSTM.Cluster variants and LSTM.All are highly significant.

Table 13 shows the results of the second step of our statistical testing evaluation. The overall result of the aligned Friedman rank sum test is a p-value of 8.24×10^{-11}, which is highly significant. The LSTM.Cluster.PAM.Silhouette cluster variant performs best and is used as the control method. Also, according to Table 13, we see that the baseline LSTM method (LSTM.All) performs significantly worse than the control method, whereas other LSTM.Cluster variants do not.

5. Conclusions

Nowadays, large quantities of related and similar time series are available in many application cases. To exploit the similarities between multiple time series, recently methods to build global models across such time series databases have been introduced. One very promising approach in this space are Long Short-Term Memory networks, a special type of recurrent neural networks.

However, in the presence of disparate time series, the accuracy of such a model may degenerate, and accounting for the notion of similarity between the time series becomes necessary. Motivated by this need, we have proposed a forecasting framework that exploits the cross-series information in a set of time series by building separate models for subgroups of time series, specified by a clustering methodology. To assess the robustness of our framework, we use various clustering techniques to determine the optimal number of clusters in a group of time series.

We have evaluated our proposed methodology on two benchmark competition datasets, and have achieved competitive results. On the CIF2016 dataset, the majority of our proposed methods outperform all the other methods from the competition with respect to the evaluation metric used in the competition. Also, in the NN5 competition dataset, one of the proposed methods ranks 6th overall, while another cluster variant is the best performing method, in terms of the average rankings over the evaluated error measures.

The results also indicate that in the majority of cases, the proposed LSTM Cluster variants achieve improvements over the baseline LSTM model. Therefore, a prior time series clustering has supplemented the LSTM training procedure by improving the homogeneousness of the trainable time series. Nevertheless, we observe that the number of time series groups identified by the clustering algorithm influences the final performance of the model. We attribute this to the information loss that could occur as a result of the grouping of time series. Hence, the process of obtaining the optimal number of clusters becomes a challenging factor in this approach, as it needs to balance the cross series information available across a group of time series. Therefore, approaches to exploit the cross series information available across a collection of time series, while obviating any information loss, are necessary in this space. Furthermore, we have also observed that the model parallelism enabled by the proposed time series clustering procedure has benefited the training time of the LSTM baseline model significantly.

Overall, the results show that the LSTM is a competitive method, effectively exploiting similarities of the time series and therewith being able to outperform state-of-the-art univariate forecasting methods. Subgrouping of similar time series with our pro-

posed methodology augments the accuracy of this baseline LSTM model in many situations.

Declaration of Conflict of 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

Credit authorship contribution statement

Kasun Bandara: Conceptualization, Writing - original draft, Writing - review & editing, Formal analysis, Investigation, Data curation. **Christoph Bergmeir:** Conceptualization, Writing - original draft, Writing - review & editing, Visualization, Supervision, Project administration. **Slawek Smyl:** Conceptualization, Writing - original draft, Writing - review & editing, Visualization, Supervision, Project administration.

References

Aghabozorgi, S., Seyed Shirkhorshidi, A., & Ying Wah, T. (2015). Time-series clustering – a decade review. *Information Systems, 53*, 16–38.

Andiojaya, A., & Demirhan, H. (2019). A bagging algorithm for the imputation of missing values in time series. *Expert Systems with Applications, 129*, 10–26.

Armstrong, J. S. (2006). Findings from evidence-based forecasting: Methods for reducing forecast error. *International Journal of Forecasting, 22*(3), 583–598.

Ben Taieb, S., Bontempi, G., Atiya, A., & Sorjamaa, A. (2011). A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition,.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks, 5*(2), 157–166.

Berkhin, P. (2006). A survey of clustering data mining techniques. In J. Kogan, C. Nicholas, & M. Teboulle (Eds.), *Grouping multidimensional data: Recent advances in clustering* (pp. 25–71). Berlin, Heidelberg: Springer Berlin Heidelberg.

Box, G. E. P., & Cox, D. R. (1964). An analysis of transformations. *The Journal of the Royal Statistical Society. Series B (Methodological), 26*(2), 211–252.

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

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

Crone, S. F., Hibon, M., & Nikolopoulos, K. (2011). Advances in forecasting with neural networks? empirical evidence from the NN3 competition on time series prediction. *International Journal of Forecasting, 27*(3), 635–660.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems, 2*(4), 303–314.

Davydenko, A., & Fildes, R. (2013). Measuring forecasting accuracy: The case of judgmental adjustments to SKU-level demand forecasts. *International Journal of Forecasting, 29*(3), 510–522.

Duan, Y., Lv, Y., & Wang, F.-Y. (2016). Travel time prediction with LSTM neural network. In *2016 IEEE 19th international conference on intelligent transportation systems (ITSC)* (pp. 1053–1058).

Elman, J. L. (1990). Finding structure in time. *Cognitive Science, 14*(2), 179–211.

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the second international conference on knowledge discovery and data mining*. In *KDD'96* (pp. 226–231). Portland, Oregon: AAAI Press.

Faraway, J., & Chatfield, C. (2008). Time series forecasting with neural networks: a comparative study using the air line data. *The Journal of the Royal Statistical Society, Series C (Applied Statistics), 47*(2), 231–250.

Fei, M., & Yeung, D. Y. (2015). Temporal models for predicting student dropout in massive open online courses. In *2015 IEEE international conference on data mining workshop (ICDMW)* (pp. 256–263).

Fernando (2017). bayesian-optimization: Bayesian optimization of hyperparameters.

Fulcher, B. D., & Jones, N. S. (2014). Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering, 26*(12), 3026–3037.

García, S., Fernández, A., Luengo, J., & Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences, 180*(10), 2044–2064.

Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: continual prediction with LSTM. *Neural Computation, 12*(10), 2451–2471.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning*: 1. MIT press Cambridge.

Graves, A., r. Mohamed, A., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing* (pp. 6645–6649). ieeexplore.ieee.org.

Guerrero, V. M. (1993). Time-series analysis supported by power transformations. *Journal of Forecasting, 12*(1), 37–48.

Hahsler, M., & Piekenbrock, M. (2018). dbscan: Density based clustering of applications with noise (DBSCAN) and related algorithms.

Hans Franses, P. (1992). Testing for seasonality. *Economic Letters, 38*(3), 259–262.

Hartmann, C., Hahmann, M., Lehner, W., & Rosenthal, F. (2015). Exploiting big data in time series forecasting: A cross-sectional approach. In *2015 IEEE international conference on data science and advanced analytics (DSAA)* (pp. 1–10).

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München, 91*.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation, 9*(8), 1735–1780.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks, 4*(2), 251–257.

Hyndman, R. J. (2014). Detecting seasonality. Blog.

Hyndman, R. J. (2016). Q&A time | rob J hyndman. https://robjhyndman.com/hyndsight/qa-time/. Accessed: 2017-9-5.

Hyndman, R. J., & Athanasopoulos, G. (2014). *Forecasting: principles and practice*. OTexts.

Hyndman, R. J., Athanasopoulos, G., Razbash, S., Schmidt, D., Zhou, Z., Khan, Y., … Wang, E. (2015). forecast: Forecasting functions for time series and linear models. *R package version, 6*(6), 7.

Hyndman, R. J., & Billah, B. (2003). Unmasking the theta method. *International Journal of Forecasting, 19*(2), 287–290.

Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*.

Hyndman, R. J., Wang, E., & Laptev, N. (2015). Large-Scale unusual time series detection. In *2015 IEEE international conference on data mining workshop (ICDMW)* (pp. 1616–1619).

Ilies, I., Jaeger, H., Kosuchinas, O., Rincon, M., & others (2007). Stepping forward through echoes of the past: Forecasting with echo state networks. https://neural-forecastingcompetition.com/downloads/methods/27-NN3 Herbert Jaeger report. pdf,.

Kassambara, A., & Mundt, F. (2017). factoextra: Extract and visualize the results of multivariate data analyses.

Kaufman, L., & Rousseeuw, P. J. (1990). Partitioning around medoids (program PAM). In *Finding groups in data* (pp. 68–125). Hoboken, NJ, USA: John Wiley & Sons, Inc..

Khandakar, Y., & Hyndman, R. J. (2008). Automatic time series forecasting: The forecast package for R. *Journal Statistical Software, 27*(03).

Kodinariya, T. M., & Makwana, P. R. (2013). Review on determining number of cluster in K-Means clustering. *Aquatic Microbiology Ecology, 1*(6), 90–95.

Längkvist, M., Karlsson, L., & Loutfi, A. (2014). A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters, 42*, 11–24.

Le, Q. V., Jaitly, N., & Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units,.

Lee, C.-Y., Xie, S., Gallagher, P., Zhang, Z., & Tu, Z. (2015). Deeply-Supervised nets. In *Artificial intelligence and statistics* (pp. 562–570).

Lipton, Z. C., Kale, D. C., Elkan, C., & Wetzel, R. (2015). Learning to diagnose with LSTM recurrent neural networks,.

Lloyd, S. P. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*.

Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., & Hornik, K. (2019). cluster: Cluster analysis basics and extensions.

Makridakis, S., Andersen, A., Carbone, R., Fildes, R., Hibon, M., Lewandowski, R., … Winkler, R. (1982). The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting, 1*(2), 111–153.

Makridakis, S., & Hibon, M. (2000). The M3-Competition: Results, conclusions and implications. *International Journal of Forecasting, 16*(4), 451–476.

Marseguerra, M., Minoggio, S., Rossi, A., & Zio, E. (1992). Neural networks prediction and fault diagnosis applied to stationary and non stationary ARMA modeled time series. *Progress in Nuclear Energy, 27*(1), 25–36.

Mikolov, T., Karafiát, M., Burget, L., Cernock\'y, J., & Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech: 2* (p. 3). fit.vutbr.cz.

Mörchen, F. (2003). Time series feature extraction for data mining using DWT and DFT.

Moritz, S., Sardá, A., Bartz-Beielstein, T., Zaefferer, M., & Stork, J. (2015). Comparison of different methods for univariate time series imputation in R,.

Nelson, M., Hill, T., Remus, W., & O'Connor, M. (1999). Time series forecasting using neural networks: should the data be deseasonalized first? *Journal Forecasting, 18*(5), 359–367.

Ord, K., Fildes, R. A., & Kourentzes, N. (2017). *Principles of business forecasting.2nd ed*. Wessex Press Publishing Co.

Petersen, N. C., Rodrigues, F., & Pereira, F. C. (2019). Multi-output bus travel time prediction with convolutional LSTM neural network. *Expert Systems Application, 120*, 426–435.

Prokhorov, D. V., Feldkarnp, L. A., & Tyukin, I. Y. (2002). Adaptive behavior with fixed weights in RNN: an overview. In *Neural networks, 2002. IJCNN '02. proceedings of the 2002 international joint conference on: 3* (pp. 2018–2022).

R2RT Blog (2016). Written memories: Understanding, deriving and extending the LSTM - R2RT. https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html. Accessed: 2017-9-10.

Schapire, R. E. (2003). The boosting approach to machine learning: An overview. In D. D. Denison, M. H. Hansen, C. C. Holmes, B. Mallick, & B. Yu (Eds.), *Nonlinear estimation and classification* (pp. 149–171). New York, NY: Springer New York.

Seide, F., & Agarwal, A. (2016). CNTK: Microsoft's Open-Source Deep-Learning toolkit. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. In *KDD '16* (p. 2135). New York, NY, USA: ACM.

Smyl, S., & Kuber, K. (2016). Data preprocessing and augmentation for multiple short time series forecasting with recurrent neural networks. *36th international symposium on forecasting*.

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 25* (pp. 2951–2959). Curran Associates, Inc..

Štěpnička, M., & Burda, M. (2016). Computational intelligence in forecasting (CIF) 2016 time series forecasting competition. *IEEE WCCI 2016, JCNN-13 advances in computational intelligence for applied time series forecasting (ACIATSF)*.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). 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* (pp. 3104–3112). Curran Associates, Inc..

Svetunkov, I. (2017). Statistical models underlying functions of 'smooth' package for R. *eprints.lancs.ac.uk/85045/*, 52.

Tashman, L. J. (2000). Out-of-sample tests of forecasting accuracy: An analysis and review. *International Journal of Forecasting, 16*(4), 437–450.

Trapero, J. R., Kourentzes, N., & Fildes, R. (2015). On the identification of sales forecasting models in the presence of promotions. *Journal of Operation Research Society, 66*(2), 299–307.

Wallace, C. S., & Dowe, D. L. (1994). Intrinsic classification by MML-the snob program. In *Proceedings of the 7th australian joint conference on artificial intelligence: 37* (p. 44).

Wallace, C. S., & Dowe, D. L. (2000). MML clustering of multi-state, Poisson, von mises circular and Gaussian distributions. *Statistical Computing, 10*(1), 73–83.

Wang, X., Smith, K., & Hyndman, R. (2006). Characteristic-based clustering for time series data. *Data Mining Knowledge Discovery, 13*(3), 335–364.

Warren Liao, T. (2005). Clustering of time series data—a survey. *Pattern Recognition, 38*(11), 1857–1874.

Williams, R. J., & Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, Architectures, and Applications, 1*, 433–486.

Yan, W. (2012). Toward automatic time-series forecasting using neural networks. *IEEE Transactions Neural Network Learning System, 23*(7), 1028–1039.

Zaiyong Tang, de Almeida, C., & Fishwick, P. A. (1991). Time series forecasting using neural networks vs. box- jenkins methodology. *Simulation, 57*(5), 303–310.

Zhang, G., Patuwo, B. E., & Hu, M. Y. (1998). Forecasting with artificial neural networks:: The state of the art. *International Journal of Forecasting, 14*(1), 35–62.

Zhang, G. P., & Qi, M. (2005). Neural network forecasting for seasonal and trend time series. *European Journal Operational Research, 160*(2), 501–514.

Zimmermann, H.-G., Tietz, C., & Grothmann, R. (2012). Forecasting with recurrent neural networks: 12 tricks. In *Neural networks: Tricks of the trade*. In *Lecture Notes in Computer Science* (pp. 687–707). Springer, Berlin, Heidelberg.