A First Approximation to the Effects of Classical Time Series Preprocessing Methods on LSTM Accuracy

Daniel Trujillo Viedma(✉), Antonio Jesús Rivera Rivas, Francisco Charte Ojeda, and María José del Jesus Díaz

Andalusian Research Institute on Data Science and Computational Intelligence (DaSCI), Computer Science Department, University of Jaén, 23071 Jaén, Spain
{dtviedma,arivera,fcharte,mjjesus}@ujaen.es

Abstract. A convenient data preprocessing has proven to be crucial in order to achieve high levels of accuracy, time series being no exception. For this kind of forecasting tasks, several specialized preprocessing methods have been described, being trend analysis and seasonal analysis some of them. Several have been formally grouped around a methodology that is always applied to state of the art time series forecasting models, like the well known ARIMA.

LSTM is a relatively novel architecture which has been specifically designed to get rid of the vanishing gradient problem. In these models, great results have been seen when applied for time series forecasting. Still, little is known about the impact of these traditional preprocessing methods on the accuracy of LSTM.

In this work an empirical analysis on how classical time series preprocessing methods influence LSTM results is conducted. That all considered ones can potentially improve LSTM performance is concluded, being the seasonal component removal the filter that achieves better, most robust accuracy gain.

Keywords: LSTM · Time series · Preprocessing

1 Introduction

Feeding not only clean, but also well-formatted, data to data mining models usually improves their overall performance. Many studies have highlighted the importance of a good preprocessing analysis [2,10].

In the case of time series, this is not an exception, but given the peculiarity of this kind of datasets, specialized methods must be used along with several general purpose ones. Traditional time series modelling tools like ARIMA [9], which is considered the state of the art because of its good results and strong statistical foundations, are never applied without a set of preprocessing strategies called the Box-Jenkins methodology [1]. This demonstrate the great usefulness of applying these preprocessing techniques.

I. Rojas et al. (Eds.): IWANN 2019, LNCS 11506, pp. 270–280, 2019.
https://doi.org/10.1007/978-3-030-20521-8_23

Box-Jenkins methodology [1] can be divided into several simpler tasks to transform the original time series into an stationary one: trend and seasonal components detection and removal. They will be addressed in this work along with another useful one: an analysis of lags.

LSTM (*Long Short-Term Memory*) [7] are a kind of Deep Learning, recurrent neural networks that have been designed with an important limitation of other deep learning methods in mind: the vanishing gradient problem widely discussed in [6], in which several approaches to solve it are also introduced. LSTM includes several mechanisms to overcome this limitation, like the Constant Error Carrousel, that makes them able to remember filtered information. This model was later applied to time series with impressive results, mainly because of its capability to model long time dependencies.

This work highlights empirically the effect of the mentioned fundamental preprocessing methods on LSTM predictive performance, by predicting from time series that have been preprocessed in several ways.

This document is structured as follows. First, a briefly description of the LSTM is presented in Sect. 2, then the main preprocessing methods traditionally used for time series data mining is enumerated in Sect. 3. After that, the entire experimentation carried out and its results are detailed in Sect. 4. Lastly, a conclusion is given at Sect. 5.

2 LSTM

An LSTM is a recurrent artificial neural cell which has been specifically designed to solve the vanishing gradient problem. The proposal in [7] defines a complex neural cell in which information is filtered through several gates, and also transformed accordingly to a previously memorized value computed from the last forward propagation run. The LSTM's structure has been later successfully applied to time series to find that its internal mechanisms let it model very long time dependencies.

LSTM does so by minimizing the vanishing gradient problem, very common in deep learning models, like the recurrent neural networks. The vanishing gradient problem is detailed in [6]. It prevents a neural network from continuing to learn beyond a certain point due the lack of effect from layers which are furthest from the output.

An LSTM cell consist of a relatively complex data pipeline that makes it able to remember an internal state, while operating on input data to compute its output. The pipeline is composed of activation functions that filter and transform data; and additive and multiplicative operations that merge and also transform data.

In Fig. 1, a more detailed description of the contemporary pipeline, also referred to as *Vanilla LSTM* [5] and firstly described in [4], can be found. Comparing this LSTM with the original one, presented in [7], two main additions can be easily seen:

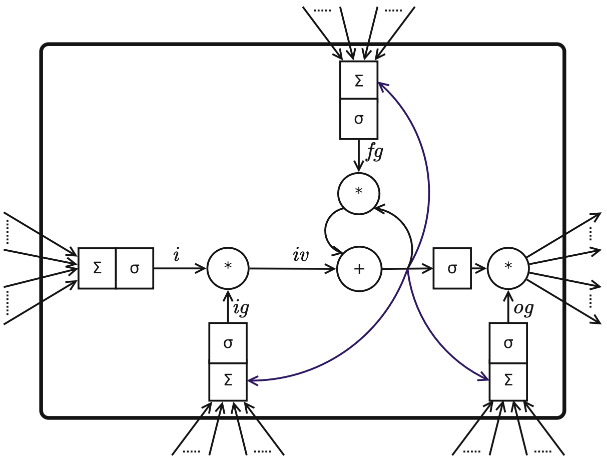


Fig. 1. Schematic of an LSTM cell.

Forget gate Introduced in [3], this gate demonstrated to be crucial to improve LSTM performance, at the cost of increasing the total number of parameters to train.

Peephole connections Data from inside the cell is made available through peephole connections to the gates, so data can be better refined before entering the cell.

Figure 1 also gives an easy way to understand how data flows inside the cell: The input data flows from the leftmost input gate to the output of the cell, located at the opposite side. In between, it is transformed by means of pointwise multiplications with values provided by additional gates, each one trained to filter data in a particular way: the input gate filter the input data to prevent noise coming from outside the cell disturb the memorized value, while the output gate performs a final filtering to protect next layer cells from bad memorized values.

Also, an internal loop can be appreciated, regulated by the forget gate. This is called CEC (*Constant Error Carousel*), and constitutes the mechanism by which the LSTM memorizes knowledge for managing long term dependencies. This memorized value can be influenced by new input data, but also filtered through the forget gate.

The data (both new and recurrent entries) flowing through the pipeline described in Fig. 1 first enter the cell through, performing the computations on the four input gates, then is merged by means of multiplicative and additive (pointwise or scalar) operations and then, feeded to a final activation function like an ordinary neural cell.

LSTM neural cells are commonly trained using the well-known BPTT (*Back-propagation Through Time*) algorithm [11], which models the contribution of each network parameter to the final error and greedily modifies it so the error is minimized.

3 Preprocessing Methods

Traditional time series data mining algorithms greatly improve their performance when are trained with a preprocessed dataset. ARIMA is one of these methods, whose prediction accuracy increases when the original time series is decomposed into trend, seasonal, and random components, and are analyzed separately. Typically, the preprocessing methods achieving better results have been:

Significant lags detection Adds additional values to every instance. More precisely, this new values are the very same time series, but lagged at certain times. As a drawback, the values corresponding to the most earliest time steps must be dropped or predicted, because the lack of real lagged data for the first values of the dataset.

Trend component removal A trend component in the time series is detected and removed. Trend removal may be useful to palliate the effect of the magnitude of the variable, but can be affected by abrupt variations on the time series. This preprocessing method feeds the network with values that are not present on the original time series (the actual real value of the series is never given to the network, but the result of the chosen transformation), so the network output prediction has to be rebuilt to revert this transformation.

Seasonal component removal A seasonal component in the time series is detected and removed. Like in the *trend* case, the input values are transformed when deleting the seasonal component, si the output of network trained with these versions of the original time series must be, as well, rebuilt.

4 Experiments and Results

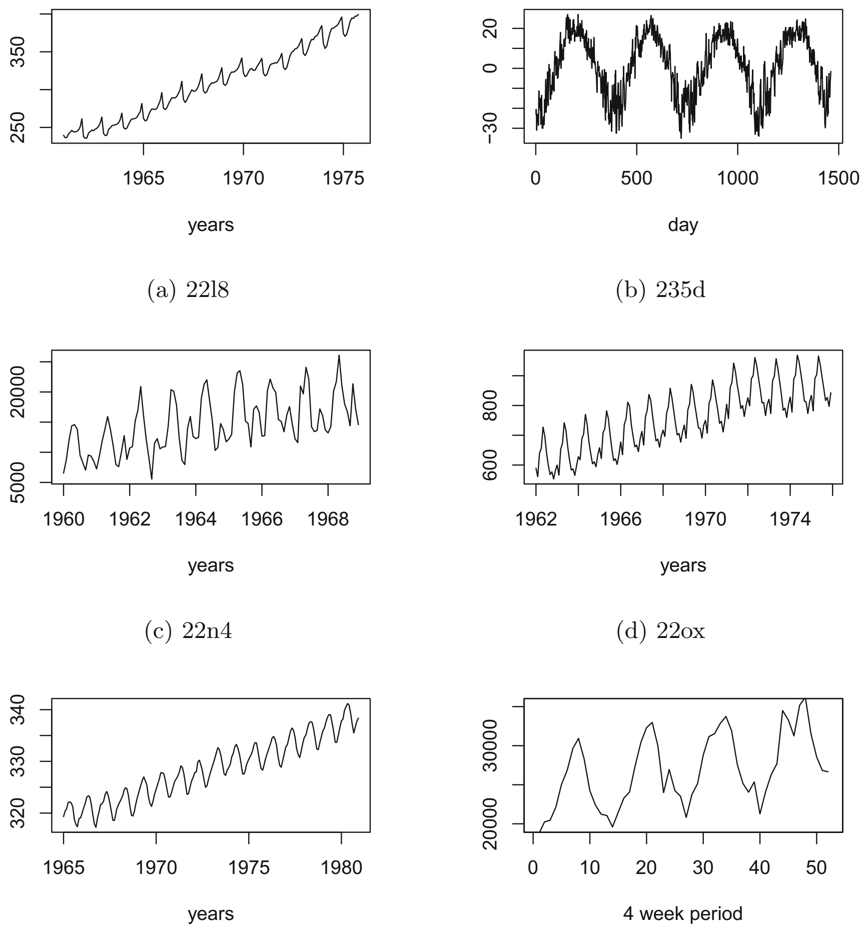
Several experiments have been conducted to extract enough data to establish a comparison between the chosen preprocessing methods. The details regarding this experimentation are introduced below.

4.1 Error Metric

In order to compare the performance of the model, a quality measure of the prediction must be established. One broadly used method to quantify this is through an error measure, and then following the *less is best* criteria. RMSE (*Root Mean Squared Error*) is a well known error metric that emphasizes the deviations more than others. It is computed as follows: given T the true values of the time series, and P the predicted ones, RMSE can be described as:

$$RMSE(T, P) = \sqrt{MSE(T, P)} \tag{1}$$

$$MSE(T, P) = \frac{\sum_{i=1}^{n}(T_i - P_i)^2}{n} \tag{2}$$



(a) 22l8 (b) 235d

(c) 22n4 (d) 22ox

(e) 22v1 (f) 2325

Fig. 2. Plots of the datasets considered

4.2 Datasets

In this work, 6 datasets from the Time Series Data Library [8] have been used. As it can be seen in Fig. 2, in general terms they expose a trending and seasonal behavior. This election of datasets has been made to maximize the effect of the preprocessing methods considered. In Table 1 descriptions of several aspects of them are included.

The size of these public, well-known time series, is small enough to let all the experimentation run in an acceptable time window while using relatively time-consuming models like the ones that are being trained.

Table 1. Datasets

Code	Description	Resolution	Period length	Dates
22l8	Wisconsin employment	Monthly	12	Jan. 1961
				Oct. 1975
22n4	Car sales in Quebec	Monthly	12	Jan. 1960
				Dec. 1968
22ox	Milk production: pounds per cow	Monthly	12	Jan. 1962
				Dec. 1975
22v1	CO_2 (ppm) mauna loa	Monthly	12	Jan. 1965
				Dec. 1980
235d	Mean temperature, Fisher River near Dallas	Daily	365	Jan. 1, 1988
				Dec. 31, 1991
2325	Totals of beer shipments	Four-weekly	14	Week 2, 1970
				Week 49, 1973

Table 2. Hyperparameters

Num. of LSTM	Epochs	Batch size	Topology	Repetitions
10	200	1	Dense	10

For evaluation purposes, these time series are divided into training and test sets, the latter being formed by the last 24 observations of the series (the most recent ones), whilst the rest of the instances fall into the training dataset.

These datasets constitute the raw data from which a baseline for comparison has to be set. Several versions of each one will be computed accordingly to the preprocessing methods that have been selected. The creation procedure and the instance form for each of these variants are detailed in Subsect. 4.4.

4.3 Model

The experiments have been coded in *Python*, making use of the *Keras* library with the *TensorFlow* backend and the *Scikit-learn* library for error computing. A single set of hyperparameters has been heuristically selected, after performing several tests, and taking execution time concernings into account. These chosen hyperparameters can be seen in Table 2.

Preliminar experiments did not achieve better accuracy after increasing the number of LSTM cells. On the other hand, given the relatively small size of the time series being considered, it is possible to set a small value for the *batch size* parameter. Low values have a great negative impact on the execution time, but because of the size of the datasets and the available computational power, a value of 1 is possible.

Several repetitions of the experiments have to be made because of the random strategy the training algorithm follows to initialize the weights of the LSTM network. In order to minimize the effect of an accidentally advantageous initial weight configuration by luck, 10 repetitions have been made, and then conclusions are extracted from a summarization statistic. This also improves the generalization of our conclusions, which are not tied to a particular execution.

4.4 Preprocessing Methods

For each one of the distinct time series considered, several datasets have been created, as a result of applying a preprocessing method to the original time series. These versions are introduced below.

Raw Time Series. This is a trivial identity preprocessing. The original time series remains unchanged. It is interesting to test the LSTM network with raw, unpreprocessed data, in order to set a baseline to which compare the rest of the non-trivial preprocessing methods.

The definition of an instance is also trivial, but will serve as an introduction to the notation of the definition of the instances. The instances in these time series have the form:

$$I_t = (t, V_t) \tag{3}$$

Being:

t Is the time period this instance refers to.
I_t The instance of the dataset at time t.
V_t The value of the dataset at time t.

Lags. In the experimentation carried out, the lags considered for each time depends on a very simple correlogram graph analysis consisting in selecting the lags whose partial autocorrelation function value exceeds the statistically significant threshold. Also, the first values have been dropped, in order to keep the lag window inside real data, instead of predicting those values and introducing noise. These datasets have enough values to be dramatically affected by this drop strategy.

The form of the lag-processed instances is defined as:

$$I_t = (t, V_t, V_{t-l_0}, \ldots, V_{t-l_n}) \tag{4}$$

Being:

t Is the time period this instance refers to.
I_t The instance of the dataset at time t.
V_t The value of the dataset at time t.
$l_0, ..., l_n$ The selected lags.

Trend. The trend component is removed by applying differentiation to the dataset. This is done by subtracting a value from the time series to the following one, so the differences between two consecutive values are given to the neural network. Trend removal may be useful to palliate the effect of the magnitude of the variable, but can be affected by abrupt variations on the time series.

After differentiate, the instances have the following form:

$$I_t = (t, V_{t+1} - V_t) \tag{5}$$

In the experiments carried out, the rebuild procedure for a particular instance takes the value at the output of the neural network, and adds it to the true value from the original dataset, so errors are not accumulated, which is a usual problem when relying solely on the network output values, and applying cumulative sum.

Seasonal. For this preprocessing method, a very simple and intuitive seasonal component detection and deletion procedures have been adopted. Then, that de-seasonalized time series have been fed to the neural network.

The detection of the seasonal component in a given dataset is performed by calculating the average of the values at the same position relative to the start of a period (it is a prerequisite that the time series period length is known).

Then, once the seasonal component has been modelled, seasonality is removed by subtracting the corresponding value of the modelled seasonality to the value of the series:

$$D_t = V_t - s_{u=mod(t,l)} \tag{6}$$

Being:

D_t Deseasonalized value at time t.
s_u Detected seasonality component value at position u.
l Period length, detected seasonality component length.

The instance for this particular preprocessing method is described as follows:

$$I_t = (t, D_t) \tag{7}$$

Just like in the *trend* case, transformed values are fed to the neural network, so another rebuilt has to be performed. This is done just by adding the previously detected seasonal component to each value on the prediction.

4.5 Results

In Table 3, average, standard deviation and variation of the error from each set of experiments can be found. These statistics are highlighted when the average of a preprocess improves the accuracy of the raw methodology. As pointed out in Subsect. 4.3, each value summarize 10 repetitions of the corresponding set of experiments that have been performed. The variation is calculated relative to the raw average statistic, so predictive performance can be compared across multiple datasets.

Table 3. LSTM RMSE

Dataset	Statistic	Raw	Lags	Trend	Seasonal
22l8	Average	8,982	14,453	**7,528**	**3,517**
	Std. Dev.	0,334	1,803	**0,226**	**0,520**
	Var.		60,909%	**−16,181%**	**−60,844%**
22n4	Average	3841,427	**2944,574**	4088,950	**1705,087**
	Std. Dev.	19,440	**152,559**	16,493	**12,934**
	Var.		**−23,347%**	6,443%	**−55,613%**
22ox	Average	45,148	**20,453**	**35,900**	**12,363**
	Std. Dev.	0,194	**2,102**	**1,074**	**0,756**
	Var.		**−54,699%**	**−20,485%**	**−72,616%**
22v1	Average	0,830	**0,562**	0,874	**0,639**
	Std. Dev.	0,031	**0,060**	0,003	**0,059**
	Var.		**−32,32%**	5,26%	**−23,04%**
235d	Average	5,317	9,931	5,437	**4,667**
	Std. Dev.	0,019	0,205	0,021	**0,037**
	Var.		86,794%	2,263%	**−12,225%**
2325	Average	2970,936	**2221,160**	2998,783	**1778,490**
	Std. Dev.	42,565	**111,955**	13,650	**34,674**
	Var.		**−25,237%**	0,937%	**−40,137%**

Analyzing average statistic values, we find out that the lagged preprocessing method makes the LSTM to predict better most of the cases. Something similar happens to the seasonal analysis, but with a more consistent behaviour: better LSTM performance is achieved in all cases studied. By contrast, the trend preprocessing method achieves almost the same performance.

In the case of datasets 22l8 and 235d, lagged preprocessing performs surprisingly bad, with almost double the error achieved without any data treatment. Taking a closer look at the plots (Figs. 2a and b) shows, for the dataset 235d, several differences between periods of a strong seasonal component, which may cause this preprocess to fail. On the other hand, on dataset 22l8 we found what probably is a slight concept drift at the end of the time series, falling most of it in the test partition. In the case of the trend analysis, two of the datasets with the strongest trend component, 22ox and 22l8, performs better than the raw training.

Looking at the standard deviation there is a large variability between the lagged preprocessing and the other ones, including the experiments made with raw time series. In a general sense, both the trend and the seasonal analysis have a standard deviation comparable with the raw time series experimentation. The unusual variability on the 22ox dataset may be explained by the stabilization of the overall trend observed at the last 5 years of the time series.

5 Conclusion

LSTM neural networks are being actively used recently, while being proposed time ago. In this work, a preliminar experimentation has been conducted to quantify how much this kind of networks are affected by the use of several preprocessing methods in the context of time series forecasting.

Performing a trend deletion (differentiation) on the input time series usually has no effect on the accuracy, compared with a raw time series. A lagged dataset, unlike a differentiated one, can make the LSTM predict better in some cases, while severely worsening it in other cases. Further work would be needed to determine theses cases. On the other hand, a seasonal component removal has achieved an important accuracy gain on all the datasets considered, dropping the error below a 50% of the raw time series error in many cases. Our results show that special care has to be taken with lags and differentiating preprocessing methods, in which a severe performance drop has been seen with some datasets. In the case of the seasonal component removal preprocessing method, a more robust, accurate predictive behavior has been found.

Acknowledgement. This work is partially supported by the Spanish Ministry of Science and Technology under project TIN2015-68454-R.

References

1. Box, G.E., Jenkins, G.M.: Time Series Analysis: Forecasting and Control. Holden-Day Series in Time Series Analysis. Holden-Day, San Francisco (1976). Revised edition
2. Famili, A., Shen, W.M., Weber, R., Simoudis, E.: Data preprocessing and intelligent data analysis. Intell. Data Anal. **1**(1), 3–23 (1997). https://doi.org/10.3233/IDA-1997-1102
3. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: continual prediction with LSTM. Neural Comput. **12**(10), 2451–2471 (2000). https://doi.org/10.1016/j.neunet.2014.09.003
4. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Netw. **18**(5–6), 602–610 (2005). https://doi.org/10.1016/j.neunet.2005.06.042
5. Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J.: LSTM: a search space odyssey. IEEE Trans. Neural Netw. Learn. Syst. **28**(10), 2222–2232 (2017). https://doi.org/10.1109/TNNLS.2016.2582924
6. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al.: Gradient flow in recurrent nets: the difficulty of learning long-term dependencies (2001)
7. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
8. Hyndman, R.J., Akram, M.: Time series data library (2010). http://robjhyndman.com/TSDL
9. Hyndman, R.J., Khandakar, Y., et al.: Automatic time series for forecasting: the forecast package for R. No. 6/07, Monash University, Department of Econometrics and Business Statistics (2007). https://doi.org/10.18637/jss.v027.i03

10. Kotsiantis, S., Kanellopoulos, D., Pintelas, P.: Data preprocessing for supervised leaning. Int. J. Comput. Sci. **1**(2), 111–117 (2006)
11. Werbos, P.J.: Generalization of backpropagation with application to a recurrent gas market model. Neural Netw. **1**(4), 339–356 (1988). https://doi.org/10.1016/0893-6080(88)90007-X