Corrector LSTM: Built-in Training Data Correction for Improved Time Series Forecasting

YASSINE BAGHOUSSI, INESC TEC, Faculdade de Engenharia, Universidade do Porto
CARLOS SOARES, Fraunhofer AICOS Portugal and LIACC, Faculdade de Engenharia, Universidade do Porto
JOÃO MENDES-MOREIRA, INESC TEC, Faculdade de Engenharia, Universidade do Porto

Errors in the data are an important challenge for predictive ML algorithms. They have a negative impact on the forecasting accuracy because (a) they are hard to predict, as they do not necessarily follow a pattern, and (b) in the recurrent model, they may lead to changes in the model and thus increasing the errors in future predictions. In this paper, we address the latter challenge. We introduce corrector LSTM (cLSTM), a variant of LSTM that (a) detects errors in the data by analysing the hidden states of the LSTM, and (b) adjusts the training data accordingly. We start by empirically validating the assumption of the approach, i.e. that the analysis of the hidden states of LSTM can be used to detect data errors. Additional experiments also confirm that the approach improves the forecasting accuracy on artificial, NAB, and M4 Competition data.

CCS Concepts: • **Computer systems organization → Computer Science**.

Additional Key Words and Phrases: Time series forecasting, recurrent neural networks

ACM Reference Format:
Yassine Baghoussi, Carlos Soares, and João Mendes-Moreira. 2022. Corrector LSTM: Built-in Training Data Correction for Improved Time Series Forecasting . In *8th SIGKDD International Workshop on Mining and Learning from Time Series – Deep Forecasting: Models, Interpretability, and Applications, Aug 15th, 2022, Washington DC*. ACM, Washington DC, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

It is widely admitted that the data quality affects the machine learning (ML) model performance, and a significant amount of effort is spent by practitioners on cleaning data before training their models. The quality of ML models is only as good as the quality of the data they are trained on. Data cleaning has been essential for building high-quality ML models. Not surprisingly, both ML and database communities have been working on low-quality data problems over the last decades. On the one hand, the database community has mostly focused on understanding the fundamental data cleaning process without considering its impact on ML models. On the other hand, the ML community has been focusing on understanding the effect of errors such as anomalies, noise, concept drifts etc. on ML models without actually doing data cleaning [18]. Recently, however, a few data cleaning techniques have been developed to improve data quality and so model accuracy. For instance, ActiveClean is a framework that iteratively cleans samples of dirty data and updates the model.

The core idea of recurrent neural networks (RNN) is to allow passing information from the past to the next observations using the hidden states. The latter have led RNN to be universal approximators, data-driven, and able to better capture nonlinear patterns in data [14]. They have been subject to several improvements both in usage (e.g. autoencoders) and computation (LSTM [11]). Following their nature, the learning at each time step is, therefore, influenced by the previously processed inputs. The computed activations contain Hidden States Dynamics (HSD) which describe the processed information. When any error or disruption appears in the data, the performance of the sequential model may be highly affected both in the currently processed instance and on the following instances. The error, if it exists, flows through the hidden states. By analyzing these dynamics, we may detect the disrupting data when it occurs. Besides anomalies and other statistical problems, sequential algorithms may also be influenced by data that are considered normal but negatively influences the processing of information.

RNN's variant, LSTM, has an output gate that contains information about how the model processed the data. We focus on analyzing the dynamics occurring on this gate to detect errors over time. It will allow us to detect errors in data and reconstruct it accordingly. The output gate is mainly producing the output of the LSTM cell as follows:

$$o_t = \sigma(W_o \cdot h_{t-1} + V_o \cdot x_t + b_o) \tag{1}$$

Where W_o is the non-recurrent weight matrix and V_o is the recurrent weight matrix (also called hidden-to-hidden weights). The observation x_t, strongly affects the output gate output. When a normal observation occurs, the output gate process and pass it to the following timestamp. Thus, causing a slight change in the current model. However, when the observation is affected by some errors in the data generation process, the changes made to the model may significantly mislead the following observations. In this paper, we first demonstrate that this phenomenon can be observed in the dynamics of the hidden states and can be defined as constant data error (CDE) flow.

As a solution to this, we propose the *corrector LSTM* (cLSTM) algorithm to improve forecasting tasks where the data may be affected by errors. The goal is to render LSTM data-centric. cLSTM is an LSTM that uses a SARIMA model to predict the activations of the hidden output gate at each time step. Then, it computes the similarity between the predicted and the learned activations using a distance measure. If the distance exceeds a given detection threshold, an error is detected. The algorithm will immediately start reconstructing the data at this time step. The data is changed until

Conference acronym 'Milets, Aug 15, 2022, Washington DC

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

the distance between the two activations is below a reconstruction threshold. The latter represents the minimum tolerated distance between predicted and learned hidden states.

This paper starts by giving state-of-the-art improvements made on RNN and introducing LSTM. Followed by summarizing existing work on data reconstruction using RNN. Then, it introduces hidden states dynamics and corrector LSTM. Finally, the experimental setup is described, and the results are discussed.

The main contributions of the paper can be summarized as follows:

(1) It shows that Hidden States Dynamics can be used to improve data quality;
(2) It presents a method, corrector LSTM, which uses HSD to detect and reconstruct data to make standard LSTM more robust;
(3) It shows the validity of the proposed method based on some empirical results.

2 BACKGROUND

While data collection and quality issues have always been very important, machine learning research has mainly focused on training algorithms instead of improving the data. In the industry, a common complaint [28] is that research institutions spend 90% of their machine learning efforts on algorithms and 10% on data preparation, although based on the amounts of time paid, the numbers should be 10% and 90% the other way [33]. More recently, data-centric AI [1] is becoming popular where the primary goal is not to improve the model training algorithm but to improve the data pre-processing for a better model accuracy.

However, as the pre-processing is separated from the model training. Data quality may still be an issue during model training even after collecting the right data and cleaning it. It is widely recognized that real-world datasets are erroneous despite the data cleaning process ([34]; [25]). These faults in datasets can be grouped depending on whether data values are anomalies such as (noise, missing information etc.) and depending on whether these flaws exist in data features (attributes) or labels.

To combine the data quality with the learning, data cleaning techniques with the specific purpose of improving model accuracy have emerged [6]. For instance, ActiveClean is a seminal framework that iteratively cleans samples of dirty data and updates the model (see: Figure 1). This framework constrains the changes in the data to the model feedback. Thus, the data quality is improved according to the model performance. In [32], authors use prediction errors to detect anomalies using a threshold. When the model produces a prediction error of an RNN higher than a threshold, the input is considered an anomaly. However, this method waits until the model is trained completely to detect anomalies. While in RNN, an anomaly can impact future observation, leading to false-positive alerts. Also, this paradigm is made through two stages (i.e. model training and anomaly detection).

2.1 Long Short Term Memory

Long-Short Time Memory (LSTM) is a Recurrent Neural Network (RNN) that can more effectively learn long-term interactions in the data. RNNs differ from feed-forward neural networks by the recurrent connections, which allow them to learn from sequential data. They attempt to model and remember temporal dependencies in sequences. RNNs have been criticized due to the well-known problem of vanishing and exploding gradients [10]. These networks are not suited to learn long term dependencies in a sequence. They are not effective at learning more than 5 to 10-time steps apart in data [27]. LSTM overcomes this issue by introducing a gating mechanism to RNN [11]. The input, output, and forget gate give LSTM the ability to remember and to forget. They are implemented through specific mathematical transformations (Eq. 2b-2d), producing states known as hidden, h (Eq. 2g) and cell, c (Eq. 2f) states. These states allow the network to retain information from previous time steps and combine it with the current one. As data is fed into the network, unnecessary information will be declined using a sigmoid function. The information kept is combined with the new information entering the network.

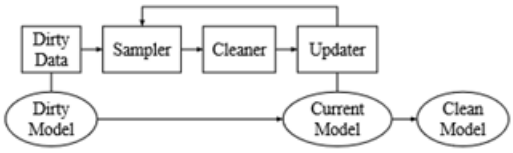

Fig. 1. ActiveClean [15] iteratively selects data that is likely to be dirty and cleans it.

$$i_t = \sigma(W_i \cdot h_{t-1} + V_i \cdot x_t + b_i) \tag{2a}$$
$$o_t = \sigma(W_o \cdot h_{t-1} + V_o \cdot x_t + b_o) \tag{2b}$$
$$f_t = \sigma(W_f \cdot h_{t-1} + V_f \cdot x_t + b_f) \tag{2c}$$
$$\hat{C_t} = tanh(W_c \cdot h_{t-1} + V_c \cdot x_t + b_c) \tag{2d}$$
$$C_t = i_t \cdot \hat{C_t} + f_t \cdot C_{t-1} \tag{2e}$$
$$h_t = o_t \cdot tanh(C_t) \tag{2f}$$
$$z_t = h_t \tag{2g}$$

At each time t, a cell receives an input x_t, together with the previous hidden state h_{t-1} and other internal cell data, to produce the next instances of the hidden and memory states via both feed-forward and recurrent connections. So, at each time step, the LSTM cell outputs a hidden state (Eq. 2g) and a cell state (Eq. 2f). The hidden state h_t can either be the network's final output or, in the case of multiple layers, subsequently, be fed to another LSTM cell (i.e. instead of the x_t). The last hidden vector is often fed through a linear, fully connected layer to produce the predictions. The output is a vector of values in sequence learning.

A single LSTM cell cannot handle learning, thus several cells must be used and organized into a specific network architecture. There are several common architectures of LSTM. First, the uni-directional LSTM consists of replacing the recurrent cells in the RNN network with LSTM cells. The latter architecture is trained in a one-time direction (i.e. forward). [8] introduced the bi-directional LSTM, which

is trained in both time directions simultaneously. The authors replaced recurrent cells in bi-directional RNN network [26] (also called bi-vanilla) with LSTM cells. This architecture has been shown to perform better than the feed-forward recurrent network in several domains such as text translation [19]. Network architectures have often been necessary to solve particular tasks. For instance, authors in [31] used LSTM networks to train a sequence to sequence model (namely seq2seq). This architecture is commonly known as the encoder-decoder. The encoder is often used to capture the context of the input sequence, and the decoder uses the hidden states from the encoder to produce the output sequence. One potential issue with encoder-decoder is that the encoders need to compress all the necessary information of a source sample into a fixed-length vector, making it difficult for the neural network to deal with long samples. [2] introduced an attention mechanism to preserve the context by focusing on different parts of the input sequence. There are two major architectures of attention models Bahdanau's architecture [2] and Luong's architecture [20]. As aforementioned, this has produced several variants that updated the LSTM cell or changed the network architecture. With the widespread of interpretable machine learning, several attempts have been conducted on RNN. They often consider hidden states analysis for prediction interpretability [29]. Works in this area can be categorized into two groups: attention methods and post-analyzing on trained models (e.g. prediction error analysis). Attention methods are mainly applied to hidden states across time steps and has gained tremendous popularity ([5];[9];[16]; [24]; [29]). These works explore the structure of LSTM recurrent neural networks to learn variable-wise hidden states. The goal is to capture different dynamics in multi-variable time series and distinguish the contribution of variables to the prediction. To the best of our knowledge, hidden states analysis has not been used to improve data quality.

2.2 Data-centric Recurrent Neural Network

Data reconstruction is the task of correcting issues present in the data to help models learn a correct representation of the data. The data collection process generates various types of faulty data, such as bias, drift, missing information, or anomalies. Some faults (e.g. missing data) are relatively easy to detect. In contrast, other faults (e.g. anomalies) are somewhat difficult to identify. Like any different forecasting model, Recurrent neural networks are sensitive to faults in data. Despite this, few works addressed data reconstruction using RNN. One notable variant of RNN, namely, dLSTM [22], relied on the predictive errors generated from a model trained on normal data (i.e. non-anomalous) to detect anomalies. The model use delayed prediction errors to measure deviation from the normal state. The model selects the value closest to the normal state from several candidate values to reconstruct this latter. A similar approach has been used in [13]. However, the prediction errors were derived from a bi-directional LSTM, which measures past and future correlations between observations. Moreover, other RNN architectures have also been shown to be effective such as stacked LSTM [21] and autoencoder [17]. To avoid relying on prior assumptions (i.e. the distribution of prediction errors), authors in [23] combined an autoencoder based on LSTM with a one-class support vector machine (OCSVM) algorithm to separate anomalies from the data outputted based on the LSTM autoencoder network. A recent approach [3] suggests sharing the responsibility of prediction errors between the network weights and the data. At the end of each epoch, the computed derivatives are used to alter each observation in the input sample. Instead of using prediction errors at the end of the training, this method suggests detecting the model failures (i.e prediction errors) during the learning. Following this idea, we suggest analyzing hidden states dynamics to detect data samples that lead to the model failures and reconstruct the data to avoid its influence on future trained examples, i.e., overall learning.

3 HIDDEN STATES DYNAMICS ANALYSIS

The core assumption of the data reconstruction method proposed here, cLSTM, is related to the hidden state dynamics. Standard LSTM networks capture and store information in the hidden states. LSTM uses this information to predict the future and considers it correct and relevant. However, input data has situations where it can mislead the learning of the model and so the predictions. Such is the case when the training model tries to capture complex components from the data and includes a set of observations that may not be suitable for the model. In the worst case, these can negatively affect the model and decrease the data modelling.

3.1 Hidden States Dynamics

Hidden States Dynamics (HSD) correspond to the changes in the properties of the hidden state over time [29]. In an LSTM, the dimension of hidden states vectors corresponds to the number of hidden units. In other words, at each time step, a cell outputs a hidden state vector of length equal to the number of hidden units. The analysis of HSD has been used to interpret RNN's models, which, similarly to feed-forward neural networks, are considered a black box. Hidden states vectors are commonly encoded as a heatmap along the time axis. This representation has been favoured to view the complete hidden state vectors $h_1, ..., h_T$. Although effective for interpretability (especially in a moderate dimensionality), heatmaps suffer limitations when further operations are needed [30]. For instance, with the increasing dimensionality of hidden state vectors, the heatmaps scaling becomes unnecessary because they scale the hidden state values by a colour hue, making it non-effective for extracting the most important information. Moreover, heatmaps consider a relative order of hidden states in each vector, but the model does not consider this particular order. Authors in [30] encode HSD using the changes of Hidden States values (y-axis) during time (x-axis) see Figure 2-Left. These changes are regarded as activation signals. When an LSTM network of N hidden units processes a time series, N signals are produced at each time step. They are dominated by the seasonality of length equal to the input length and LSTM size at each iteration. This seasonality summarizes the information captured from the input sequence.

Figures-2-top and 2-bottom show the output gate activations learned from one time series during different periods. As it can be verified on its corresponding time series on the right, the hidden output states in Figure 2-top did not report any significant change in the signal during the time. In Figure 2-bottom, the signal has

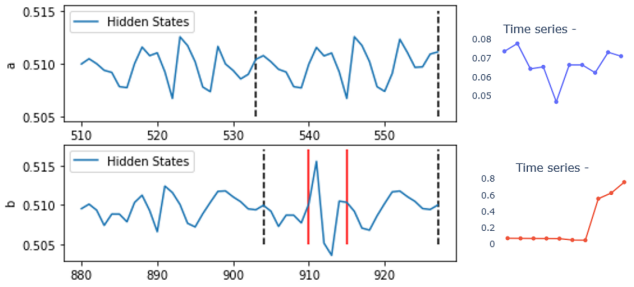

Fig. 2. The time series in the left corresponds to the hidden states values of two data samples (top and bottom). On the right, we have the corresponding data samples. Dashed lines correspond to the end of a training iteration of data sample at a particular time step. Top shows no significant change in the hidden states values during the time, in the seasonality more particularly. In bottom, the hidden states have partly changed (red lines). We assume that this phenomenon corresponds to errors of the current input sequence at the corresponding time step. The x-axis of the hidden states corresponds to the input position at each iteration and depends on the input length and LSTM size, (top) without anomalous behaviour and (bottom) with anomalous behaviour.

partly changed, in the seasonality more particularly, due to the influence of the current input sequence at the corresponding time step. Hidden states (including hidden-to-hidden) signals represent the data at each timestep following what has been learned previously. Over time these hidden states change seasonally when the input is considered normal for the model learning. However, when the input is unexpected or irrelevant to the model, LSTM is forced to adapt its learning to it, and that causes a change in the hidden state signal seasonality as was reported in Figure 2-bottom. Therefore, we can detect when learning goes wrong by analyzing these signals.

4 CORRECTOR LSTM

To test whether changes in the hidden states dynamics correspond to errors in the data sample. We propose corrector LSTM. This LSTM variant analyses the signals as follows: it learns a Seasonal ARIMA model that forecasts the values of the hidden states at each time step simultaneously during the learning of LSTM. A naive approach can substitute this part (i.e., using the previous time step). cLSTM, then, uses one of the similarity measures to quantify the difference between the forecast and the actual hidden states signal. A threshold σ is used to detect changes in the signal: a similarity measure higher than σ informs LSTM of a candidate for reconstruction. The corresponding point is recognized, and cLSTM starts the reconstruction.

Let $x = (x_1, x_2, \cdots, x_T)$ be a sequence drawn from a distribution $\mathcal{D}_{data}$. At each iteration, a subsequence is observed by LSTM, which uses that to update all its units through forward-passing and computes the error signal for all its weights (backward pass). For a subsequence length of k, an observation x_t is activated by input and passed to other gates. The output gate over the range $[t-k, t]$ can be represented as follows:

$$net_{out} = \sum_{p=t-k}^{t} \sigma(W_o \cdot h_p + V_o \cdot x_{p+1} + b_o) \tag{3}$$

where W_o is the hidden-to-hidden weight matrix associated with gate $_o$. V_o is the non-recurrent weight of the LSTM. W_o and V_o are updated after each iteration. An iteration consists of a forward pass and a backpropagation cycle. If an anomalous behaviour occurs at time step t, a constant data error (CDE) will gradually flow over time [7]. CDE can be represented as follows:

$$net_{out} = \sum_{p=t-k}^{t} net_{out,p+1} + \epsilon_p \tag{4}$$

Where ϵ_p represents the CDE caused by the learnt weights related to disrupting point from the previous iteration.

4.0.1 Seasonal ARIMA. Seasonal Auto Regressive Integrated Moving Average is an adaptive ARIMA model used when the timeseries exhibits seasonal variation. ARIMA is defined using (p,d,q) parameters, also called the ARIMA order. d is the level of differencing, p is the autoregressive order, and q is the moving average order [4]. The ARIMA model is defined as follows:

$$z_t = \delta + \phi_1 z_{t-1} + \phi_2 z_{t-2} + ... + \phi_p z_{t-p} + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - ... - \theta_q a_{t-q}$$

where z_t is the level of differencing, the constant is denoted by δ, while ϕ_i is an autoregressive operator, a_i is a random shock corresponding to time period t, and θ_i is a moving average operator.

SARIMA adds to ARIMA an order $(P, D, Q)_s$ which corresponds to seasonal autoregressive (P) and a seasonal moving average notation (Q). The variable s indicates the length of the seasonal period. For example, with an input sequence of 20 observations, s would be 20.

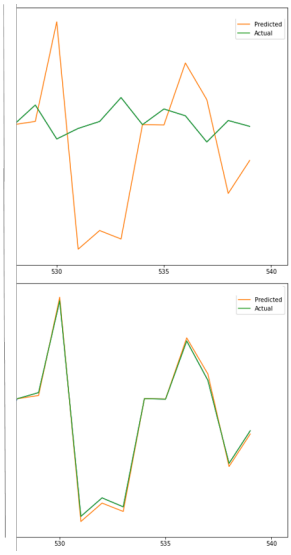

Fig. 3. Examples of hidden state dynamics with errors in data (top) and without errors (bottom). The figure also shows the HSD before and after correction at the same timestamp.

4.1 Similarity Measure

cLSTM detects errors in the data by predicting the values of the hidden states and comparing them to the observed ones. Therefore, it is necessary to measure the similarity between these two series. Therefore, Euclidean point-by-point mapping approach has been used.

4.2 Data Reconstruction

As aforementioned, hidden states dynamics can instantly inform us about the conflicts/problems occurring when the model processes the input data. Therefore, in order to make LSTM robust to errors, we reconstruct every data point that disrupts the model. We do this by changing the training inputs that lead to drifts in the hidden state. This implies that when an error is detected, the corresponding data point is changed accordingly. The learning process is repeated until the difference between the observed hidden states and their predicted values is below the defined reconstruction threshold δ. This objective function can be defined as follows

$$\min Euclidean(p_t, h_t)$$

Where p_t is the predicted output gate activations and h_t is the actual output gate activations.

The LSTM model is saved, and after every change to the data, the model is loaded and re-trained on the new value until the stopping condition is satisfied. For the reconstruction process, we use a different threshold than the detection threshold (which is smaller than the one used for detection). Thus, we can confirm that the new value is not causing any harm.

5 EXPERIMENTAL SETUP

To evaluate cLSTM, we have compared its performance with standard LSTM on multiple timeseries. Experiments assumed that the algorithms received a univariate time series as training data. Multi-time step samples are used to predict multi-time step labels during the learning process. At the end of the training, the outputs are the network's weights and the corrected time series. Since cLSTM has been designed to improve data quality, we have also tested the algorithm's ability to detect and reconstruct anomalies using Numenta Anomaly Benchmark.

5.1 Research questions

We identify the following as the main research question of our empirical study:

- Does cLSTM avoids incorrect learning of the LSTM when data contains errors ?

5.2 Baseline algorithms

The procedure just described was the main methodology developed to conduct our study. LSTM is the explicit competitor of cLSTM, so in this study we compare cLSTM solely to it.

Both LSTM and cLSTM have been executed on the same datasets.

Table 1. Descriptive analytics of M4 Forecasting Competition Datasets.

	M4			
	Daily	Hourly	Monthly	Weekly
Timeseries	200	200	200	200
Average Length	558	700	366	1255
Mean	2978	19935	4222	3925
Standard Deviation	688	3621	1160	1998

Table 2. Descriptive analytics of Numenta Anomaly Benchmark Datasets.

	NAB					
	RKC	Traffic	EX	AWS	Art	Tweets
Timeseries	7	7	5	17	9	10
Average Length	9872	2236	-	3980	4030	15854
Mean	40.18	124.73	-	742771	27	42.74
Standard Deviation	6.88	125.03	-	3.1e+06	26.84	98.74

5.3 Datasets

We have used 11 datasets (a total of 855 time-series) falling under a variety of application domains. We have used the **Macro M4 competition** datasets (a total of 800 time-series).[1] The dataset includes six subsets, from which we have taken four datasets and 200 timeseries from each. Additionally, we have used 55 signals from **Numenta Anomaly Benchmark** (NAB). NAB includes multiple types of time series data from various application domains. We have picked six datasets: Art, RKC, AWS, Traf, and Tweets. These signals are from different sources and contain different quantities of real continuous anomalies. Basic information for each dataset is summarized in Table 2. Except for min-max normalization, no pre-processing has been applied. Moreover, we have used 1000 observations from each time series to reduce computational costs.

5.3.1 cLSTM Architecture. The LSTM version used for applying our changes is a PyTorch implementation. For each iteration, the model forward and backwards pass an input of length one and outputs hidden states vector of length (i.e. LSTM size) = 12 per unit, which is further used for analysis. The outputs are taken from a selected LSTM unit. 50 was the number of epochs, and only one LSTM layer has been used. To detect errors, we chose to analyze the hidden states dynamics produced by a single unit. The errors can only be detected after given iterations so that SARIMA model has minimum hidden states to learn.

We have run experiments using a fixed detection threshold. In this work, no fine-tuning of this parameter has been conducted. Thus, no analysis will be shown regarding the impact of such a parameter. It would be interesting for future works to analyze it, especially in cases where cLSTM fails against LSTM.

5.3.2 Evaluation Metrics. We compare our model performance using both Mean Absolute Scaled Error [12], Mean Square Error (MSE) and percentage decrease on MSE. As the timeseries have different scales, we have used MASE using the training set of the original series. Moreover, Wins have been reported instead of MSE.

[1] https://mofc.uniac.cy/the-dataset/

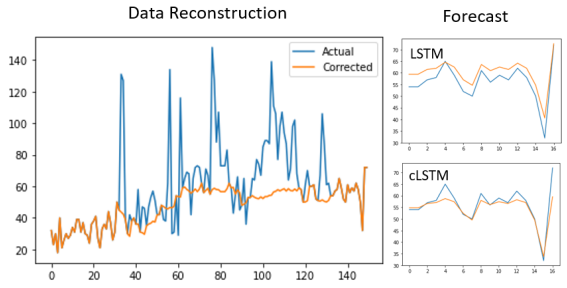

Fig. 4. Example 1 - Timeseries reconstruction example on NAB. Blue is the original time series, orange is the corrected timeseries. Right plots show the predictions (orange) and actual values (blue) with the train data from the left plot being the corrected for cLSTM and actual for LSTM.

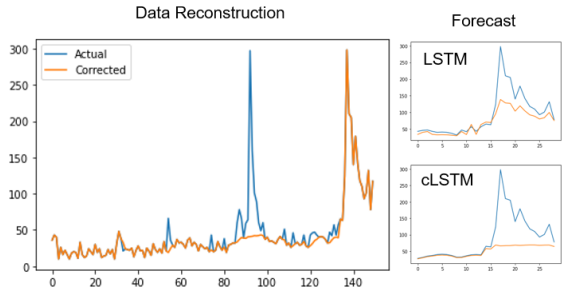

Fig. 5. Example 2 - Timeseries reconstruction example where error is recurring in the test set. Blue is the original time series, orange is the corrected timeseries. Right plots show the predictions (orange) and actual values (blue) with the train data from the left plot being the corrected for cLSTM and actual for LSTM.

Generally, the input impact on the learning occurs over a limited interval (i.e. a continuous sequence of points). Since a wrong reconstruction might decrease forecasting performance, it is important to avoid false positive FPs. As a result, the optimal detection threshold needs to be fine-tuned. As aforementioned, in this work, we do not address this. Moreover, as the change points of concept drifts can be considered an error, cLSTM may correct the whole new distribution. In this work, we did not address this type of error, which might explain some of the results showing a negative impact of cLSTM.

Also, as our learning mechanism is sequential, each iteration forwards past information to the next points, which causes the propagation of reconstruction errors. Overcoming this requires choosing a moderate reconstruction threshold. We have limited the search for an optimal reconstruction value to a fixed reconstruction threshold.

Table 3. Number of wins based on MSE by timeseries on NAB.

Wins (MSE)	Art	EX	AWS	RKC	Traffic	Tweets
LSTM	3	2	7	6	3	2
cLSTM	1	3	10	1	4	7

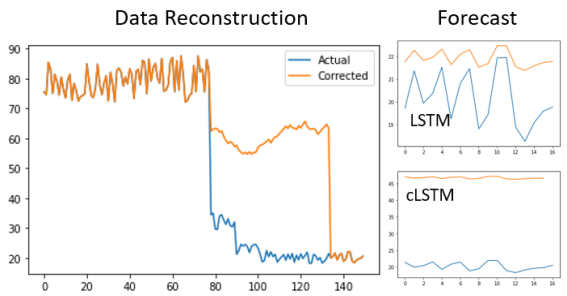

Fig. 6. Example 3 - Timeseries reconstruction example with a **concept drift**. Blue is the original time series, orange is the corrected timeseries. Right plots show the predictions (orange) and actual values (blue) with the train data from the left plot being the corrected for cLSTM and actual for LSTM.

Table 4. Number of wins based on MSE by timeseries on M4.

Wins MSE	Daily	Hourly	Monthly	Weekly
LSTM	188	166	113	116
cLSTM	106	55	108	83

Table 5. NAB - Percentage decrease on MSE errors by cLSTM.

Art	EX	AWS	RKC	Traffic
85.761130	27.782211	19.677456	2.801852	27.974179

Table 6. M4 - Percentage decrease on MSE errors by cLSTM.

Daily	Hourly	Monthly	Weekly
21.503485	33.710400	23.705499	24.934199

5.4 Experimental Results

The experiments illustrate two forecasting results: M4 competition and NAB. NAB and M4 experiments were conducted by comparing a PyTorch LSTM and cLSTM implementations with 50 epochs trained on the original data. A seed has been set for the results to be comparative. As a response to whether cLSTM is more robust to errors than LSTM, results of the training on M4 competition data in Table 4 show cLSTM winning over standard LSTM in a significant number of timeseries. As for NAB, Table 3 reports the number of wins of each method with a higher number of wins achieved by cLSTM in 4 from 6 datasets.

As aforementioned, cLSTM has a hyperparameter that allows the detection of error and triggers the correction of data. This hyperparameter has been kept fixed in all the experiments presented in this work. Figure 4 shows some instances of timeseries reconstruction on real Twitter timeseries in NAB. It is worths noting that cLSTM depends on the first observations and epochs to decide whether an input contains an error. Thus, in cases where a supposed error is recurring in the future, the model would be unable to forecast it (see: Figure 5). The forecasting results are given on the right of the figure. LSTM was trained on the actual (blue) timeseries on the left. While cLSTM has been trained on its corrected (orange) timeseries

Table 7. NAB - MASE values of cLSTM and LSTM.

	Art	EX	AWS	RKC	Traffic	Tweets
MASE LSTM	0.3856	1.010880	0.556056	5.167250	0.552625	0.697960
MASE cLSTM	0.1455	1.001771	0.569140	5.602950	0.632668	0.689308

Table 8. M4 - MASE values of cLSTM and LSTM.

	Daily	Hourly	Monthly	Weekly
MASE LSTM	1.984319	1.271852	1.683081	2.552384
MASE cLSTM	1.941947	1.285469	1.623858	1.818637

on the left. These examples were not excepted by our experiments which explain the number of losses of cLSTM. Moreover, the model detection of error is shown to be efficient on detecting anomalies and concept drifts. However, the reconstruction strategy has to be adapted for them.

M4 and NAB datasets contain timeseries with concept drifts (see: Figure 6). In this work, cLSTM is not adapted to such a case. It can detect the change point but continues to change all the new distribution data. The latter leads to a model failing to predict the new distribution. These examples were not excepted by our experiments which explain the number of losses of cLSTM.

To evaluate cLSTM potential, we have analysed the cases where cLSTM won over LSTM, which we consider are, most of the time, timeseries without concept drifts or recurring errors. Table 5 and 6 show the percentage decrease on MSE made by cLSTM compared to LSTM. For NAB, cLSTM has reduced the MSE by 85% on artificial data with errors. It can be explained by artificial errors, which can be considered anomalies and easy to detect and reconstruct. As for other datasets in NAB, the decrease was around 32% on average. In M4 datasets, the decrease was around 25.95% on average.

On the other hand, MASE results in Table 7 and 8 show a lower cLSTM MASE over LSTM in 3 over six datasets of M4 and 3 over four datasets of NAB. Based on our understanding, MASE results can be explained by the fact that cLSTM is not changing the global performance of LSTM. However, it eliminates the most important errors.

6 CONCLUSIONS

Recurrent neural networks can be affected by errors made by the data collection system. Making RNN more robust to errors requires making it able to detect and reconstruct them. One advantage of RNN is that it keeps a history of the processed information on its hidden states. The latter contains dynamics that can give useful model feedback on the data quality. This paper presents a new method, namely, corrector LSTM, which uses the dynamics of the hidden states to detect errors in the data and reconstruct them accordingly. cLSTM uses a SARIMA model responsible for the predictions of the hidden states. These predictions are compared with the hidden states produced by LSTM using a distance measure. If the measurement exceeds a detection threshold, the method decides to change the processed data. The reconstruction is carried out until the errors in the data are no longer detected. The process stops changing the data when the distance equals a given reconstruction threshold. The method has been compared to standard LSTM on several datasets. The results show that cLSTM could significantly improve data quality without affecting the LSTM global performance as reported by MASE results.

ACKNOWLEDGMENTS

This work has been partially supported by the SONAE IM LAB@FEUP, under a research Ph.D. project funded by Inovretail.

REFERENCES

[1] 2021. Data-centric ai competition. (2021). https-deeplearning-ai.github.io/data-centric-comp.
[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
[3] André Baptista, Yassine Baghoussi, Carlos Soares, João Mendes-Moreira, and Miguel Arantes. 2021. Pastprop-RNN: improved predictions of the future by correcting the past. *CoRR* abs/2106.13881 (2021). arXiv:2106.13881 https://arxiv.org/abs/2106.13881
[4] Bruce L Bowerman and Richard T O'Connell. 1993. Forecasting and time series: An applied approach. 3rd. (1993).
[5] Heeyoul Choi, Kyunghyun Cho, and Yoshua Bengio. 2018. Fine-Grained Attention Mechanism for Neural Machine Translation. *CoRR* abs/1803.11407 (2018). arXiv:1803.11407 http://arxiv.org/abs/1803.11407
[6] Xin Dong and Theodoros Rekatsinas. 2018. Data integration and machine learning: a natural synergy. *Proceedings of the VLDB Endowment* 11 (08 2018), 2094–2097. https://doi.org/10.14778/3229863.3229876
[7] Alex Graves. 2012. *Long Short-Term Memory*. Springer Berlin Heidelberg, Berlin, Heidelberg, 37–45. https://doi.org/10.1007/978-3-642-24797-2_4
[8] Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks* 18, 5-6 (2005), 602–610.
[9] Tian Guo, Tao Lin, and Yao Lu. 2018. An interpretable LSTM neural network for autoregressive exogenous model. *CoRR* abs/1804.05251 (2018). arXiv:1804.05251 http://arxiv.org/abs/1804.05251
[10] Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6, 02 (1998), 107–116.
[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[12] Rob J. Hyndman and Anne B. Koehler. 2006. Another look at measures of forecast accuracy. *International Journal of Forecasting* 22, 4 (2006), 679–688. https://doi.org/10.1016/j.ijforecast.2006.03.001
[13] Seongwoon Jeong, Max Ferguson, and Kincho Law. 2019. Sensor data reconstruction and anomaly detection using bidirectional recurrent neural network. 25. https://doi.org/10.1117/12.2514436
[14] Mehdi Khashei and Mehdi Bijari. 2011. A novel hybridization of artificial neural networks and ARIMA models for time series forecasting. *Applied Soft Computing* 11, 2 (2011), 2664–2675. https://doi.org/10.1016/j.asoc.2010.10.015 The Impact of Soft Computing for the Progress of Artificial Intelligence.
[15] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J. Franklin, and Ken Goldberg. 2016. ActiveClean: Interactive Data Cleaning for Statistical Modeling. *Proc. VLDB Endow.* 9, 12 (aug 2016), 948–959. https://doi.org/10.14778/2994509.2994514
[16] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. 2017. Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks. *CoRR* abs/1703.07015 (2017). arXiv:1703.07015 http://arxiv.org/abs/1703.07015
[17] Nikolay Laptev, Jason Yosinski, Li Erran Li, and Slawek Smyl. 2017. Time-series extreme event forecasting with neural networks at uber. In *International conference on machine learning*, Vol. 34. sn, 1–5.
[18] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2019. CleanML: A Benchmark for Joint Data Cleaning and Machine Learning [Experiments and

Analysis]. *CoRR* abs/1904.09483 (2019). arXiv:1904.09483 http://arxiv.org/abs/1904.09483

[19] G. Liu and J. Guo. 2019. Bidirectional LSTM with attention mechanism and convolutional layer for text classification. *Neurocomputing* 337 (2019), 325–338. https://doi.org/10.1016/j.neucom.2019.01.078 cited By 133.

[20] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).

[21] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. 2015. Long Short Term Memory Networks for Anomaly Detection in Time Series.

[22] Shigeru Maya, Ken Ueno, and Takeichiro Nishikawa. 2019. dLSTM: a new approach for anomaly detection using deep learning with delayed prediction. *International Journal of Data Science and Analytics* 8 (09 2019). https://doi.org/10.1007/s41060-019-00186-0

[23] H.D. Nguyen, K.P. Tran, S. Thomassey, and M. Hamad. 2021. Forecasting and Anomaly Detection approaches using LSTM and LSTM Autoencoder techniques with the applications in supply chain management. *International Journal of Information Management* 57 (2021), 102282. https://doi.org/10.1016/j.ijinfomgt.2020.102282

[24] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison W. Cottrell. 2017. A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction. *CoRR* abs/1704.02971 (2017). arXiv:1704.02971 http://arxiv.org/abs/1704.02971

[25] Yuji Roh, Geon Heo, and Steven Euijong Whang. 2021. A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective. *IEEE Transactions on Knowledge and Data Engineering* 33, 4 (2021), 1328–1347. https://doi.org/10.1109/TKDE.2019.2946162

[26] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing* 45, 11 (1997), 2673–2681.

[27] Ralf C Staudemeyer and Eric Rothstein Morris. 2019. Understanding LSTM–a tutorial into Long Short-Term Memory Recurrent Neural Networks. *arXiv preprint arXiv:1909.09586* (2019).

[28] Michael Stonebraker and El Kindi Rezig. 2019. Machine Learning and Big Data: What is Important? *IEEE Data Eng. Bull.* 42 (2019), 3–7.

[29] Hendrik Strobelt, Sebastian Gehrmann, Bernd Huber, Hanspeter Pfister, and Alexander M. Rush. 2016. Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks. *CoRR* abs/1606.07461 (2016). arXiv:1606.07461 http://arxiv.org/abs/1606.07461

[30] Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M Rush. 2017. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE transactions on visualization and computer graphics* 24, 1 (2017), 667–676.

[31] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. *CoRR* abs/1409.3215 (2014). arXiv:1409.3215 http://arxiv.org/abs/1409.3215

[32] Kim Phuc Tran, Huu Du Nguyen, and Sébastien Thomassey. 2019. Anomaly detection using Long Short Term Memory Networks and its applications in Supply Chain Management. *IFAC-PapersOnLine* 52, 13 (2019), 2408–2412. https://doi.org/10.1016/j.ifacol.2019.11.567 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.

[33] Steven Euijong Whang, Yuji Roh, Hwanjun Song, and Jae-Gil Lee. 2021. Data Collection and Quality Challenges in Deep Learning: A Data-Centric AI Perspective. *CoRR* abs/2112.06409 (2021). arXiv:2112.06409 https://arxiv.org/abs/2112.06409

[34] Steven Euijong Whang, Yuji Roh, Hwanjun Song, and Jae-Gil Lee. 2021. Data Collection and Quality Challenges in Deep Learning: A Data-Centric AI Perspective. *arXiv preprint arXiv:2112.06409* (2021).