Evaluation Techniques for Long Short-Term Memory Models: Overfitting Analysis and Handling Missing Values

Roland Bolboacă[1][0000−0002−4825−8786], Piroska Haller[1][0000−0002−5611−2429], and Bela Genge[1][0000−0003−1390−479X]

"George Emil Palade" University of Medicine, Pharmacy, Science, and Technology of Târgu Mureş, 540139 Târgu Mureş, Romania
{roland.bolboaca,piroska.haller,bela.genge}@umfst.ro

Abstract. Long Short-Term Memory models have demonstrated their effectiveness across various domains, providing solutions to numerous problems. In time-series data originating from process monitoring, Long Short-Term Memory models are particularly useful due to their capability to capture and represent spatial and temporal dependencies within the data. However, when it comes to modeling nonlinear systems described by differential equations, the relationship between input and output variables cannot be established solely by considering the current inputs. A well-known technique called Teacher Forcing enables the inclusion of previous output true values as additional inputs to Recurrent Neural Networks. However, applying this technique poses a significant challenge: the resulting models might tend to excessively rely on the previous true value, leading to predictions that replicate the previous value at each time step. As a solution, this paper presents a novel methodology to assess whether these models suffer from such overfitting to the previous true value. Additionally, this paper introduces a method to evaluate the robustness of these models against missing values, which may occur due to unforeseen events such as communication faults or erroneous sensor readings. The experimental results, conducted on the Tennessee Eastman Process Dataset, demonstrate the effectiveness of our proposed solutions in terms of overfitting tests and handling missing values.

Keywords: Long Short-Term Memory · Evaluation Metrics · Overfitting · Missing Values · Time-Series · Process Modeling

1 Introduction

The usage of Long Short-Term Memory (LSTM) models [12] has gained significant popularity in recent years due to their demonstrated effectiveness and improved performance across diverse domains including medicine [16], finance [5], natural language processing [28] and various industrial domains [26].

For neural networks to effectively capture the temporal dependencies and dynamics present in control systems, that can be described by differential equations, the current inputs alone may not be sufficient. In such instances, the

previous outputs or states of the modeled system may provide important information about the system's behavior and be necessary for accurate predictions of the subsequent outputs [18, 21]. In such systems, the previous output or system state can be seen as an additional input that helps the neural network capture the system's behavior and dynamics over time [8].

Two recent papers [2, 3] proposed a modified version of Long Short-Term Memory (LSTM) with Teacher Forcing (TF) [29], named LSTMTF, which utilizes the previous ground truth output values as input at each time-step during both training and inference. In this manner, LSTMTF effectively addresses a pair of challenges: the vanishing and exploding gradient problem [10], and the issue of Exposure Bias [25]. Their proposed solution is utilized for industrial systems modeling, restricted only by the availability of the output ground truth observations. Moreover, in [3] the same authors showed that such models can outperform the classical LSTM models, even with reduced architectures. However, one major issue with the above solution is that such models might overfit to the previous output ground truth value. Consequently, at each step, they might predict only close to the previous ground truth value, while ignoring the spatio-temporal relationships between other exogenous inputs.

To address this type of overfitting, further named *y-overfitting*, we are testing if feeding the previous output ground truth value as input, during training, causes the model to adjust the parameters in such a way that during inference each new prediction will be close or equal to the previous ground truth value. Specifically, this implies that the model becomes a simple naïve forecasting method [13]. Furthermore, in the current context, considering that the trajectory of nonlinear systems describing complex processes incorporates the previous system states together with the current system inputs, a naïve approach to prediction leads to unrealistic and unreliable results. In this direction, we propose an overfitting testing methodology inspired by sensitivity analysis [30]. To analyze the y-overfitting of the trained model, we generated different scenarios by sequentially disabling inputs or setting constant ground truth values for the output variable. This is followed by measuring the changes in the model's prediction error distribution on new and unseen data originating from the same generative process. Assuming that in such scenarios the model will naively predict the previous ground truth value every time, the distribution of the prediction error will not suffer changes, signaling that the model is y-overfitting. Our proposed approach outputs an overfitting score, measuring the distance between the prediction error distribution in various scenarios, using three distribution distance metrics.

Additionally, for the same model, it is assumed that the output ground truth values are available during both training and inference. Nevertheless, a significant issue is raised in our approach when the output ground truth values are missing due to unforeseen events (e.g., communication faults and erroneous sensor readings). As a solution to this issue, we are proposing an approach that does not require using any additional models for imputation, but rather use the same

model and switch between using the true output values to using the predicted values, for short periods.

The rest of the paper is structured as follows. Section 2 provides an overview of the background and related studies. Section 3 delineates our proposed approaches in detail. The experimental assessment methodology and the results are presented in Sections 4 and 5 respectively. Section 6 encompasses discussions pertaining to the results, and ultimately concludes the paper.

2 Background and Related Studies

In short, TF denotes a training algorithm for RNNs, where during training the output (e.g., response variable) ground truth value is fed as additional input to the model while during inference the model takes the prediction from the previous time step as an extra input. However, vanilla RNNs suffer from what is known as vanishing and exploding gradient [10]. Moreover, TF has its drawbacks, namely Exposure Bias. Exposure Bias appears as a result of a model being trained with the previous output ground truth value, but tested using the predicted value from the previous time step. As the distribution of the data that the model sees during training might differ from what it sees during inference causes the model to yield inaccurate and unreliable predictions. Despite the fact that TF has been applied in numerous fields [1, 15], there has been no investigation into the fact that LSTMs trained with TF might overfit to the previous ground truth value. However, authors who included models trained with TF in their work identified that such models might overfit the training data [15, 31] or might suffer from Exposure Bias [20].

Nguyen *et al.* [19] proposed an RNN based approach for predicting the transient behavior of a nonlinear electronic circuit. In their work, the authors suggest that TF might improve training convergence, but it might also lead to overfitting. Song *et al.* in [1] proposed an adversarial training scheme for autoregressive sequence generative models used for text generation. In this paper, the authors offer solutions for dealing with Exposure Bias and highlight the fact that TF might cause overfitting. Nicolai and Silfverberg [20] proposed a novel adaptation to the TF technique, named "student forcing". This approach involves substituting the model's predicted outputs with accurate labels for a portion of the training examples, to mitigate the adverse effects of Exposure Bias and overfitting. Wang and Lee [27] also found that Generative Adversarial Models, used for text prediction, might overfit the training data.

This paper addresses the challenge of y-overfitting in LSTMs, where the output target value is fed back as input during both training and testing. In comparison to previous works, our proposed solution introduces a quantitative evaluation on overfitting.

Moving on to related studies in the direction of missing values. In the past decades, this research field has been of high interest to a lot of researchers, and over the years various solutions have been proposed for handling missing values in data originating from multiple domains. The work of Enders, Craig K,

Applied Missing Data Analysis [9] offers a comprehensive overview of the field, including missing data patterns, missing data mechanisms, and also a systematic overview of the literature, analyzing the proposed solutions from the past 35 years. In a similar direction, other review papers offer comprehensive summaries of recent advances in the field [14]. These surveys point out different missing data patterns and two common approaches for dealing with missing values, namely conventional methods (e.g., ignoring or removing values) and modern methods (e.g., data imputation techniques, that is, replacing the missing values using various techniques).

Among recently proposed solutions for data imputation, we find the work of Han and Kang [11]. Here, the authors propose a dynamic imputation technique aimed at improving the training procedures of neural networks. The proposed approach involves the training of a neural network using a dataset that features dynamic imputations. Specifically, the model employs a layer that generates distinct values for identical missing slots during the training process. This methodology has the potential to improve the network's ability to generalize and accurately predict values for incomplete data points.

Cui *et al.* [6] introduced a stacked bidirectional LSTM network architecture for network-wide traffic state prediction. Their proposed solution involves the addition of an imputation unit inside the LSTM cell, prior to the four LSTM gates. This imputation unit estimates the missing values in each time step using the previous hidden and cell states. In a similar approach, Che *et al.* [4] proposed a Gated Recurrent Unit (GRU) that estimates missing values using a decay mechanism applied to the latest available value.

Lin *et al.* [17] compared the imputation performance of deep neural networks (i.e., Multilayer Perception, Deep Belief Network) with other imputation techniques, such as mean value replacement, K-Nearest Neighbors, Classification and Regression Trees, and Support Vector Machines. Their results revealed that the deep neural networks outperformed the other techniques on 14 datasets. In particular, between the two deep neural networks, the best performance was obtained by the Deep Belief Network. Furthermore, in the same paper, the authors proposed two differently ordered combinations of data discretization.

Compared to other solutions, we emphasize that our proposed metric can be used to quantify the effects of imputing missing values. This metric offers the possibility of comparing the prediction efficiency while using not only our imputation method but also other methods as well.

3 Proposed Approach

3.1 Long Short-Term Memory and Teacher Forcing

LSTMs incorporate memory units that are capable of learning long and short-term dependencies in time-series data. A standard LSTM layer is composed of blocks that incorporate memory cells and three gates (e.g., input, output, forget gates). Additionally, each block has two recurrent connections (e.g., hidden state

and cell state), where the hidden state and cell state represent the short-term and long-term memory respectively. The three gates regulate the information flow from and to the memory cell. As the name suggests, the forget gate controls the discarded information, the input gate regulates the information that is to be saved and finally, the output gate computes the current output of the cell.

When training RNNs using TF, a process is followed where the output target value from the previous time step, known as the ground truth output $y(t-1)$, is used as input for the current time step. However, during testing or inference, instead of using the ground truth, the network's own output $\hat{y}(t-1)$ is used as input. This approach of using the network's own outputs as input during testing has some drawbacks. One major issue is that the inputs seen by the network during training can be significantly different from the inputs encountered during inference, leading to a phenomenon called Exposure Bias. The original methodology of TF assumes that the ground truth values will not be accessible after the training phase. However, the model employed in this paper (e.g. LSTMTF), as originally proposed in [3], incorporates the use of the previous output ground truth value, denoted as $y(t-1)$, both during training and inference stages. For a more detailed description of LSTMs with TF, the reader can consult the following papers [2, 3].

3.2 Overfitting Tests

To develop the overfitting tests, we begin with the premise that LSTM models with TF overfit the previous output ground truth value. Consequently, during inference, these models predict close to the previous ground truth value, ignoring the spatio-temporal relationship between the rest of the inputs and the output variable, this is *y-overfitting*.

If a model y-overfits, it introduces the following assumptions. First, disabling any of the additional inputs would not have any influence on the model's performance, as it "relies" only on the previous output ground truth value to make new predictions. Second, disabling all additional inputs would not affect the model's performance, based on the same assumption as above. Third, setting the output ground truth value to a constant would not affect the performance, as the model would only predict close to the previous ground truth value.

We can formally define these assumptions as follows. Let X_e denote the vector of exogenous inputs, where $[x_1(t), x_2(t), ..., x_n(t)] \in X_e$ represent the n exogenous inputs at time t. The vector containing all the input variables, denoted as X, can be written as $X = [X_e, y(t-1)]$. Let $\hat{Y}$ denote the vector that contains the model's predictions, where $[\hat{y}(0), \hat{y}(1), ..., \hat{y}(t)] \in \hat{Y}$. Let Y denote the vector of ground truth values, where $[y(0), y(1), ..., y(t)] \in Y$. Likewise, let e and $\hat{e}$ denote the computed prediction error vectors between Y and $\hat{Y}$, where the former represents the model's measured error in normal conditions and the latter represents the model's measured error during the y-overfitting tests. Additionally, let $\mathcal{E}(e)$ and $\mathcal{E}(\hat{e})$ denote the distributions of e and $\hat{e}$ respectively. Assuming that the model was trained accordingly, both e and $\hat{e}$ will follow a Gaussian Distribution with a mean value close to zero.

Assumption 1 *If the model is y-overfitting then if any input sequence $x_i \in X$ is disabled and y is not changed then $\mathcal{E}(e) \approx \mathcal{E}(\hat{e})$, $\forall i \in [1, n]$.*

Assumption 2 *If the model is y-overfitting then if all input sequences $x_i \in X$ are disabled and y is not changed then $\mathcal{E}(e) \approx \mathcal{E}(\hat{e})$, where $i \in [1, n]$.*

Assumption 3 *If the model is y-overfitting then if all ground truth values $y(t) = c$ (constant), $\forall t$, and all $x_i \in X$ are not changed then $\mathcal{E}(e) \approx \mathcal{E}(\hat{e})$.*

To quantify the distance between $\mathcal{E}(e)$ and $\mathcal{E}(\hat{e})$, three distance metrics are used, namely (i) Energy Distance (ED) [24], (ii) Wasserstein Distance (WD) [22], and (iii) the Histogram Euclidean Distance (HD). The motivation behind choosing distribution distance metrics lies in the fact that even though both series (e.g., training and testing) originate from the same generative process and have the same distribution, the individual trajectories are inherently different. Therefore, measuring the point-by-point or sequence-by-sequence distance between the training and testing prediction errors yields unreliable results.

Algorithm 1: y-overfitting Tests

Input: S_{tr} - Training Dataset $(X_{tr}, Y_{tr} \subset S_{tr})$
S_{tst} - Testing Dataset $(X_{tst}, Y_{tst} \subset S_{tst})$
w - Prediction Window
$NrOfScenarios$ - y-overfitting Scenarios
DM - The Selected Distance Metric
Result: OS - y-overfitting Score

begin
  $mdl \leftarrow @TrainNetwork(S_{tr})$;
  $\hat{Y}_{tr} \leftarrow @Predict(X_{tr})$;
  $e \leftarrow Y_{tr} - \hat{Y}_{tr}$;
  $\theta \leftarrow @ComputeThreshold(mdl, e, S_{tst}, w, DM)$;
  for $i \leftarrow 1$ **to** $NrOfScenarios$ **do**
    $DV \leftarrow []$; /* Distance Vector */
    $index \leftarrow 1$;
    $mdl \leftarrow @InitializeScenario(mdl, i)$;
    for $j \leftarrow 1$ **to** $len(X_{tst} - w)$ **by** w **do**
      $\hat{Y}_{tst} \leftarrow @Predict(X_{tst}[j : j + w])$;
      $\hat{e} \leftarrow Y_{tst}[j : j + w] - \hat{Y}_{tst}[j : j + w]$;
      $DV[index] \leftarrow @Distance(DM, e, \hat{e})$;
      $DV[index] \leftarrow DV[index]/\theta$;
      $index \leftarrow index + 1$;
    end
    $OS[i] \leftarrow mean(DV)$;
  end
  return OS;
end

Algorithms 1 and 2 describe the developed y-overfitting tests and the decision threshold computation. Here, S_{tr} and S_{tst} denote the training and testing

datasets, respectively. Both datasets include the sets of inputs and outputs, so that $X_{tr}, Y_{tr} \subset S_{tr}$ and $X_{tst}, Y_{tst} \subset S_{tst}$. Given w a prediction window, a selected distance metric, and a testing scenario, the first Algorithm will output a y-overfitting score. Here, the testing scenarios represent the three assumptions from above.

As shown in Algorithm 1, the model is trained and tested on the training data set, this is followed by the computation of e (e.g., the training prediction error). In what follows in Algorithm 1, the function $@Distance()$ computes the distance between e and $\hat{e}$ using one of the distance metrics (i) – (iii) described above. The threshold computation is illustrated in Algorithm 2. This involves predicting over the testing dataset, using a w window, and computing a distance vector between the prediction error on each window and the prediction error from the training dataset. Finally, the threshold is set to be the maximum from all the computed distances.

Algorithm 2: Threshold Computation

Input: mdl - Trained Model
e - Training Prediction Error
S_{tr} - Testing Dataset ($X_{tst}, Y_{tst} \subset S_{tst}$)
w - Prediction Window
DM - The Selected Distance Metric
Result: θ - Threshold

begin
  $index \leftarrow 1$;
  $DV \leftarrow []$; /* Distance Vector */
  for $i \leftarrow 1$ **to** $len(X_{tst} - w)$ **by** w **do**
    $\hat{Y}_{tst} \leftarrow @Predict(X_{tst}[i : i + w])$;
    $\hat{e} \leftarrow Y_{tst}[i : i + w] - \hat{Y}_{tst}[i : i + w]$;
    $DV[index] \leftarrow @Distance(DM, e, \hat{e})$;
    $index \leftarrow index + 1$;
  end
  $\theta \leftarrow MAX(DV)$;
  return θ;
end

To compute the y-overfitting score, the changes to the dataset and the model are made according to each of the tested assumptions; this is shown in Algorithm 1 with the call $@InitializeScenario()$. Similarly to the threshold computation, a distance vector is computed between the prediction error, over the testing dataset, using the moving window w, and the training prediction error. Each resulting distance value is divided by the threshold θ. Finally, the y-overfitting score is computed as the average of the computed distances over the entire testing dataset.

3.3 Missing Values

Our proposed approach for dealing with missing values involves switching the same model from using the output ground truth value, as extra input, to using its previous predicted value, in real time, when the target values are missing. More formally, we consider a masking vector $m(t) \in \{0, 1\}$ which denotes the missing output ground truth values at time t. Specifically, we have:

$$m(t) = \begin{cases} 0, & \text{if y(t-1) is missing at time } t \\ 1, & \text{if y(t-1) is available at time } t. \end{cases} \tag{1}$$

At each time-step t, the model will take as extra input: $m(t)y(t-1) + (1 - m(t))\hat{y}(t-1)$. Here, depending on whether the previous true output value is available, the model will switch from using the true output previous value $y(t-1)$ to using the previous predicted output value $\hat{y}(t-1)$.

4 Experimental Assessment

The experimental assessment is split into four scenarios. In Scenarios I-III, the three assumptions formulated for y-overfitting are tested. Scenario IV deals with testing the model against missing values. Additionally, threshold computation is performed in the base reference scenario, where no modifications are made to the datasets. The threshold is computed using the testing datasets. All experiments were implemented in Python, utilizing Keras with the TensorFlow backend.

4.1 The Tennessee Eastman Process Dataset

Introduced by Downs and Vogel in 1992 [7], the Tennessee Eastman Process (TEP) represents a model of an industrial chemical process aimed at facilitating the design, exploration, and evaluation of process control technologies. The TEP dataset has been utilized in a wide range of research studies, encompassing the design of control strategies for plant-wide applications, multivariate control analysis, educational purposes, anomaly detection, and fault diagnosis. Additionally, this dataset is publicly available [23].

The training subset comprises 500 simulations, each simulation containing 500 observations, resulting in a total of 250,000 observations. In each simulation, the variables were sampled every 3 minutes and the simulations ran for 25 hours in the case of the training subset. The testing subsets, on the other hand, ran for 48 hours and consisted of 500 simulations, each simulation having 960 observations, amounting to a total of 480,000 data points. Every subset consists of 55 columns, which encompasses 52 variables, the simulation number, the sample number, and an additional column for supplementary information.

4.2 Architecture, Hyperparameters and Additional Parameters

The LSTMTF model comprises a single hidden layer containing 16 hidden units, along with an output layer consisting of 1 neuron. In this model, the inputs include A Feed, A and C Feed, Product Separator Pressure, Stripper Pressure, Stripper Temperature, Stripper Steam Flow, Reactor Cooling Water Outlet Temp, Reactor feed Analysis B, Reactor feed Analysis E, A feed flow, Reactor Cooling Water Flow and the corresponding output is Product Analysis F.

The training process entails 300 epochs, with an initial learning rate of 0.005, a batch size of 32, and a sequence length of 40. The models are trained using the Adam optimizer, with a learning rate decay of 4% every 20 epochs. Furthermore, the LSTMTF model is designed to be stateful and in a many-to-many mode. In terms of additional parameters, the prediction window w, utilized in Algorithm 1 and 2, was set to be equal to 10,240, the same length as the selected training set. For the Histogram Euclidean Distance metric the number of bins and bin edges were computed using the default *Sturges rule* [2] for each experiment. This results in a value for the ψ number of bins of 15.

5 Experimental Results

To minimize the possibility of obtaining biased results, each experiment was repeated 10 times, requiring the model to be re-trained and retested in each instance. Consequently, the results presented in this section represent the average of the 10 individual experiments for every scenario.

5.1 Overfitting Tests

The results of the y-overfit tests are illustrated in Table 1. Recall, in Algorithm 1, the computation of the y-overfitting scores included dividing the result by the threshold (e.g., the maximum of the distribution distances from the base reference scenario), thus the y-overfitting score represents the factor increase from the threshold.

In the first scenario, for each experiment, an independent exogenous input variable was disabled. In total, 11 experiments were performed, one for each of the exogenous inputs. As shown in the same table, the results using the three proposed metrics exhibited numerous similarities, with values ranging from 1.4 to 3.5. This corresponds to a percentage increase range of 40% to a maximum of 250% from the threshold.

Moving forward to the results for Scenario II, where all the exogenous inputs were disabled, leaving as input only the previous output ground truth value. The experiments reveal y-overfitting scores ranging from 4.56 to 10.24 between the three distance metrics. These numbers translate to a 356% increase (measured via the Euclidean Histogram Distance) and a 924% increase (measured using the Wasserstein Distance).

The third assumption tested in this study entailed the adoption of a constant value for the previous ground truth value. The y-overfitting scores exhibit a

range spanning from a minimum of 7.15 (measured via the Histogram Euclidean Distance) to a maximum of 13.21 (measured using the Wasserstein Distance). These numerical values signify a multiplication factor that varies between 6 and 12 above the designated threshold and corresponds to a percentage range of 600% to 1200%.

Table 1. Experimental results for three tested y-overfitting scenarios.

	Scenario I										
	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}
ED	2.4	2.3	2.35	2.01	1.44	1.83	2.79	2.79	1.5	2.06	1.73
WD	2.99	2.89	2.98	2.59	1.8	2.2	3.39	3.51	1.84	2.59	2.11
HD	2.16	2.11	2.27	1.97	1.44	1.6	2.42	2.5	1.42	1.99	1.64
	Scenario II						**Scenario III**				
ED	10.11						11.11				
WD	10.24						13.21				
HD	4.56						7.15				

5.2 Missing Values

For every simulation (e.g., 500 observations), a maximum of 5 consecutive values were removed. The placement of the missing values was selected at random for each simulation. The results, measured using the Energy Distance, reveal a factor of 0.72 from the maximum value, denoting a decrease of 28%, and a factor of 1.0014 from the average value, denoting a 0.14% increase from the average. Similar results were obtained when utilizing the Wasserstein Distance with a factor of 0.74 from the maximum value and a factor of 1.0012 from the average value, representing an increase of 0.12%. Finally, the Histogram Euclidean Distance yields a factor of 0.47 from the maximum value, corresponding to a 53% decrease from the maximum value. With respect to the average value, a factor of 1.0011 was obtained, indicating a 0.11% increase.

6 Discussions and Conclusions

In this study, we have addressed two critical challenges in the field of system modeling, namely that models that use the previous output ground truth value as an additional input might excessively rely on this additional input; and real-time handling of missing data. Our research proposed a novel method to test the former issue, named y-overfitting, specifically focusing on LSTMs trained with the TF algorithm. Additionally, our study introduced a method to deal with missing data in real time. Using this method, the model can effectively handle missing values while making predictions, enhancing the robustness and reliability of the LSTM-based system. Although the experimental evaluation results

highlighted that all the selected distance metrics clearly demonstrated that the model did not y-overfit in any of the tested scenarios, they also revealed the sensitivity of various distance metrics, with the Histogram Euclidean Distance as the least sensitive. This could easily be explained by the fact that converting the distributions to histograms could result in information loss, depending on the construction of the histogram. In this direction, the authors in [2] empirically tested the sensitivity of these models using various bin selection methodologies.

When testing the first y-overfitting assumption, namely that disabling a single exogenous input would result in a similar distribution of the prediction error, it was observed that disabling certain inputs yielded different results. These results ranged from an increase of 44% over the baseline to 140%. Further investigation has revealed that the model exhibits varying degrees of sensitivity to certain inputs. However, the fact that in all experiments there was a significant increase over the baseline threshold also indicates that all the inputs contributed to the resulting prediction. This indicates that the model successfully discovered the underlying relationships between all inputs and the output.

In terms of missing values, the experimental assessment revealed a tiny increase in the prediction error (with respect only to the average value), with a maximum increase of 0.14% and an average increase value of 0.12%, further proving that such a solution would be robust to a realistic small number of missing values related to communication faults or erroneous sensor readings.

In future work, the experimental evaluation presented in this article will be extended with other datasets and comparisons with established techniques. The valuable knowledge obtained from this paper will be utilized to further develop feature analysis, anomaly detection, and system monitoring techniques.

References

1. Alibabaei, K., Gaspar, P.D., Lima, T.M.: Modeling evapotranspiration using encoder-decoder model. In: International Conference on Decision Aid Sciences and Application. pp. 132–136. IEEE (2020)
2. Bolboacă, R.: Adaptive ensemble methods for tampering detection in automotive aftertreatment systems. IEEE Access **10**, 105497–105517 (2022)
3. Bolboacă, R., Haller, P.: Performance analysis of long short-term memory predictive neural networks on time series data. Mathematics **11**(6) (2023)
4. Che, Z., Purushotham, S., Cho, K., Sontag, D., Liu, Y.: Recurrent neural networks for multivariate time series with missing values. Scientific reports **8**(1), 6085 (2018)
5. Chen, K., Zhou, Y., Dai, F.: A lstm-based method for stock returns prediction: A case study of china stock market. In: 2015 IEEE International Conference on Big Data (Big Data). pp. 2823–2824 (2015)
6. Cui, Z., Ke, R., Pu, Z., Wang, Y.: Stacked bidirectional and unidirectional lstm recurrent neural network for forecasting network-wide traffic state with missing values. Transportation Research Part C: Emerging Technologies **118**, 102674 (2020)
7. Downs, J.J., Vogel, E.F.: A plant-wide industrial process control problem. Computers & chemical engineering **17**(3), 245–255 (1993)
8. Elmaz, F., Özgün Yücel: Data-driven identification and model predictive control of biomass gasification process for maximum energy production. Energy **195**, 117037 (2020)

9. Enders, C.K.: Applied missing data analysis. Guilford Publications (2022)
10. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)
11. Han, J., Kang, S.: Dynamic imputation for improved training of neural network with missing values. Expert Systems with Applications **194**, 116508 (2022)
12. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
13. Hyndman, R.J., Athanasopoulos, G.: Forecasting: principles and practice. OTexts (2018)
14. Khayati, M., Lerner, A., Tymchenko, Z., Cudré-Mauroux, P.: Mind the gap: an experimental evaluation of imputation of missing values techniques in time series. In: Proceedings of the VLDB Endowment. vol. 13, pp. 768–782 (2020)
15. Lem, N.: An adaptive model of pulse in jazz percussion: Rhythmic generation in quasi-periodic musical contexts using sequence-to-sequence learning
16. Lin, H., Zhang, S., Li, Q., Li, Y., Li, J., Yang, Y.: A new method for heart rate prediction based on lstm-bilstm-att. Measurement **207**, 112384 (2023)
17. Lin, W.C., Tsai, C.F., Zhong, J.R.: Deep learning for missing value imputation of continuous data and the effect of data discretization. Knowledge-Based Systems **239**, 108079 (2022)
18. Narendra, K.S., Parthasarathy, K.: Neural networks and dynamical systems. International Journal of Approximate Reasoning **6**(2), 109–131 (1992)
19. Nguyen, T., Lu, T., Sun, J., Le, Q., We, K., Schut-Aine, J.: Transient simulation for high-speed channels with recurrent neural network. In: 2018 IEEE 27th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS). pp. 303–305. IEEE (2018)
20. Nicolai, G., Silfverberg, M.: Noise isn't always negative: Countering exposure bias in sequence-to-sequence inflection models. In: Proceedings of the 28th International Conference on Computational Linguistics. pp. 2837–2846 (2020)
21. Pearson, R.K.: Nonlinear input/output modelling. Journal of Process Control **5**(4), 197–211 (1995)
22. Ramdas, A., García Trillos, N., Cuturi, M.: On wasserstein two-sample testing and related families of nonparametric tests. Entropy **19**(2), 47 (2017)
23. Rieth, C., Amsel, B., Tran, R., Cook, M.: Additional tennessee eastman process simulation data for anomaly detection evaluation. Harvard Dataverse **1** (2017)
24. Rizzo, M.L., Székely, G.J.: Energy distance. wiley interdisciplinary reviews: Computational statistics **8**(1), 27–38 (2016)
25. Schmidt, F.: Generalization in generation: A closer look at exposure bias. arXiv preprint arXiv:1910.00292 (2019)
26. Tang, Y., Wang, Y., Liu, C., Yuan, X., Wang, K., Yang, C.: Semi-supervised lstm with historical feature fusion attention for temporal sequence dynamic modeling in industrial processes. Engineering Applications of Artificial Intelligence **117** (2023)
27. Wang, Y.S., Lee, H.Y.: Learning to encode text as human-readable summaries using generative adversarial networks. arXiv preprint arXiv:1810.02851 (2018)
28. Wei, W., Li, X., Zhang, B., Li, L., Damaševičius, R., Scherer, R.: Lstm-sn: complex text classifying with lstm fusion social network. The Journal of Supercomputing pp. 1–26 (2023)
29. Williams, R.J., Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. Neural computation **1**(2), 270–280 (1989)
30. Yeung, D.S., Cloete, I., Shi, D., wY Ng, W.: Sensitivity analysis for neural networks. Springer (2010)
31. Zhou, B., Yang, G., Shi, Z., Ma, S.: Interpretable temporal attention network for covid-19 forecasting. Applied Soft Computing **120**, 108691 (2022)