A preliminary study of LSTM networks and other deep learning tools for multivariate time series anomaly detection

Zacharie Rodière
zacharie.rodiere@ensea.fr

Mohamed Ait Taknioune
mohamed.aittakniouine@ensea.fr

***Abstract*—We present our study of time series anomaly detection, through the reading and understanding of relevant literature, and our own experiments using long short-term memory networks. We first attempt to summarize the key concepts behind the Exathlon benchmark which was our first assignment, before diving into our own practical experimentations using deep learning.**

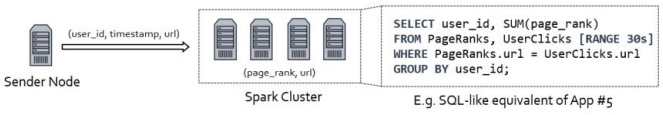
WHAT IS EXATHLON?

Exathlon [1] is a benchmark for explainable anomaly detection over high-dimensional time series data, it was developed by researchers from Intel Labs and Ecole Polytechnique.

UNDERSTANDING EXATHLON

Exathlon dataset

The **Exathlon** dataset [1] is a collection of data traces that were collected from a use case scenario implemented on **Apache Spark**. The spark workload consisted of 10 different applications that processed user clickstreams from the WorldCup 1998 website on a Spark cluster of 4 nodes. The researchers collected the data for a **2.5 month** period from both the Spark Monitoring, the Instrumentation Interface and the underlying Operating System. Researchers removed manually some of the data, which were reduced to only **93 traces**, where a trace represents the monitoring a single machine for a given duration.



Anomaly Types

Several types of anomalies were created in order to evaluate if AD systems are able to detect them: Bursty Input, Bursty Input Until Crash, Stalled Input, CPU Contention, Driver Failure and Executor Failure. These are anomaly types that could occur in the real world, and that maintenance technicians in data centers might want to look out for, so this is a useful industry use case, as would be many other uses for anomaly detection in time series (biomedical for example to create a more accurate polygraph lie detector, or uses in other engineering disciplines like detecting faults in a jet engine).

Trace Type	Anomaly Type	# of Traces	Anomaly Instances	Anomaly Length (RCI + EEI) min, avg, max	Data Items
Undisturbed	N/A	59	N/A	N/A	1.4M
Disturbed	T1: Bursty input	6	29	15m, 22m, 33m	360K
Disturbed	T2: Bursty input until crash	7	7	8m, 35m, 1.5h	31K
Disturbed	T3: Stalled input	4	16	14m, 16m, 16m	187K
Disturbed	T4: CPU contention	6	26	8m, 15m, 27m	181K
Disturbed	T5: Driver failure	11	9	1m, 1m, 1m	128K
Disturbed	T6: Executor failure		10	2m, 23m, 2.8h	
Ground Truth	(app_id, trace_id, anomaly_type, root_cause_start, root_cause_end, extended_effect_start, extended_effect_end)				

Fig. 1: Anomaly Types

Pipeline

The researchers designed a full pipeline to explain anomaly detection (AD), starting by the **Data partitioning** where data is divided into training and test data, **Data Transformation** with the following steps: resampling,dimensionality reduction and rescaling, **AD Modeling**, **AD Interface** and **AD Evaluation** where a DL method is used to create a normality model to detect anomalies and finally, the model is run over each trace to detect anomalies, and then it's ability to detect and separate normal from anomalous data is evaluated. Finally there are **ED Execution and Evaluation** where the ED module returns an explanation for each reported anomaly.

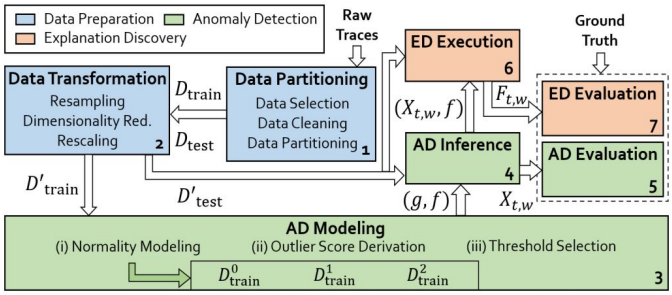


Fig. 2: A pipeline for explainable AD on multivariate time series

Deep-learning models evaluated in the Exathlon benchmark

The Exathlon benchmark is a comparison between three different deep-learning methods (autoencoder, BiGAN, and LSTM) used for detecting anomalies in data. The LSTM architecture is implemented using Tensorflow, and the benchmark

evaluates the performance of the three methods against each other.

UNDERSTANDING LSTM

Have you ever found it difficult to remember all the details of a long story ? LSTM works in a similar fasion to the way our brain processes and stores information. It does this by taking in information in small portions (words in a sentence for example) and retaining the significant details of it. This type or Recurrent Neural Network was introduced by Sepp Hochreiter in 1997 [2], and has since then found many applications in the prediction of sequential data, and was created as a solution to the vanishing and exploding gradient problem found in classical recurrent neural networks. The mathematical justification of its inner workings are brilliantly explained in [11].

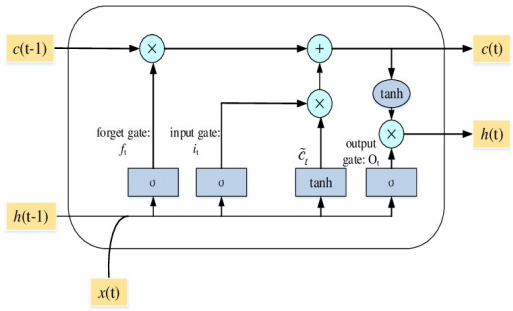


Fig. 3: LSTM cell

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

Fig. 4: Precise LSTM cell equation. PyTorch documention [8]

An **LSTM network** consists in unrolled **LSTM cells** that are capable of storing and retrieving information from the previous cells, allowing for the modeling of complex sequential dependencies (in time, space or anything else sequential). Each LSTM cell has three gates that control how it stores and retrieves information: the input gate, the forget gate, and the output gate.

The input gate controls the amount of information to keep based on what's come before. The forget gate decides which pieces of information can be discarded. Finally, the output gate determines which are most useful.

There are two different types of signals that go through the LSTM network: the **hidden state** and the **cell state**.

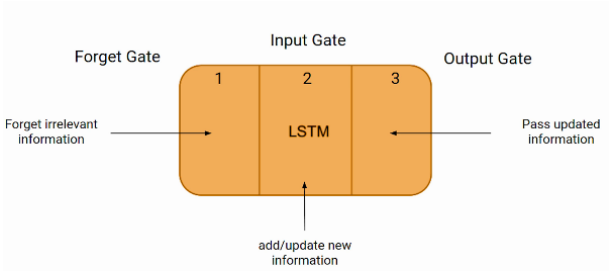


Fig. 5: Gates in LSTM

The cell state represents the long term memory of the model, it is a specificity of LSTM models, and the hidden state is used as the working memory of the model, i.e the short-term memory. So an LSTM has both long and short term memories, which explains why it performs better when there are long term dependencies in the data. The reasoning behind the design and invention of the LSTM cell was explained by Shertinsky et. al [11], and this reasoning lead them to the invention of an improved LSTM cell (but this implementation was not tested at the time of publishing).

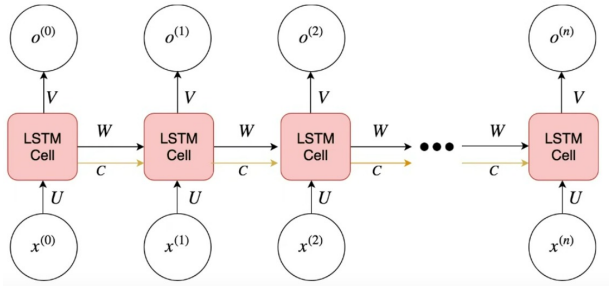


Fig. 6: Unrolled LSTM layer

IMPLEMENTING LSTM AD IN PYTORCH

Dataset

The dataset used for this exploration of anomaly detection is the SMD dataset, provided by Ya Su et. al [4] [9] in their ACM conference paper on TSAD (time series anomaly detection). It contains 38 server monitoring metrics at one minute intervals for different machines, similar to the Exathlon dataset [1]. We only worked with data from one machine. A model trained with data from different machines would probably perform poorer because the servers might be inherently different in terms of hardware, or the workloads they are exposed to. Each machine has associated with it a training set and a test set, for which we know the ground truth anomalies. There seems to be debate on whether those ground truth labels are correct or not, or if the anomaly detection is actually trivial, but we went for this dataset anyway, as a means to learn *how to practically implement* the methods, knowing that our results might not be as good as other state of the art methods.

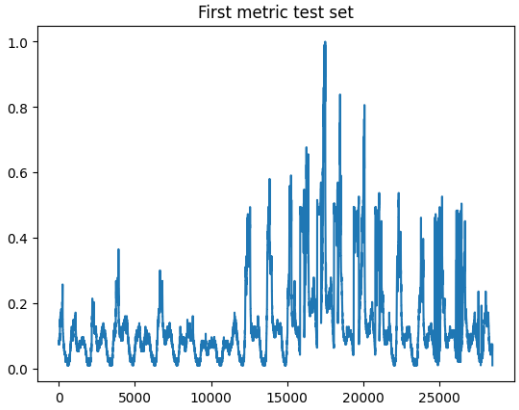


Fig. 7: Evolution of the first metric in the test set of `machine-1-1` in the SMD dataset

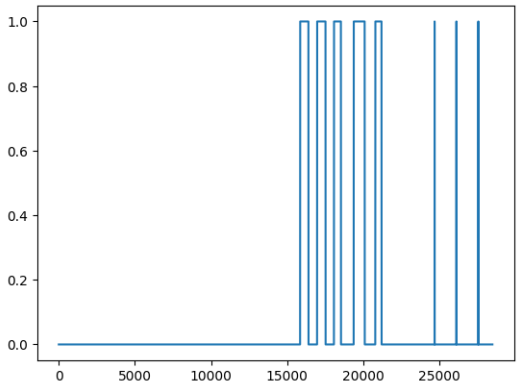


Fig. 8: Ground truth anomaly label for `machine-1-1` in the SMD dataset. 1 represents anomalous timestamps, and 0 normal timestamps

Predictive Models

We tried two models. A first simple model contains a single LSTM layer, unrolled for `sequence_length=20` time steps, with `hidden_dim=256` latent size. The output layer is a simple fully connected layer. We train this first model by predicting 1 time step in advance. In the more complex model, we used 2 stacked LSTMs, where the output of the first LSTM layer is fully connected to the input of the second LSTM. This method was first proposed by Malhotra et. al [3].

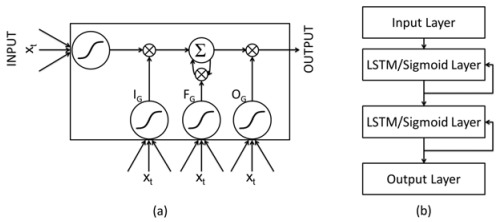


Fig. 9: Overview of the suggested model [3]

```
LSTM_Predictor(
  (lstm): LSTM(38, 256)
  (linear): Linear(in_features=256, out_features=38, bias=True)
)
Size of a batch input: (20, 16, 38)
Size of a batch target: (1, 16, 38)

Stacked_LSTMs(
  (double_lstm): LSTM(38, 256, num_layers=2)
  (linear): Linear(in_features=256, out_features=190, bias=True)
)
Size of a batch input: (20, 16, 38)
Size of a batch target: (5, 16, 38)
```

Fig. 10: First and second model architectures seen in PyTorch [3]

Forecasting based anomaly detection and outlier score derivation

The principle of forecasting based anomaly detection is the following: we predict the future behavior of a multivariate time series from the past: a given number of preceding time steps are used to compute each following step. We then compare the predictions to reality, and thus obtain error vectors. This procedure is better explained in [3].

In the case where we predict multiple steps in advance, there are multiple predictions for the same time step, in which case the error vector is the concatenation of the error vectors for each prediction. We then compute the covariance matrix and mean vector of those error vectors, which amounts to fitting a multivariate normal distribution using MLE (maximum likelihood estimation). We then compute the likelihood of each error vector (we have one error vector per time step) according to the fitted normal distribution, using its PDF. To that end, we used the `multivariate_normal` function from *Scipy* [10]. Those likelihoods can be very high (such as 10^{75}), so we take the log-likelihood, then multiply it by -1.The result is a human-readable outlier score (the higher, the more anomalous), but is is still not normalized between 0 and 1, we thus max-normalize it, to obtain outlier scores between 0 and 1. If we denote $p^{(t)}$ the likelihood of a given error vector, the corresponding outlier score $s^{(t)}$ is thus:

$$s^{(t)} = \frac{-\log(p^{(t)})}{\max(-\log(\mathbf{p}))}$$

Model training

We worked on *Google Colab* and impemented the two networks using *PyTorch*. The two models managed to train relatively well with some trial and error for hyper-parameter tuning. They are trained only on the data of the first machine `machine-1-1` (all the 38 metrics are used).

As can be observed, the test loss contains large peaks, which correspond to anomalies in the data (this is after all, an anomaly detection task, so it makes complete sense). It means that our predictive model might indeed allow us to predict anomalies.

Model inference

Inference was done with both models over the test set, and we visually compared the evolution (see Fig.10) of the ground

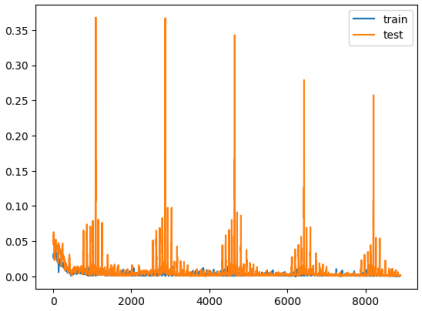


Fig. 11: Simple LSTM predictive model train and test losses

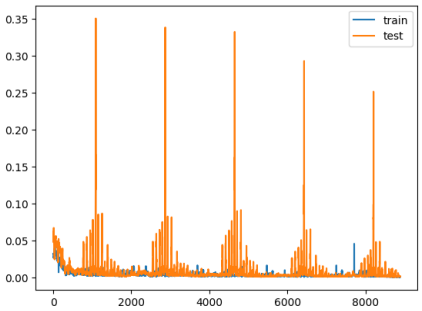


Fig. 12: Stacked LSTMs predictive model train and test losses

truth metric and the model prediction. For the stacked LSTM model, since we have 5 different predictions for each time step, we averaged them into one, to compare with the ground truth. We visually found that there was a clear improvement in the prediction performance by using the *stacked LSTMs* model.

Outlier score derivation

In both cases we computed the error vectors on the test set as explained earlier, and obtained their covariance matrix and mean, which we can display as images. In the case of the *simple LSTM*, we then displayed the detected anomalies for an arbitrary threshold $\tau_{ar} = 0.7$ and for the optimal threshold choice $\tau_{opt} = 0.18$, found by maximization of the $F_{0.1}$ score over a range of possible thresholds, as suggested in [3].

For the *stacked LSTMs* model, we did not manage to compute the outlier scores, since the probability distribution is too sparse in dimension 190, so each error vector is extremely unlikely, such that all the likelihoods were found equal to zero. The sparsity of the distribution makes outlier score derivation impossible, since we rely on the likelihoods. Reducing the dimensionality of the base problem (ignoring some dimensions, for instance those that are constant for their whole duration) could help alleviate that problem.

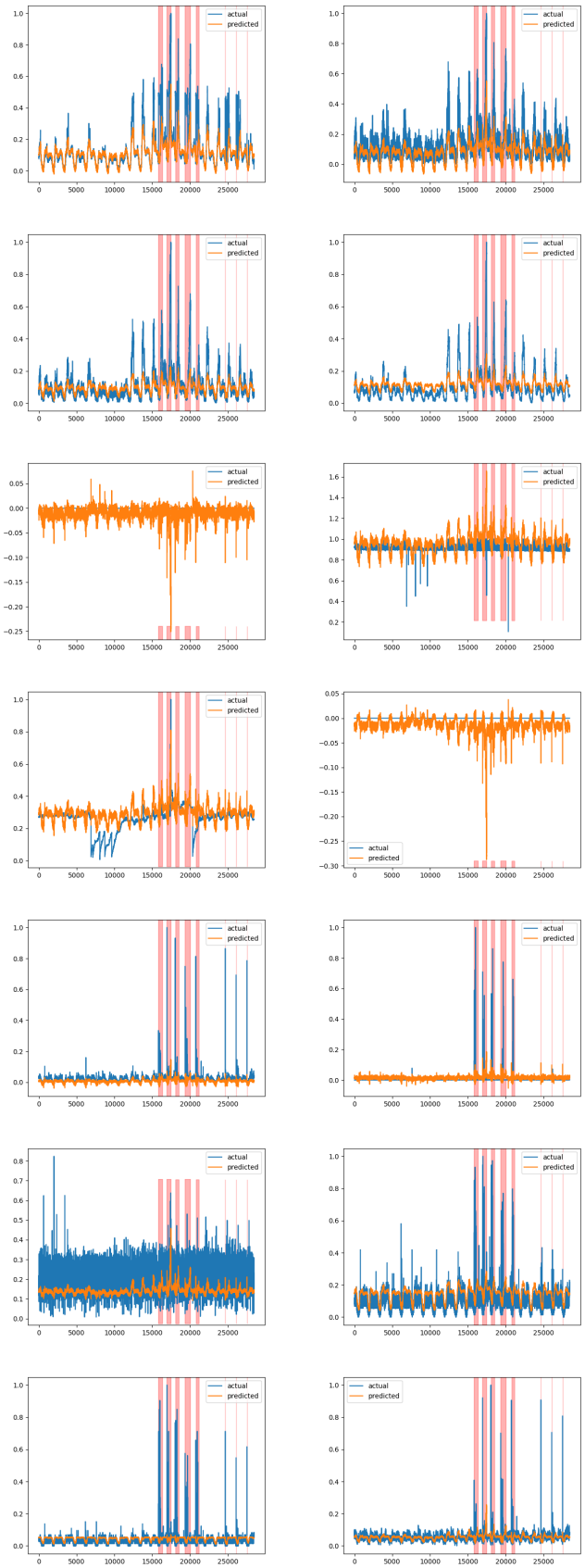


Fig. 13: Inference on 14 out of 38 metrics using the *simple LSTM* model

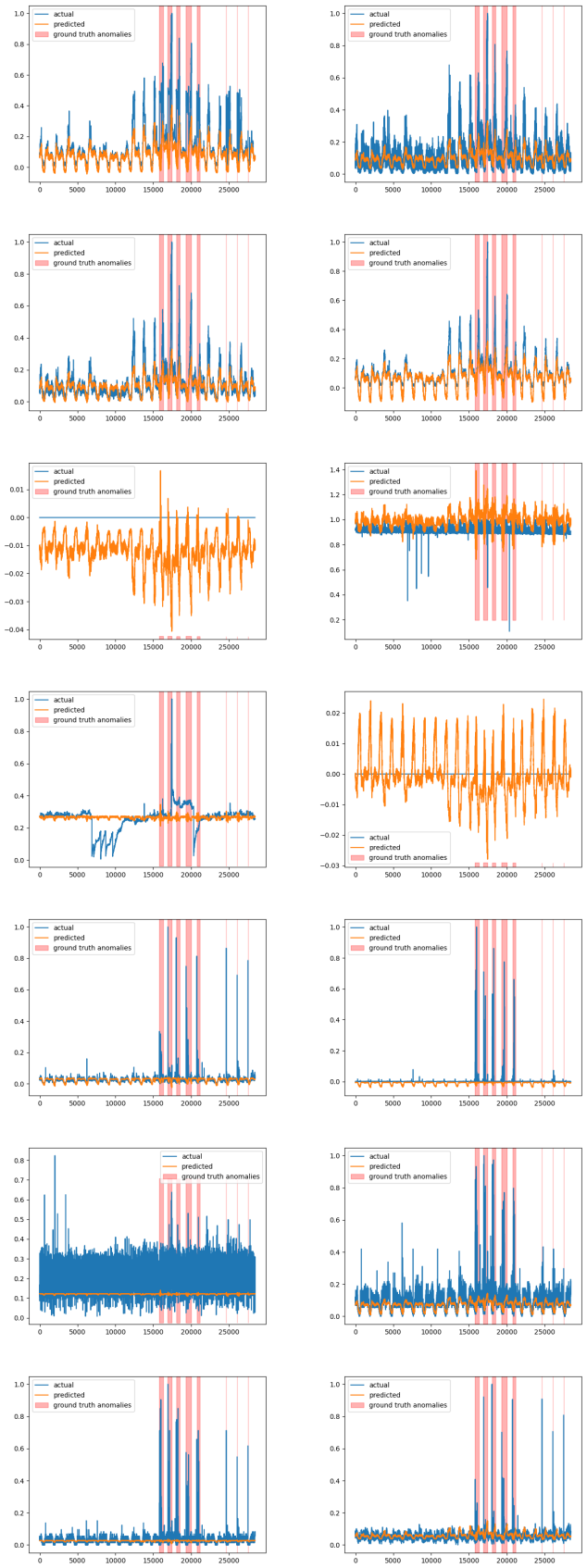


Fig. 14: Inference on 14 out of 38 metrics using the *stacked LSTMs* model. We can observe better quality in the predictions.

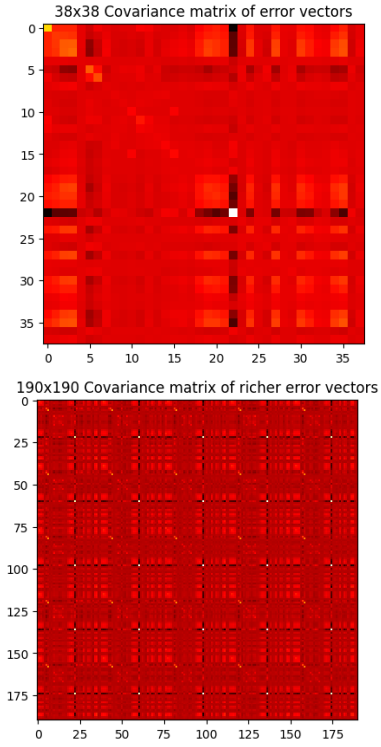


Fig. 15: Error vector covariance matrices for the *simple LSTM* and *stacked LSTMS* models

$$F_\beta = (1+\beta^2).\frac{\text{precision.recall}}{(\beta^2.\text{precision}) + \text{recall}}$$

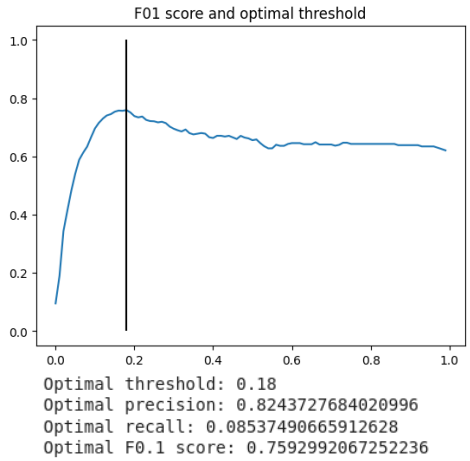


Fig. 16: Finding the optimal threshold by maximizing the $F_{0.1}$ score

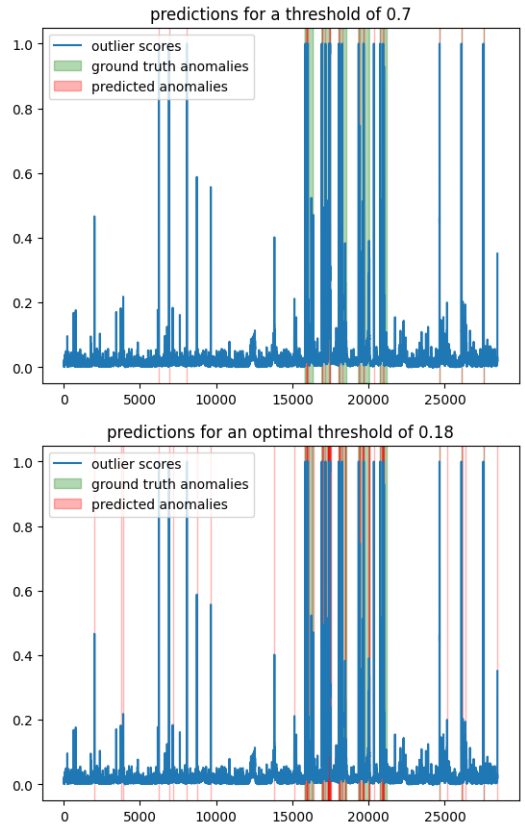


Fig. 17: Comparison of the prediction quality for an arbitrary $\tau_{ar} = 0.7$ threshold and for the optimal threshold $\tau_{opt} = 0.18$ for the *simple LSTM* model

REFERENCES

[1] Vincent Jacob,Fei Song,Arnaud Stiegler, Bijan Rad,Yanlei Diao,Nesime Tatbul."Exathlon: A Benchmark for Explainable Anomaly Detection over Time Series".

[2] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long Short-Term Memory." Neural Computation, vol. 9, no. 8, 1997, pp. 1735–1780, https://doi.org/10.1162/neco.1997.9.8.1735.

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

[4] Ya Su, et al. "Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network." Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, https://doi.org/10.1145/3292500.3330672.

[5] Alex Graves, Abdel-rahman Mohamed, Geoffrey Hinton.Speech Recognition with Deep Recurrent Neural Networks. https://doi.org/10.48550/arXiv.1303.5778

[6] StatQuest with Josh Starmer.Recurrent Neural Networks.https://www.youtube.com/watch?v=AsNTP8Kwu80

[7] StatQuest with Josh Starmer. Long Short-Term Memory (LSTM). https://www.youtube.com/watch?v=YCzL96nL7j0

[8] PyTorch documentation on the LSTM layer. https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

[9] OmniAnomaly dataset. GitHub repository. https://github.com/NetManAIOps/OmniAnomaly

[10] Multivariate normal random variable generation using Scipy.https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.multivariate_normal.html

[11] Sherstinsky, Alex. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network." Physica D: Nonlinear Phenomena, vol. 404, 2020, p. 132306, https://doi.org/10.1016/j.physd.2019.132306.

APPENDIX: NOTEBOOK REPORT TO REPRODUCE OUR RESULTS

See the adjoined file `TSAD_LSTM.pdf`, which is a *pandoc* generated report of the *Jupyter* notebook created in *Google Colab*.