Time Series Encodings with Temporal Convolutional Networks

Markus Thill$^{1[0000-0002-6429-180X]}$, Wolfgang Konen$^{1[0000-0002-1343-4209]}$, and Thomas Bäck$^{2[0000-0001-6768-1478]}$

[1] TH Köln – University of Applied Sciences, Germany
{markus.thill,wolfgang.konen}@th-koeln.de
[2] LIACS, Leiden University, The Netherlands
t.h.w.baeck@liacs.leidenuniv.nl

Abstract. The training of anomaly detection models usually requires labeled data. We present in this paper a novel approach for anomaly detection in time series which trains unsupervised using a convolutional approach coupled to an autoencoder framework. After training, only a small amount of labeled data is needed to adjust the anomaly threshold. We show that our new approach outperforms several other state-of-the-art anomaly detection algorithms on a Mackey-Glass (MG) anomaly benchmark. At the same time our autoencoder is capable of learning interesting representations in latent space. Our new MG anomaly benchmark allows to create an unlimited amount of anomaly benchmark data with steerable difficulty. In this benchmark, the anomalies are well-defined, yet difficult to spot for the human eye.

Keywords: Time Series Representations · Temporal Convolutional Networks · Autoencoder · Anomaly Detection · Unsupervised Learning · Mackey-Glass Time Series · Chaos

1 Introduction

For the operation of large machines in companies or other critical systems in society, it is usually necessary to record and monitor specific machine or system health indicators over time. In the past, the recorded time series were often evaluated manually or by simple heuristics (such as threshold values) to detect abnormal behavior. With the more recent advances in the fields of ML (machine learning) and AI (artificial intelligence), ML-based anomaly detection algorithms are becoming increasingly popular for many tasks such as health monitoring and predictive maintenance. Supervised algorithms need labeled training data, which are often cumbersome to get and to maintain in real-world applications. Yet, unsupervised anomaly detection remains up to now a challenging task.

In this paper we propose a novel autoencoder architecture for sequences (time series) which is based on temporal convolutional networks [3] and shows its efficacy in unsupervised learning tasks. Our experiments show that the architecture can learn interesting representations of sequences in latent space. The idea of unsupervised anomaly learning is based on the assumption that in real-world tasks the overwhelming part of the time-series data will be normal. Without the need to label the data, we train a model that learns the normal behavior, i. e. assigns a low score to normal and a higher score to anomalous data. Finally, only a small fraction of labeled data is needed to find a suitable threshold for the anomaly score. This can also be fine-tuned in operation, with an already trained model.

For the initial benchmarking and comparison of our algorithm, we introduce a new synthetic benchmark based on Mackey-Glass (MG) time series [21]. In its current form, the Mackey-Glass Anomaly Benchmark (MGAB) consists of 10 MG time series in which anomalies were inserted using a clearly defined procedure. Although the anomalies are inserted synthetically, spotting them is rather difficult for the human eye. Due to the structured insertion process and the clear labeling of nominal and anomalous data, no domain knowledge is required to correctly label the data. Additionally, the difficulty of the anomaly detection task is steerable by simply adjusting a few parameters of the MGAB generation process (e. g. time delay, smoothness parameters).

2 Related Work

Other well known time series anomaly benchmarks are Yahoo Webscope S5 [16] and NAB [17]. The Webscope S5 benchmark mostly contains simple/trivial spatial anomalies. In NAB [17], the labeling process is not always immediately comprehensible without having domain-dependent knowledge of the time series. Furthermore, the amount of data is often too small for many deep learning approaches. In [29], we introduce an anomaly benchmark based on electrocardiogram recordings of the MIT-BIH ECG dataset [10].

In recent years a lot of effort was put into the design of time series anomaly detection algorithms and many new methods have been proposed: A common approach is to use the prediction error of a time series regression model as anomaly score [23,29,30]. Commonly, also autoencoder-based approaches are used [11,22], where the reconstruction error of the time series serves as an indicator for anomalous behaviour. Other approaches are based on generative adversarial networks (GANs) [13,18] or variational-based networks/autoencoders [27,32,26]. There exists also an architecture [33] where the parameters of a deep autoencoder and of a Gaussian mixture model are simultaneously learned during training. Most of the aforementioned algorithms are unsupervised.

In this work we will compare several state-of-the-art algorithms on MGAB: The first one is DNN-AE, an anomaly detection algorithm based on a regular deep neural network autoencoder [11]. DNN-AE takes short sequences from a time series and attempts to encode and reconstruct these. Large reconstruction errors indicate anomalous behavior. Similar to DNN-AE, the algorithm LSTM-ED [22] uses an encoder-decoder approach, but now based on LSTM networks [12] to encode short sub-sequences taken from a time series. A third algorithm, Numenta's anomaly detection algorithm NuPIC [28] is based on the hierarchical temporal memory (HTM) algorithm [9] which is biologically inspired by the neocortex of the brain. Finally, the LSTM-AD algorithm [29] uses stacked LSTM networks to predict a time series for several prediction horizons and learns a statistical model of normal behavior in order to detect anomalous events. All algorithms compared in this work are unsupervised, since no anomaly labels are passed to the algorithms during training. Only during the test phase a small fraction of the labels are used to determine a suitable anomaly threshold.

3 TCN Autoencoder

In computer vision architectures, convolutional neural networks (CNN) are very popular due to their equivariance properties and sparse interactions. Temporal convolutional networks (TCN) translate these convolutional advantages from computer vision into the time domain, as we will detail in Section 3.1 and Section 3.2.

The central idea of the TCN autoencoder (TCN-AE) is to encode a sequence of length T into a significantly shorter sequence of length T/s (where $s \in \mathbb{Z}^+$ is a sampling factor) and subsequently to reconstruct the original sequence from the compressed sequence (using a decoder network). The idea is similar to a classical (deep) autoencoder, where fixed-sized inputs are encoded into a latent space representation and the latent variables are used to reconstruct the original input. Similarly, the TCN-AE encodes sequences along the temporal axis into a compressed representation and then attempts to reconstruct the original sequence. However, it differs from a regular autoencoder in so far that it replaces the dense layer architecture of a regular autoencoder with the more powerful convolutional architecture. Due to this, it is also more flexible with respect to the input length. Our TCN autoencoder consists of two temporal convolutional neural networks (TCNs) [3], one for encoding and one for decoding. Additionally, a downsampling and upsampling layer are used in the encoder and decoder, respectively. The individual components will be described in more detail in the following.

3.1 Discrete Dilated Convolutions

The dilated acausal convolution of a d-dimensional sequence $\mathbf{x} : \{0, 1, \ldots, T-1\} \to \mathbb{R}^d$, and a discrete filter with a finite impulse response (FIR filter) $\mathbf{h}[n]$, $\mathbf{h} : \{0, 1, \ldots, k-1\} \to \mathbb{R}^d$, can be defined as:

$$y[n] = (\mathbf{x} *_q \mathbf{h})[n] = \sum_{i=0}^{k-1} \mathbf{h}[i]^\intercal \cdot \mathbf{x}[n - q \cdot (i - k/2)], \tag{1}$$

where $y[n] \in \mathbb{R}$ is the output of the filter with size $T-k+1$, $q \in \mathbb{N}$ is the dilation rate, $\mathbf{h}[i] \in \mathbb{R}^d$ is the impulse response of the filter with kernel size k. While the regular convolution ($q = 1$) applies the filter to adjacent elements of the input

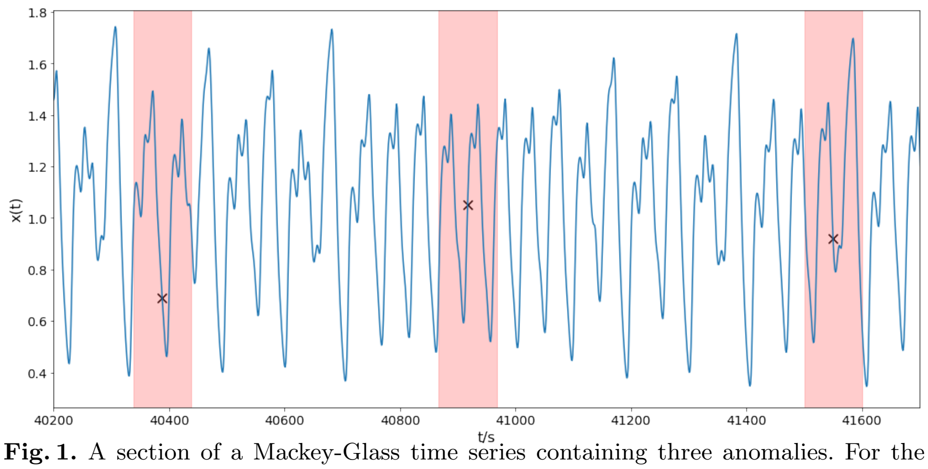


Fig. 1. A section of a Mackey-Glass time series containing three anomalies. For the human eye these anomalies would be very hard to spot, if we took the red bars away.

sequence, the dilated convolution $*_q$ allows to skip several values in the input sequence before the next filter tap $\mathbf{h}[i]$ is applied. The convolution operation slides a $k \times d$-dimensional filter stepwise over the input sequence $\mathbf{x}[n]$ and computes a weighted average of $\mathbf{x}[n]$ and the corresponding weights $\mathbf{h}[i]$ in each step. Since the filter is only sled along the (discrete) time-axis, the operation is commonly referred to as one-dimensional convolution. The convolution in Eq. (1) is slightly acausal due to the term $k/2$. In some applications it might also be reasonable to use causal convolutions.

Many neural network architectures for sequence modeling utilize dilated convolutions in order to build a hierarchical temporal model with a large receptive field. These models are capable of learning long-term temporal patterns in the input data. The main idea is to construct a stack of dilated convolutional layers, where the dilation rate increases with every additional layer. A common choice is to start with a dilation rate of $q = 1$ for the first layer of the network and to double q with every new layer. This approach allows to increase the receptive field of the model exponentially.

3.2 Temporal Convolutional Networks

The temporal convolutional network (TCN) [3] is inspired by several convolutional architectures [6,8,14,24], but differs from these approaches, according to the authors, insofar as it combines simplicity, auto-regressive prediction, residual blocks and very long memory. A full description of TCN would be out of scope for this paper, the reader is referred to [3] for the details. Its main elements are however the dilated convolutions of Section 3.1. A TCN can be basically described by three elements: a list of dilation rates $(q_1, q_2, \ldots, q_{n_r})$, the number of filters n_{filters}, and the kernel size k, which is the same for all filters in a TCN.

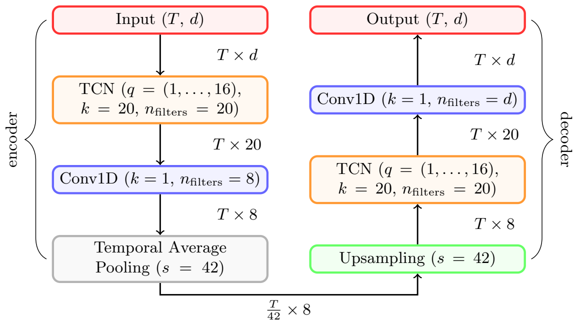

Input (T, d)
$T \times d$
TCN ($q = (1, \ldots, 16)$, $k = 20$, $n_{\text{filters}} = 20$)
$T \times 20$
Conv1D ($k = 1$, $n_{\text{filters}} = 8$)
$T \times 8$
Temporal Average Pooling ($s = 42$)
$\frac{T}{42} \times 8$
Upsampling ($s = 42$)
$T \times 8$
TCN ($q = (1, \ldots, 16)$, $k = 20$, $n_{\text{filters}} = 20$)
$T \times 20$
Conv1D ($k = 1$, $n_{\text{filters}} = d$)
$T \times d$
Output (T, d)
encoder
decoder

Fig. 2. Architecture of TCN-AE. Each layer is described by its parameters inside the box. The input of the TCN-AE is a sequence $\mathbf{x}[n]$ with length T and dimensionality d.

3.3 An Autoencoder Using TCNs

The novel element we propose in this paper is an autoencoder (AE) for time series *which employs TCNs* as building blocks. This architecture, which we name TCN-AE, is sketched in Fig. 2. Like any autoencoder, TCN-AE consists of an encoder and a decoder. The encoder initially processes the input sequence $\mathbf{x}[n]$ of length T and dimension d using a TCN. Subsequently, in order to reduce the size of the feature map (dimensionality) of the TCN's output, a one-dimensional convolutional layer (1×1 convolution [19]) is used with $q = 1$, $k = 1$ and a smaller number of filters (i.e., $n_{\text{filters}} = 8$). The temporal average pooling layer is the last layer in the encoder and responsible for downsampling the series by a factor s. It does so by averaging groups of size s along the time axis.

Right afterwards, the downsampled sequence is passed to the decoder module and brought back to its original length using an upsampling layer which simply performs a nearest neighbor interpolation. The upsampled sequence is passed through a second TCN, which is parameterized in the same way as the encoder-TCN, but has independent weights. Finally, the reconstruction of the input sequence is generated with a Conv1D layer which ensures (by setting $k = 1$ and $n_{\text{filters}} = d$) that the dimensionality of the input is matched. Once TCN-AE is trained, the input sequence and its reconstruction will be used for detecting anomalies, as described in the next section.

3.4 Anomaly Detection with TCN-AE

A natural application of TCN-AE is the anomaly detection in time series. When TCN-AE is trained on time series containing predominantly nominal data, the network will attempt to minimize the reconstruction error for these nominal patterns. At the same time, the reconstruction error for anomalous patterns or patterns which differ significantly in their characteristics should be larger. One possibility to identify these unusual patterns is to estimate a distribution for the reconstruction error. In our approach, we decide to slide a window of length ℓ over our reconstruction error and compute a mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. Subsequently, the Mahalanobis distance can be used as anomaly score. The unified algorithmic description of the anomaly detection procedure in combination with TCN-AE is listed in Algorithm 1. Only for determining the anomaly threshold, 10% of the true labels are used, as described in Section 5.3.

Algorithm 1 Anomaly detection algorithm using the TCN-AE architecture.

```
1: Adjustable parameters:
2:     $\mathcal{M}_\tau$: anomaly threshold, obtained as described in Section 5.3
3:     $\ell$: window length for constructing the error vectors
4:     $T_{\text{train}}$: length of training sub-sequences
5:
6: function ANOMALYDETECT($\mathbf{x}_{tr}[n], \mathbf{x}[n]$)    ▷ time series $\mathbf{x}_{tr}, \mathbf{x} : \mathbb{N} \to \mathbb{R}^d$ of length $T$
7:     Construct model TCNAE() and Initialize the trainable parameters
8:     for $\{1 \ldots n_{\text{epochs}}\}$ do
9:         Extract training sub-sequences $\mathbf{X}^{(i)}_{\text{train}} \in \mathbb{R}^{T_{\text{train}} \times d}$ from $\mathbf{x}_{tr}[n]$, $i = 1, \ldots, B$
10:        $\forall i \in \{1, \ldots, B\}$ : TRAIN(TCNAE, $\mathbf{X}^{(i)}_{\text{train}}$)    ▷ Train net on mini-batches
11:    end for
12:    $\hat{\mathbf{x}}[n] \leftarrow$ TCNAE($\mathbf{x}[n]$)    ▷ Encode and reconstruct whole sequence
13:    $\mathbf{e}[n] \leftarrow \mathbf{x}[n] - \hat{\mathbf{x}}[n]$    ▷ reconstruction error $\mathbf{e} : \mathbb{N} \to \mathbb{R}^d$ of length $T$
14:    $\mathbf{E}[n] \leftarrow$ SLIDINGWINDOW($\mathbf{e}[n]$, $\ell$)    ▷ $\mathbf{E} : \mathbb{N} \to \mathbb{R}^{T \times \ell \times d}$
15:    $\mathbf{E}'[n] \leftarrow$ RESHAPE($\mathbf{E}[n]$)    ▷ $\mathbf{E}' : \mathbb{N} \to \mathbb{R}^{T \times \ell \cdot d}$
16:    $\boldsymbol{\mu}, \boldsymbol{\Sigma} =$ ESTIMATE($\mathbf{E}'[n]$)    ▷ $\boldsymbol{\mu} \in \mathbb{R}^{\ell \cdot d}$,
17:    $M[n] \leftarrow (\mathbf{E}'[n] - \boldsymbol{\mu})^\intercal \boldsymbol{\Sigma}^{-1} (\mathbf{E}'[n] - \boldsymbol{\mu})$    ▷ Mahalanobis distance for each point
18:    $a[n] \leftarrow \begin{cases} 0 & \text{if } M[n] < \mathcal{M}_\tau \\ 1 & \text{else} \end{cases}$    ▷ Binary anomaly flags
19:    return $a[n]$    ▷ Return anomaly flag for each time series point
20: end function
```

4 The Mackey-Glass Anomaly Benchmark

In this work we will compare various anomaly detection algorithms on a non-trivial synthetic benchmark, named Mackey-Glass anomaly benchmark (MGAB) in the following. Mackey-Glass time series are known to exhibit chaotic behavior under certain conditions. MGAB contains 10 MG time series of length $T = 10^5$. Into each time series 10 anomalies are inserted with a procedure described in Section 4.1. In contrast to other synthetic benchmarks, the introduced anomalies are for the human eye very hard to distinguish from the normal (chaotic) behavior. Overall, we generate 100 anomalies in 10^6 time series points. The benchmark data and the detailed procedure for generating these and similar benchmark data are publicly available at GitHub [31].[3]

4.1 Generating Anomalies in Mackey-Glass Time Series

In order to create the Mackey-Glass Anomaly Benchmark, we first generate a sufficiently long time series having a dimension of $d = 1$ using the JiTCDDE [2] solver with the parameters $\tau = 18, n = 10, \beta = 0.25, \gamma = 0.1, h = 0.9$. The integration step size is set to 1. The maximal Lyapunov exponent (MLE) of $\lambda_{mle} = 0.0061 \pm 0.0002$ suggests that the generated time series is (mildly) chaotic. Subsequently, we split this series into ten same-sized individual time series and insert 10 anomalies into each time series.

[3] GitHub repository: `https://github.com/MarkusThill/MGAB/`

5 Results

5.1 Experimental Setup

Anomaly Detection Algorithms All training algorithms are unsupervised, i. e. they do not need the true anomaly labels during the training process. Only in order to find a suitable anomaly threshold, a small fraction of labels is used, as described in Section 5.3. Otherwise, the anomaly labels are only used at test time to evaluate the performance of the individual algorithms. In one run, each algorithm is trained for 10 rounds: in the i-th round the algorithms are trained on the i-th time series and evaluated on the time series $\{1, \ldots, 10\} \setminus \{i\}$. In total, we perform 10 runs with different random seeds. In order to find suitable hyper-parameters for each algorithm, we use the HYPEROPT library [4] and optimize the F_1-score on a separate MG time series.

For all neural networks we use the Adam optimizer [15] to train the weights by minimizing the MSE loss. Additionally, all time series (having a dimension of $d = 1$) are standardized to zero mean and unit variance.

DNN-AE [7]: we use a PyTorch [25] implementation for the anomaly detection algorithm based on a deep autoencoder [11]. The algorithm requires several parameters, which we choose as follows: batch size $B = 100$, number of training epochs $n_{\text{epochs}} = 40$, sequence length $T_{\text{train}} = 150$ and a hidden size of $h = 10$ for the bottle neck (which results in a compression factor of $T_{\text{train}}/h = 15$ for each sequence). Finally, we set $\%_{Gaussian} = 1\%$, which specifies that 99% of the data is used to estimate a Gaussian distribution for the anomaly detection task.

LSTM-ED [22] is also implemented using PyTorch and uses the following parameter setting: batch size $B = 100$, number of training epochs $n_{\text{epochs}} = 20$, sequence length $T_{\text{train}} = 300$, hidden size $h = 100$ and $\%_{Gaussian} = 1\%$. Both, encoder and decoder use a stacked LSTM network with two layers.

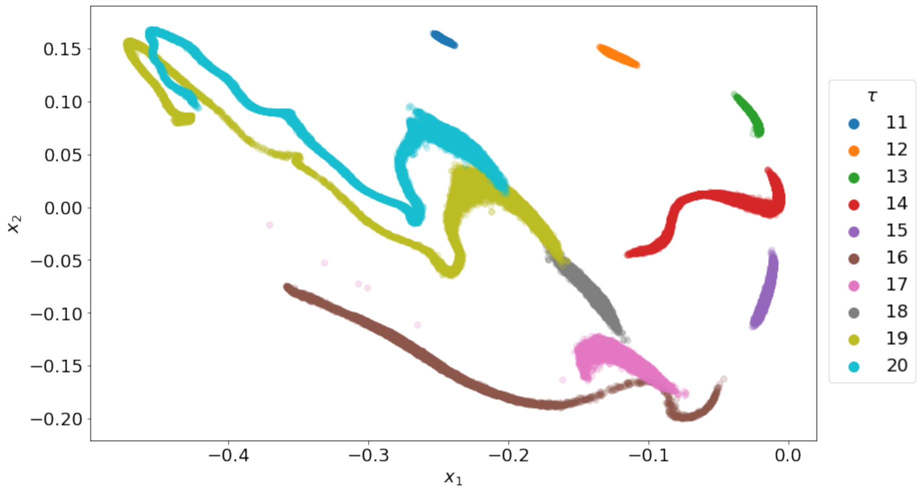
NuPIC [28]: Numenta's anomaly detection algorithm has a large range of hyper-parameters which have to be set. We use the parameters recommended by the authors in [17]. It is possible to tune the parameters with an internal swarming tool [1]. However, this is a time-expensive process which is not feasible for the large MGAB dataset.

LSTM-AD [29]: here we select the following parameters: batch size $B = 1024$, number of training epochs $n_{\text{epochs}} = 30$, and sequence length $T_{\text{train}} = 128$. A 2-layer LSTM network with 256 units in the first layer and 128 units in the second layer is used. The target horizons are chosen to be $H = (1, 3, \ldots, 51)$.

TCN-AE: The main TCN-AE parameters are given in Fig. 2. Additionally we use the sequence length $T_{\text{train}} = 1050$, batch size $B = 32$ and $n_{\text{epochs}} = 40$.

5.2 Learning Time Series Representations

In our first experiment we want to assess the capabilities of the TCN-AE architecture to learn representations of time series. For this purpose we train a TCN-AE model using many different MG time series with a varying time delay parameter τ. Ideally, TCN-AE should be able to learn the main characteristics



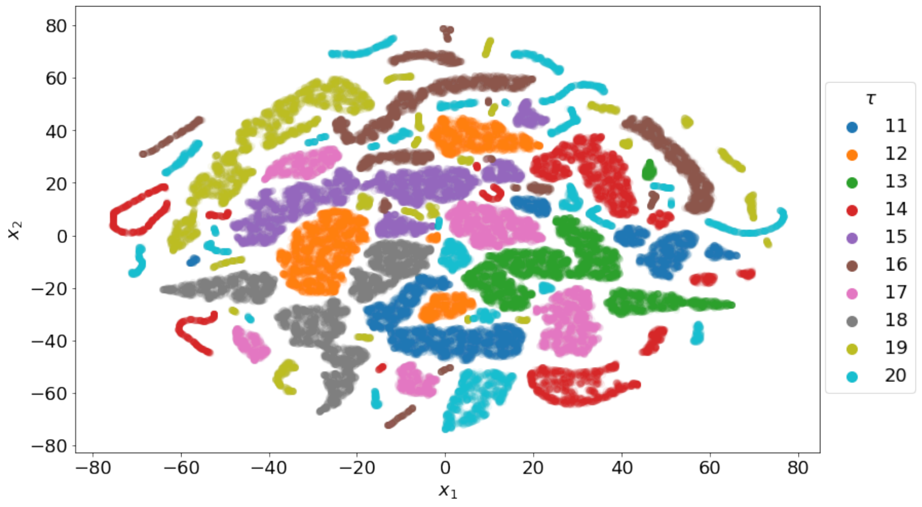


Fig. 3. Top: 2d-representation of 10^5 (10^4 for each τ) different Mackey-Glass time series using TCN-AE. The (unsupervised) algorithm is capable of learning an encoding which separates the MG time series fairly well according to their τ value.
Bottom: 2d-representation of the same MG time series, but now using t-SNE [20] to find suitable encodings.

of the individual time series and find suitable compressed representations. In our experiment we use TCN-AE on 10^5 different Mackey-Glass time series (10^4 for each τ in the range of $\tau = 11 \ldots 20$). Each time series of length 256 is encoded into a 2-dimensional compressed representation. The algorithm is trained in an unsupervised manner, hence, τ is not passed to the algorithm at any time. Surprisingly, even with this large compression rate of 128, TCN-AE can find an interesting embedding for the MG time series, as depicted in Fig. 3 (top). For a certain τ, all samples are placed in *only one* connected cluster (with the exception of a few satellites) and these clusters are mostly – with a few small exceptions – *non-overlapping.*

For comparison, we repeated the same experiment with the popular t-SNE [20] clustering algorithm. We executed t-SNE on a GPU with the help of a certain CUDA implementation [5]. We tried different parameter settings and finally fixed the perplexity parameter to 200, the learning rate to 10 and the number of iterations to 10^4. The results for t-SNE in Fig. 3 (bottom) indicate that it is not a trivial task to find suitable representations for MG time series. t-SNE has in comparison to TCN-AE more difficulties to cluster all sequences with a certain time delay parameter τ in only *one* connected region.

5.3 Algorithm Evaluation

Determining the Anomaly Threshold All algorithms output an anomaly score for each point of the time series. A low anomaly score indicates nominal behavior and high scores suggest that anomalies are present. In order to classify each point as nominal or anomalous a so-called anomaly threshold is required. Points with a score above the threshold are classified as anomalous, all other points are classified as nominal. We determine this threshold for all algorithms as follows: A sub-sequence containing 10% of the data is taken and the anomaly threshold is optimized on this short sequence, such that the F_1-score is maximized. The optimal threshold is then fixed for the complete time series and the overall results are obtained. Since the results can vary depending on which sub-sequence is used for the threshold adjustment, we repeat the above procedure, similarly to k-fold cross validation, for 10 different 10% sub-sequences of the considered time series and record the results for the 10 different sub-sequences.

Performance Measures In order to assess the performance of all algorithms and to be able to compare the results, we use several common performance metrics in this paper. Analogously to typical classification problems, a confusion matrix can be constructed for time series anomaly detection tasks, containing the number of true-positives (TP), false-positives (FP), false-negatives and true-negatives (TN). TP indicates the number of anomalies, which were correctly identified within an anomaly window (a small range around the actual anomaly point). Only the first detection in an anomaly window is counted. On the other hand, a missed anomaly window (no point inside the window is flagged) will be judged as a FN. If a point is incorrectly presumed to be anomalous (detection outside any anomaly window), this will be considered as a FP. All other points, which are not marked as anomalous, are considered as true-negatives (TN). From these four quantities the well known performance measures precision, recall and F_1-score can be derived.

5.4 Anomaly Detection on the Mackey-Glass Anomaly Benchmark

In a second experiment, we compare TCN-AE to several state-of-the-art anomaly detection algorithms on the Mackey-Glass Anomaly Benchmark. For each algorithm, except NuPIC, 10 runs were performed. Hence, for each algorithm and time series 10 different models are trained and each model is evaluated on the other nine time series. NuPIC is completely deterministic and does not require several runs. Additionally, as described in Section 5.1, the anomaly threshold for each algorithm and time series is tuned on 10 different sub-sequences. We add up the TP, FN and FP over all 10 time series and summarize the results in Tab. 1. Up to 100 anomalies can be detected in total. We can see that the (deep) DNN-AE detects most of the anomalies (approx. 92), missing only about 8 on average. However, this result is achieved at the expense of producing many false-positives. Overall, DNN-AE produces more than 60 false positives on average, while TCN-AE produces less than one. Hence, DNN-AE achieves the highest recall among all algorithms but ranks only 3$^{\text{rd}}$ in F_1-score, due to its low precision. TCN-AE scores best in F_1-score and precision. NuPIC has the poorest performance in all measures.

Table 1. Results for MGAB. The results shown here (mean and standard deviation of 10 runs and 10 sub-sequences, Section 5.3) are for the sum of TP, FN and FP over all 10 time series. For each algorithm and time series the anomaly threshold was tuned on 10% of the data using a cross-validation approach: the threshold is tuned on 10 different 10%-sequences of the data.

Algorithm	TP	FN	FP	Precision	Recall	F_1-score
NuPIC [28]	3.00 ± 0.00	97.00 ± 0.00	132.00 ± 0.00	0.02 ± 0.00	0.03 ± 0.00	0.03 ± 0.00
LSTM-ED [22]	14.60 ± 5.86	85.40 ± 5.86	57.00 ± 20.43	0.21 ± 0.08	0.15 ± 0.06	0.17 ± 0.06
DNN-AE [11]	91.79 ± 1.22	8.21 ± 1.22	62.58 ± 13.65	0.60 ± 0.06	0.92 ± 0.01	0.72 ± 0.04
LSTM-AD [29]	88.80 ± 2.59	11.20 ± 2.59	0.62 ± 0.61	0.99 ± 0.01	0.89 ± 0.03	0.94 ± 0.01
TCN-AE [this work]	90.54 ± 1.72	9.46 ± 1.72	0.20 ± 0.47	1.00 ± 0.01	0.91 ± 0.02	0.95 ± 0.01

5.5 Discussion

The initial results that we obtained with our new TCN-AE architecture are promising. The learned representations (Fig. 3) on different MG time series appear to be useful and may reveal interesting insights. For anomaly detection we achieve with TCN-AE and LSTM-AD the highest F_1-score on the non-trivial MG benchmark. Remarkably, all algorithms except NuPIC require many trainable weights. TCN-AE had 164 451 parameters, DNN-AE 241 526, LSTM-ED

244 101 and LSTM-AD 464 537. That is, the other high-performing algorithms require 50%–300% more trainable weights than TCN-AE.

6 Conclusion & Future Work

In this work, we proposed with TCN-AE a novel autoencoder architecture for multivariate time series and evaluated it on various Mackey-Glass (MG) time series with respect to two relevant tasks: representation learning and anomaly detection. TCN-AE could learn a very interesting representation in only two dimensions which accurately distinguishes MG time series differing in their time delay values τ (Section 5.2). On the Mackey-Glass Anomaly Benchmark (MGAB), which was introduced in this paper, TCN-AE achieved better anomaly detection results than other state-of-the-art anomaly detectors (Section 5.4).

Possibilities for future work on TCN-AE include: (a) Gaining more insights from the representations that TCN-AE learns unsupervisedly (Fig. 3). (b) Since the network architecture allows to train TCN-AE with training sequences of arbitrary length, another improvement could be to start the training process with short sequences and then successively increase the sequence length after each epoch. This approach could enable a faster learning progress in the beginning and allow fine tuning of the weights towards the end of the training. (c) We are planning to evaluate TCN-AE on other real-world anomaly detection benchmarks containing (multi-variate) time series. Possible options are electrocardiogram signals [10] or industrial monitoring tasks [16,17].

References

1. Ahmad, S.: Running swarms. `http://nupic.docs.numenta.org/0.6.0/guide-swarming.html` (Retreived: 2020-06-29) (2017)
2. Ansmann, G.: Efficiently and easily integrating differential equations with JiTCODE, JiTCDDE, and JiTCSDE. Chaos **28**(4), 043116 (2018)
3. Bai, S., Kolter, J.Z., Koltun, V.: An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. CoRR **abs/1803.01271** (2018)
4. Bergstra, J., et al.: Hyperopt: A Python library for model selection and hyperparameter optimization. Comput. Sci. Discov **8**(1), 014008 (2015)
5. Chan, D.M., Rao, R., Huang, F., Canny, J.F.: GPU accelerated t-distributed stochastic neighbor embedding. JPDC **131**, 1–13 (2019)
6. Dauphin, Y.N., Fan, A., Auli, M., Grangier, D.: Language Modeling with Gated Convolutional Networks. In: ICML'17. p. 933–941 (2017)
7. Fischer, M., et al.: Anomaly Detection on Time Series: An Evaluation of Deep Learning Methods. `https://github.com/KDD-OpenSource/DeepADoTS` (2019)
8. Gehring, J., Auli, M., Grangier, D., Yarats, D., Dauphin, Y.N.: Convolutional Sequence to Sequence Learning. CoRR **abs/1705.03122** (2017)
9. George, D., Hawkins, J.: Towards a mathematical theory of cortical micro-circuits. PLoS Comput Biol **5**(10), e1000532 (2009)
10. Goldberger, A.L., et al.: PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation **101**(23), e215–e220 (2000)

11. Hawkins, S., He, H., Williams, G., Baxter, R.: Outlier detection using replicator neural networks. In: DaWaK. pp. 170–180. Springer (2002)
12. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural computation **9**(8), 1735–1780 (1997)
13. Jiang, W., Hong, Y., Zhou, B., He, X.: A GAN-Based Anomaly Detection Approach for Imbalanced Industrial Time Series. IEEE Access **7**, 143608–143619 (2019)
14. Kalchbrenner, N., Grefenstette, E., Blunsom, P.: A Convolutional Neural Network for Modelling Sentences. In: ACL. pp. 655–665. Baltimore, Maryland (2014)
15. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980 (2014)
16. Laptev, N., Amizadeh, S.: Yahoo anomaly detection dataset S5. `http://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70` (2015)
17. Lavin, A., S. Ahmad, S.: Evaluating Real-time Anomaly Detection Algorithms – the Numenta Anomaly Benchmark. In: ICMLA (2015)
18. Li, D., et al.: MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks. In: ICANN. pp. 703–716. Springer (2019)
19. Lin, M., Chen, Q., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2013)
20. van der Maaten, L., Hinton, G.: Visualizing Data using t-SNE . Journal of Machine Learning Research **9**, 2579–2605 (2008)
21. Mackey, M.C., Glass, L.: Oscillation and chaos in physiological control systems. Science **197**(4300), 287–289 (1977)
22. Malhotra, P., et al.: LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection. CoRR **abs/1607.00148** (2016)
23. Munir, M., et al.: DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series. IEEE Access **7**, 1991–2005 (2019)
24. van den Oord, A., et al.: WaveNet: A Generative Model for Raw Audio. CoRR **abs/1609.03499** (2016)
25. Paszke, A., et al.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H., et al. (eds.) NIPS, pp. 8024–8035. Curran Assoc. (2019)
26. Pereira, J., Silveira, M.: Unsupervised Anomaly Detection in Energy Time Series Data Using Variational Recurrent Autoencoders with Attention. In: Wani, M.A., et al. (eds.) ICMLA. pp. 1275–1282. IEEE (2018)
27. Sölch, M., et al.: Variational Inference for On-line Anomaly Detection in High-Dimensional Time Series. CoRR **abs/1602.07109** (2016)
28. Taylor, M., et al.: numenta/nupic: 1.0.5. `https://doi.org/10.5281/zenodo.1257382` (2018)
29. Thill, M., Däubener, S., Konen, W., Bäck, T.: Anomaly Detection in Electrocardiogram Readings with Stacked LSTM Networks. In: ITAT. CEUR Workshop Proceedings, vol. 2473, pp. 17–25 (2019)
30. Thill, M., Konen, W., Bäck, T.: Online anomaly detection on the Webscope S5 Dataset: A comparative study. In: EAIS. pp. 1–8. IEEE (2017)
31. Thill, M., Konen, W., Bäck, T.: MGAB: The Mackey-Glass Anomaly Benchmark (2020). https://doi.org/10.5281/zenodo.3762385
32. Xu, H., et al.: Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications. In: WWW. pp. 187–196 (2018)
33. Zong, B., et al.: Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection. In: ICLR (2018)