MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks (Machine Learning 2021 Course)

Valerii Baianov [1] **Ilya Dubovitskii** [1] **Dmitrii Leshchev** [1] **Gleb Mezentsev** [1]

Abstract

Revealing anomalies from time series data is a crucial task in many fields (e.g. bank operations anomalies, IoT sensors anomalies). The main idea of this paper is to review state-of-the-art methods of this task, reproduce one of the modern approaches for sequential data anomaly detection - MAD-GAN(Li et al., 2019) and expand this idea to detect anomalies in MNIST data sequence (e.g. "11711" - 7 is anomaly). As a result, we reimplemented base paper (Li et al., 2019) code with PyTorch, compared obtained results with paper ones, improved architecture to work with sequences of images and analyzed obtained results for this task expension.

Github repo: https://github.com/Glebzok/MAD-GAN
Video presentation: Drive, YouTube

1. Introduction

Anomaly detection for sequential data is a popular task that reflects in a various fields as e.g. Internet of Things (IoT), banking, manufacturing. Such problems as fraud bank operations, cyber-attacks detection or mechanism wear identification require a particular approach that will take into account the difficult structure of data and special metrics. One of the popular trends to detect anomaly is using Generative Adversarial Networks (GANs). While the GAN-based approach is well-investigated for anomaly detection in such data as images, there are a few unresolved issues for sequential data, e.g. considering temporal data structure (Li et al., 2019).

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Valerii Baianov <valerii.baianov@skoltech.ru>, Ilya Dubovitskiy <Ilya.Dubovitskii@skoltech.ru>, Dmitrii Leshchev <Dmitrii.Leshchev@skoltech.ru>, Gleb Mezentsev <Gleb.Mezentsev@skoltech.ru>.

Final Projects of the Machine Learning 2020 Course, Skoltech, Moscow, Russian Federation, 2020.

In this project, we try to reproduce the approach from one of the resent work (Li et al., 2019) that adapt common GANs frameworks for anomaly detection for numerical time series. Moreover, we will explore the possibility of expanding this approach for more complex sequential data, e.g. image sequences.

The main contributions of this report are as follows:

1. Reviewing SOTA methods for anomaly detection for sequential data
2. Reproducing MAD-GAN from the paper (including reimplementing it with PyTorch)
3. Evaluating MAD-GAN on different datasets and comparing results with the paper
4. Modifying the architecture to deal with image sequences of MNIST images

The rest of this paper is organized as follows. Section 2 presents an overview of the state-of-the-art methods for anomaly detection for sequential data. Section 3 introduces MAD-GAN framework and anomaly score function from the paper and also present our architecture for processing image sequences. In Section 4, we introduce the tested datasets for both tasks. In Section 5, we show the experimental results of our paper re implementation and image sequence processing model. Finally, Section 6 summarizes both objectives and suggests possible future work. Note: All the experiments were carried out using Google Colab

2. Related work

For better understanding of the task we firstly analyzed state of the art (SOTA) methods for anomaly detection for sequential data. Anomaly detection methods can be formally divided into three main categories:

1. Statistical approaches
2. Machine learning approaches

3. Deep learning approaches

In (Braei & Wagner, 2020) authors studied and evaluated 20 anomaly detection methods from all three categories on univariate time-series data. Their evaluation of ROC-AUC metric on 5 datasets (NYCT, Real Yahoo Services Network traffic and three its synthetic variations) showed, that statistical methods [AutoRegressive model (AR), Moving Average model (MA)] showed both better ROC-AUC and computational time results, than machine learning and deep learning approaches on Yahoo dataset and its synthetic variations. As for NYCT dataset, top 3 ROC-AUC scores were K-Mean (machine learning method), LSTM and Wavenet (both deep learning methods) with 0.914, 0.84 and 0.823 scores respectively, while best statistical approach AR gave only 0.637 score. Authors call the presence of contextual anomalies in NYCT as the reason for such bad result of statistical approaches and point out, that statistical methods overfits to such data, while deep learning approaches are more flexible due to the broad range of hyperparameters to achieve high ROC-AUC values. **So we can see, that for the simplest univariate time-series case statistical methods perform better and faster, while deep learning methods show themselves more flexible to different types of anomalies.**

Then we continued our research of SOTA methods for multivariate timeseries. One of the most popular nowadays approaches is using GANs. In our base paper (Li et al., 2019) authors used GAN architecture, where generator and discriminator are two Long-Short-Term Recurrent Neural Networks (LSTM-RNN), to take into account sequential format of data. Generator tries to create plausible anomaly data, while discriminator trains to distinguish anomaly data from normal one. After enough pre-training procedure, authors detect anomalies based on both reconstruction (GAN Generator) and discrimination (GAN Discriminator) losses. In (Geiger et al., 2020) authors propose TadGAN approach, using cycle-consistent GAN architecture. The idea is simmilar to MAD-GAN, but it is trained with cycle consistency loss (Zhu et al., 2017) for effective time-series data reconstruction. They also compare it with ARIMA, LSTM, MAD-GAN and other methods on 11 datasets (with multivariate time-series data) from 3 reputable entities (NASA, Yahoo, and Numenta). As a metric authors used F1-Score. Top-3 models by the mean F1-Score values on all datasets are TadGAN, LSTM and Arima, with 0.7 ± 0.123, 0.623 ± 0.163 and 0.599 ± 0.148 F1-Scores respectively. Authors also calculated that LSTM methods outperform traditional Arima in 5 cases and TadGAN in 8. The advantage of Arima is mostly seen on synthetic data, however deep learning approaches produces compatetive results in this scenario.

Another approach is Decomposition + Convolutional Neural Network, which was proposed in (Gao et al., 2020). Authors firstly decompose time series as the sum of trend, seasonality, and remainder components, using their algorithm from previous works (Qingsong Wen, 2020). After that authors adopt an encoder-decoder network architecture with skip connections (U-Net structure). Authors propose, that " Many adjustments are needed to achieve the optimal results" for U-Net, so they describe them in the paper (e.g. Weight Adjusted Loss).

So the overall trend of anomaly detection in recent years is deep learning approaches: GANs (Schlegl et al., 2019), (Yoon et al., 2019), LSTM AutoEncoders (Nguyen et al., 2021), etc. Summarizing recent years trend, authors of the paper (Sovilj et al., 2020) compared different deep learning anomaly detection architectures on Yahoo, Synthetic, CICIDS2017 and Rank Software Inc. datasets. For synthetic data Encoder-Decoder model gave slightly better results, for other datasets different variations of LSTM (bidirectional (Bi), stacked (S), Bi+S) showed better Precision/Recall Score. **Summarizing the result of SOTA methods review, main trend for current researches is using deep learning models, mainly - GANs. However, for different types of data different models could be useful.**

3. Models description

3.1. MAD-GAN

3.1.1. Architecture

To handle the time-series data, authors constructed the GAN's generator and discriminator as two Long-Short-Term Recurrent Neural Networks (LSTM-RNN). The generator (G) generates fake time series with sequences from a random latent space as its inputs, and passes the generated sequence samples to the discriminator (D), which will try to distinguish the generated (i.e. "fake") data sequences from the actual (i.e. "real") normal training data sequences. As an input MAD-GAN uses not each stream independently, but the entire variable set concurrently in order to capture the latent interactions amongst the variables into the models.

As in standard GAN framework, parameters of D and G are updated based on the outputs of D, where D is trained to assign correct labels for both real and fake sequences, while generator is trained to be as smart as possible to fool the discriminator (i.e. to mislead D to assign real labels to fake sequences) after sufficient rounds of iterations. By being able to generate realistic samples, the generator G will have captured the hidden multivariate distribution. At the same time, the resulting discriminator D has also been trained to be able to distinguish fake (i.e. abnormal) data from real (i.e. normal) data with high sensitivity. In the paper, authors propose to exploit both G and D for the anomaly detection task by (i) reconstruction: exploiting the residuals between real-time testing samples and reconstructed samples by G based on the mapping from real-timespace to the GAN latent space; and (ii) discrimination: using the discriminator D to classify the time series. So they use so called Discrimination and Reconstruction Anomaly Score (DR-Score) to combine the two losses to detect potential anomalies in the data.

3.1.2. GAN-based Anomaly Detection

The anomaly detection problem using GAN is formulated as following. Given a training dataset $X \subseteq R^{M \times T}$ with T streams and M measurements for each stream, and a testing dataset $X^{test} \subseteq R^{N \times T}$ with T streams and N measurements for each stream, the task is to assign binary (0 for normal and 1 for anomalous) labels to the measurements of testing dataset. All the points in the training dataset are normal.

To effectively learn from X, authors propose to apply a sliding window with window size s_w and step size s_s to divide the multivariate time series into a set of multivariate sub-sequences $X = x_i, i = 1, 2, ..., m \subseteq R^{s_w \times T}$, where $m = \frac{(M - s_w)}{s_s}$ is the number of sub-sequences. Similarly, $Z = z_i, i = 1, 2, ..., m$ is a set of multivariate sub-sequences taken from a random space. By feeding X and Z to the GAN model, authors train the generator and discriminator with the following two-player minimax game:

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[log D(x)] \\ + E_{x \sim p_z(x)}[log(1 - D(G(x)))] \quad (1)$$

After sufficient rounds of training iterations, the trained discriminator D_{rnn} and the generator G_{rnn} can then be employed to detect anomalies in X test using a combined Discrimination and Reconstruction Anomaly Score (DR-Score), which will be introduced in Section 3.1.3.

For detection, the testing dataset $\chi^{test} \subseteq R^{N \times T}$ is similarly divided into multivariate sub-sequences $X^{tes} = x_j^{tes}, j = 1, 2, ..., n$ with a sliding window, where $n = \frac{(N - s_w)}{s_s}$. Using the computed DR-Score (DRS) of the testing dataset, authors label each of the sub-sequences in the testing dataset as follows:

$$A_t^{tes} = \begin{cases} 1, & if\, H(DRS_t, 1) > \tau \\ 0, & else \end{cases} \quad (2)$$

where $A_t^{tes} \subseteq R^{N \times 1}$ is a label vector for the testing dataset, where non-zero values indicate an anomaly is detected, i.e. the cross entropy error $H(.,.)$ for the anomaly score is higher than a predefined value τ.

3.1.3. DR-Score: Anomaly Detection using both Discrimination and Reconstruction

An advantage of using GAN is that we will have a discriminator and a generator trained simultaneously. Authors propose to exploit both the discriminator and generator that has been jointly trained to represent the normal anatomical variability for identifying anomalies. Following the formulation in (Schlegl et al., 2017), the GAN-based anomaly detection consists of the following two parts:

1. **Discrimination-based Anomaly Detection**
   Given that the trained discriminator D can distinguish fake data (i.e. anomalies) from real data with high sensitivity, it serves as a direct tool for anomaly detection.

2. **Reconstruction-based Anomaly Detection**
   The trained generator G, which is capable of generating realistic samples, is actually a mapping from the latent space to real data space: $G(Z) : Z \rightarrow X$, and can be viewed as an inexplicit system model that reflects the normal data's distribution. Due to the smooth transitions of latent space mentioned in (Radford et al., 2016), the generator outputs similar samples if the inputs in the latent space are close. Thus, if it is possible to find the corresponding Z^k in the latent space for the testing data X^{tes}, the similarity between X^{tes} and $G(Z^k)$ (which is the reconstructed testing samples) could explain to which extent is X^{tes} follows the distribution reflected by G. In other words, we can also use the residuals between X^{tes} and $G(Z^k)$ for identifying anomalies in testing data.

To find the optimal Z^k that corresponds to the testing samples, we first sample a random set Z^1 from the latent space and obtain reconstructed raw samples $G(Z^1)$ by feeding it to the generator (as shown in the right part of Fig. 1). Then, we update the samples from the latent space with the gradients obtained from the error function defined with X^{tes} and $G(Z)$.

$$\min_{Z^k} Er(X^{tes}, G_{rnn}(Z^k)) = 1 - Simi(X^{tes}, G_{rnn}(Z^k)) \quad (3)$$

where the similarity between sequences was defined as covariance for simplicity.

After enough iteration rounds such that the error is small enough, the samples Z^k is recorded as the corresponding mapping in the latent space for the testing samples. The residual at time t for testing samples is calculated as:

$$Res(X_t^{tes}) = \sum_{i=1}^{n} |x_t^{tes,i} - G_{rnn}(Z_t^{k,i})| \quad (4)$$

where $X_t^{tes} \subseteq R^n$ is the measurements at time step t for n variables. In other words, the the anomaly detection loss is

$$L_t^{tes} = \lambda Res(X_t^{tes}) + (1 - \lambda) D_{rnn}(X_t^{tes}) \quad (5)$$

In our experiments we used this detection loss final DS-Score. The proposed MAD-GAN is summarized in Algo. 1.

Algorithm 1 LSTM-RNN-GAN-based Anomaly Detection Strategy

```
loop
  if epoch within number of training iterations then
    for the $k^{th}$ epoch do
      Generate samples from the random space:
      $Z = \{z_i, i = 1, \dots m\} \Rightarrow G_{rnn}(Z)$
      Conduct discrimination:
      $X = \{x_i, i = 1, \dots, m\} \Rightarrow D_{rnn}(X)$
      $G_{rnn}(Z) \Rightarrow D_{rnn}(G_{rnn}(Z))$
      Update discriminator parameters by minimizing(descending) $D_{loss}$:
      $\min \frac{1}{m}\sum_{i=1}^{m}[-log D_{rnn}(x_i) - log(1 - D_{rnn}(G_{rnn}(z_i)))]$
      Update generator parameters by minimizing(descending) $G_{loss}$:
      $\min \sum_{i=1}^{m} log(-D_{rnn}(G_{rnn}(z_i)))$
      Record parameters of the discriminator and generator in the current iteration.
    end for
  end if
  for the $i-th$ iteration do
    Mapping testing data back to latent space:
    $Z^k = \min_Z Er(X^{tes}, G_{rnn}(Z^i))$
  end for
  Calculate the residuals:
  $Res = |X^{tes} - G_{rnn}(Z^k)|$
  Calculate the discrimination results:
  $Dis = D_{rnn}(X^{tes})$
  Obtain the combined anomaly score:
  $DRS = \lambda Res + (1-\lambda)Dis$
end loop
```

We, similar to the paper authors, used mini-batch stochastic optimization based on Adam Optimizer and Gradient Descent Optimizer for updating the model parameters in this work.

3.2. GAN for image sequences

3.2.1. Architecture

To expand the application of MAD-GAN architecture for working with image sequences, we add CNN modules to initial MAD-GAN architecture. CNN for both Generator and Discriminator have similar structure (up to inverted operations and layer order). It consists of 4 convolutional layers, each of which is followed by batch normalization and activation function (ReLU \ LeakyRelu) layers. So the final structure for CNN+LSTM Generator is LSTM, followed by CNN with MNIST shape image as an output; and for CNN+LSTM Discriminator: CNN for image features extraction, followed by LSTM module for time-series data processing ending with linear layer with sigmoid activation function for predicting the probability of a sample to be abnormal.

3.2.2. Anomaly detection for images sequence data

As we expanded architecture in such a way, that its GAN component is working in exactly the same way, as for MAD-GAN anomaly detection model, the procedure of detecting anomalies is same too. As a training dataset we used sequences of same MNIST numbers with different sequence length (1, 3, 10, 30). Sequence length here is used as an analogy of window size from MAD-GAN section. Minimax game and DR-Score are the same here.

For generating test dataset we implemented two functions, for generating normal and anomaly sequences. Function for anomaly ones takes as an input normal number, anomaly number, sequence length, proportion of anomalies in sequence, number of sequences to generate, generating model, and initial dataset.

4. Datasets

There are three datasets that were used for experiments, such as SWaT (SWa), WADI (WAD) and KDD99CUP (KDD) datasets. Each dataset includes normal (or train) and abnormal (or test) data. General description of the datasets is given by Table 1.

Table 1. General Information about Datasets

Item	SWaT	WADI	KDD
Variables	51	123	34
Attacks	36	15	2
Attacks duration	$2 \sim 25$	$1.5 \sim 30$	-
Training size	496900	1209551	562387
Testing size	449919	172801	494021
$N_{rate}(\%)$	88.02	94.01	19.69

$N_{rate}(\%)$ is the rate of normal data points versus all in testing data sets.

4.1. MAD-GAN

4.1.1. SWaT

The Secure Water Treatment (SWaT) system is an operational test-bed for water treatment that represents a small-scale version of a large modern water treatment plant found in large cities. The overall testbed design was coordinated with Singapore's Public Utility Board, the nation-wide water utility company, to ensure that the overall physical process and control system closely resemble real systems in the field. The SWaT dataset collection process lasted for a 11 days with the system operated 24 hours per day. A total of 36 attacks were launched during the last 4 days of 2016 SWaT data collection process. Generally, the attacked points include sensors (e.g., water level sensors, flow-rate meter, etc.) and actuators (e.g., valve, pump, etc.). These attacks were launched on the test-bed with different intents and diverse lasting durations (from a few minutes to an hour) in the final four days. The system was either allowed to reach its normal operating state before another attack was launched or the attacks were launched consecutively. Please refer to the SWaT website for more details about the SWaT dataset.

The water purification process in SWaT is composed of six sub-processes referred to as P1 through P6. The first process is for raw water supply and storage, and P2 is for pre-treatment where the water quality is assessed. Undesired materials are them removed by ultra-filtration (UF) backwash in P3. The remaining chorine is destroyed in the Dechlorination process (P4). Subsequently, the water from P4 is pumped into the Reverse Osmosis (RO) system (P5) to reduce inorganic impurities. Finally, P6 stores the water ready for distribution.

4.1.2. WADI

Unlike a water treatment system plant which is typically contained in a secured location, a distribution system comprises numerous pipelines spanning across a large area. This highly increases the risk of physical attacks on a distribution network. The Water Distribution (WADI) testbed is an extension of the SWaT system, by taking in a portion of SWaTs reverse osmosis permeate and raw water to form a complete and realistic water treatment, storage and distribution network. There are three control processes in the water distribution system. The first process is to intake the raw water from SWaT, Public Utility Board (PUB) inlet or the return water in WADI, and store the raw water in two tanks. P2 distributes water from two elevated reservoir tanks and six consumer tanks based on a pre-set demand pattern. Water is recycled and sent back to P1 at the third process.

The WADI testbed is similarly equipped with chemical dosing systems, booster pumps and valves, instrumentation and analysers. In addition to simulating attacks and defences being carried out on the PLCs via networks, WADI has the capabilities to simulate the effects of physical attacks such as water leakage and malicious chemical injections. The WADI data collection process consists of 16 days' continuous operations, of which 14 days data was collected under normal operation and 2 days' with attack scenarios. During the data collection, all network traffic, sensor and actuator data were collected. Please refer to the WADI website for more details about the WADI dataset.

4.1.3. KDDCUP99

This is the data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining. The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between "bad" connections, called intrusions or attacks, and "good" normal connections. This database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment.

4.2. GAN for image sequences

For generating both normal and abnormal MNIST images sequences we used Convolutional Variational AutoEncoder (CVAE) and functions, for generating normal and abnormal sequences (functions)

5. Experiments

5.1. MAD-GAN

In this section we will describe achieved results on three datasets and compare it with paper ones. All the datasets are preprocessed by standard scaler and PCA (Principal Component Analysis). Number of components is selected based on explained variance ratio.

5.1.1. Anomaly Detection Performance

In first part we reproduced experiments from the paper to compare obtained metrics for the all datasets that are described above.

The pipeline of the experiment is as follows. Firstly, we divided each sequence in train and test by sliding window with length 30, as proposed in the original paper. Then, all training and testing is conducted using these subsequences. The MAD-GAN model is trained on the normal data during 100 epochs. Secondly, we define the range of possible values of the hyperparameter λ from the equation 5. For each λ we detect anomalies in abnormal data and evaluate the model performance based on classification metrics, such as precision, recall and F1-score. For each of metric above we determine the optimal classification threshold τ. To sum up the second point, we get the the optimal classification threshold for each value of λ and each way of optimization (precision, recall, F1-score). Finally, we run the grid search among λ values and select the optimal one in terms of precision, recall and F1-score.

The Table 2 compares the paper authors MAD-GAN and our implementation of MAD-GAN. The metrics are not the same. On the one hand, we got the better performance on the WADI dataset. On the other hand, paper authors had a rather consistent metrics on the KDD dataset. However, there is no big difference between implementations and their results look slightly similar.

Table 2. Comparison of the Paper and Our Implementation. Anomaly Detection Metrics for Different Datasets

Data	Metric	MAD-GAN	Pre	Rec	F1
SWaT	Pre	Paper	1	0.55	0.7
		Our	0.99	0.001	0.002
	Rec	Paper	0.12	1	0.22
		Our	0.47	1	0.64
	F1	Paper	0.99	0.64	0.77
		Our	0.49	0.98	0.65
WADI	Pre	Paper	0.47	0.25	0.32
		Our	0.99	0.03	0.06
	Rec	Paper	0.06	1	0.12
		Our	0.11	1	0.2
	F1	Paper	0.42	0.34	0.37
		Our	0.97	0.62	0.76
KDD	Pre	Paper	0.95	0.19	0.32
		Our	0.99	0.29	0.45
	Rec	Paper	0.82	0.96	0.88
		Our	0.47	1	0.64
	F1	Paper	0.87	0.95	0.9
		Our	0.98	0.61	0.76

5.1.2. Dimension reduction and metrics on test data

For further experiments we took λ and τ as for the best F-1 Score, because this metric is the most stable. Then we compared Anomaly Detection Metrics of our implementation of MAD-GAN at Different PC Resolutions with paper values for the SWaT dataset. Results of comparison can be seen in Tables 3 and 4

Table 3. Anomaly Detection Metrics of MAD-GAN at Different PC Resolutions. Part 1

EM	Model	PC=1	PC=2	PC=3	PC=4	PC=5
Pre	Paper	16.90	17.76	18.57	14.53	26.56
	Our	0	47.40	47.46	43.50	49.30
Rec	Paper	72.03	95.34	92.76	91.16	95.27
	Our	0	100	88.56	80.27	99.27
F_1	Paper	0.24	0.23	0.23	0.23	0.37
	Our	0	0.64	0.61	0.56	0.65

As it can be seen from tables, we achieved better precision and F-1 Score results, while Recall ones are mostly lower (sometimes significantly). It might be connected with the fact, that authors used another λ and τ parameters. As they noted importance of high Recall in such tasks, we believe, they used parameters, which maximize Recall in some way. However, we found it controversial, because it is always possible to make Recall = 1 (using constant model), so we used best F-1 parameters. We also tested our MAD-GAN realization on test data of these three datasets: WADI, SWaT and KDDCUP99. In appendix, in figures 5, 6 and 7, we have plotted changes in metric values depending on the iteration.

Table 4. Anomaly Detection Metrics of MAD-GAN at Different PC Resolutions. Part 2

EM	Model	PC=6	PC=7	PC=8	PC=9	PC=10
Pre	Paper	24.39	13.37	14.78	13.60	13.83
	Our	46.04	37.40	31.17	47.33	48.80
Rec	Paper	81.25	91.87	92.02	95.06	96.03
	Our	86.79	64.07	48.98	91.54	92.13
F_1	Paper	0.22	0.22	0.22	0.23	0.23
	Our	0.60	0.47	0.38	0.62	0.64

5.2. GAN for image sequences

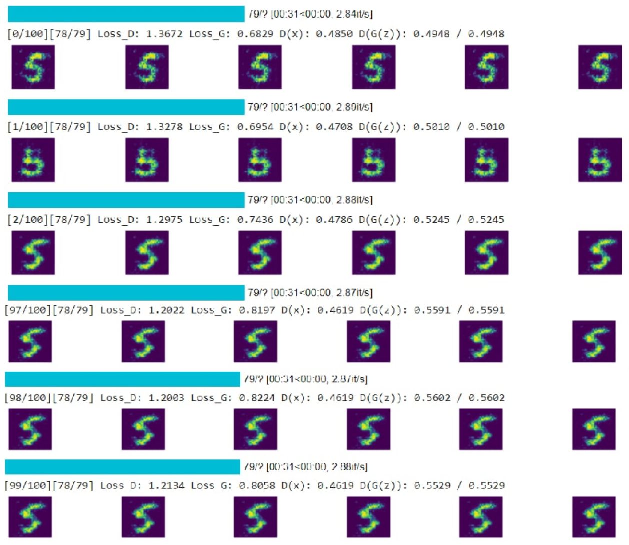
The majority of settings for experiments, such as latent space dimension, different hidden layers dimensions etc., are similar for all the experiments and can be found in our github. If any parameters will differ from base ones from github, we will clarify it in the corresponding section separately. Note: for pictures, printed images are some random samples from generator output on this step (thee are different number of images on different pictures just because we have changed this parameter during all process).

5.2.1. Checking correctness of the architecture

First of all we decided to check, whether our GAN, modified with CNN work on single elements sequences. We have generated a dataset of MNIST ”5”s (we have chosen number ”5”, because it has quite complex structure) and GAN easily trained on such data (generator gave good objects and Discriminator did well in distinguishing normal ones from fake) after 10-20 epochs.

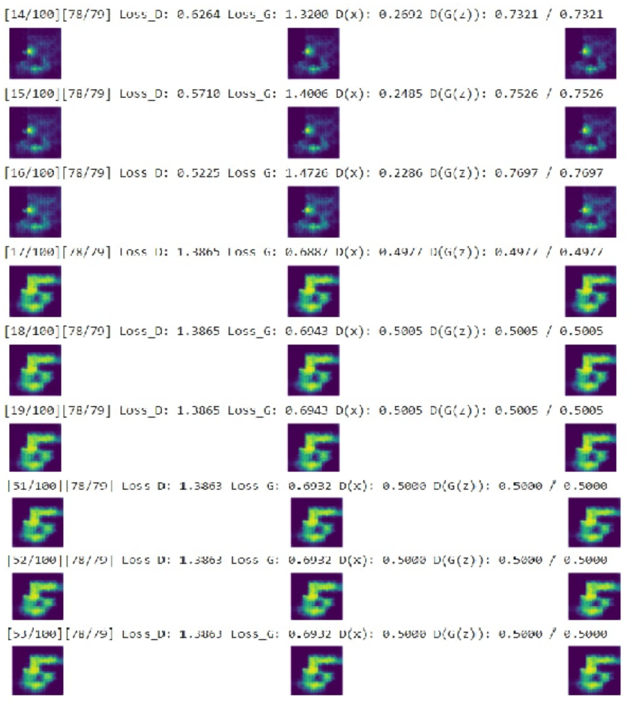
5.2.2. Single number sequences

After we convinced that the architecture works correctly, we expanded sequence length for 3, but still used only single number in all training sequences. We first tested same number of optimization steps per epoch for both Discriminator and Generator, and came to the situation, when Dicriminator dominated too much, so Generator couldn’t learn to make good samples. So we’ve changed it and used 5 Generator optimization steps on 1 Discriminator optimization step during an epoch. It leads to the results, illustrated in Figure 1. It converges quite fast to appropriate result (it took 5-10 epochs) and both Discriminator and Generator "agreed" on it for all subsequent epochs (up to 100).



```
79/? [00:31<00:00, 2.84it/s]
[0/100][78/79] Loss_D: 1.3672 Loss_G: 0.6829 D(x): 0.4850 D(G(z)): 0.4948 / 0.4948
79/? [00:31<00:00, 2.89it/s]
[1/100][78/79] Loss_D: 1.3278 Loss_G: 0.6954 D(x): 0.4708 D(G(z)): 0.5012 / 0.5010
79/? [00:31<00:00, 2.88it/s]
[2/100][78/79] Loss_D: 1.2975 Loss_G: 0.7436 D(x): 0.4786 D(G(z)): 0.5245 / 0.5245
79/? [00:31<00:00, 2.87it/s]
[97/100][78/79] Loss_D: 1.2022 Loss_G: 0.8197 D(x): 0.4619 D(G(z)): 0.5591 / 0.5591
79/? [00:31<00:00, 2.87it/s]
[98/100][78/79] Loss_D: 1.2003 Loss_G: 0.8224 D(x): 0.4619 D(G(z)): 0.5602 / 0.5602
79/? [00:31<00:00, 2.88it/s]
[99/100][78/79] Loss D: 1.2134 Loss G: 0.8058 D(x): 0.4619 D(G(z)): 0.5529 / 0.5529
```

Figure 1. Generator 5 steps and Discriminator 1 step per epoch for sequences of MNIST "5"s



```
[14/100][78/79] Loss_D: 0.6264 Loss_G: 1.3200 D(x): 0.2692 D(G(z)): 0.7321 / 0.7321
[15/100][78/79] Loss_D: 0.5710 Loss_G: 1.4006 D(x): 0.2485 D(G(z)): 0.7526 / 0.7526
[16/100][78/79] Loss D: 0.5225 Loss G: 1.4726 D(x): 0.2286 D(G(z)): 0.7697 / 0.7697
[17/100][78/79] Loss D: 1.3865 Loss G: 0.6887 D(x): 0.4977 D(G(z)): 0.4977 / 0.4977
[18/100][78/79] Loss_D: 1.3865 Loss_G: 0.6943 D(x): 0.5005 D(G(z)): 0.5005 / 0.5005
[19/100][78/79] Loss_D: 1.3865 Loss_G: 0.6943 D(x): 0.5005 D(G(z)): 0.5005 / 0.5005
[51/100][78/79] Loss D: 1.3863 Loss G: 0.6932 D(x): 0.5000 D(G(z)): 0.5000 / 0.5000
[52/100][78/79] Loss D: 1.3863 Loss G: 0.6932 D(x): 0.5000 D(G(z)): 0.5000 / 0.5000
[53/100][78/79] Loss_D: 1.3863 Loss_G: 0.6932 D(x): 0.5000 D(G(z)): 0.5000 / 0.5000
```

Figure 2. After 17th iteration we observe same output for generator and almost the same score, very close to or even exactly 0.5 for Discriminator. And it will not change during further epochs

Sometimes we got a situation, where generator starts giving good results, but after few iterations it started behaving absolutely randomly. We assumed, that after reaching some local minimum, generator 'jumped' out of it, due to big learning rate. So we have also made few experiments with changing not number of generator and discriminator fractions, but adding exponential scheduler for optimizers. However, it leads to fast convergence to recognizable, but not good enough images (noisy + thick number borders) and following inaction from both Generator and Discriminator (Figure 2).

Obtaining appropriate results for short sequences we tried to expand them for 15 images per sequence. We fixed 1 step per epoch for Discriminator and 5 for Generator, as they gave the best result during previous experiments. After few (15-20) training epochs Generator of our model again converged to some recognizable but noisy sample (Figure 3)

5.2.3. All MNIST Numbers sequences

After obtaining acceptable results for single number sequences we tried to train our MNIST GAN on different sequences of numbers (in one sequence all numbers are the same, but this number is different for different sequences).

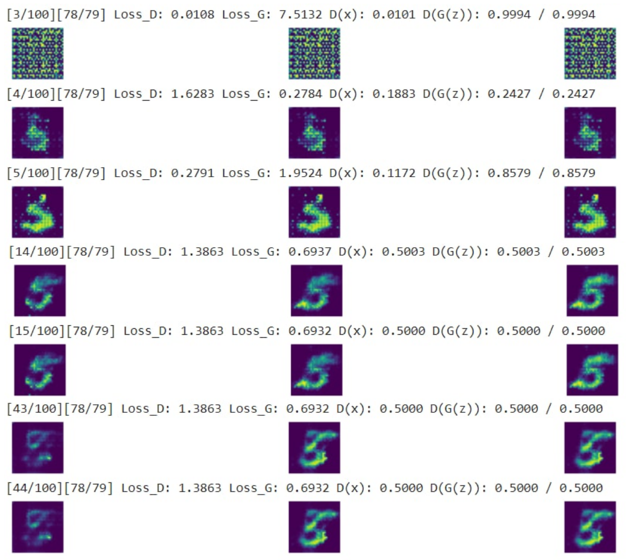
We tried to train it with different parameters (varied Discriminator and Generator rounds, latent dimension, tried using and not using schedulers). However, training process was unstable and Generator proposed some strange results, all of which, however, were concentrated in the center. We suppose, that this form is very similar to sum of all numbers samples (comparison can be seen from Figure 4

As the training procedure did not reach good results, we did not try it on test data with anomalies.

5.2.4. Anomaly detection on single number (train) sequences

However, as for single number sequences the training process was quite stable, we checked our MNIST GAN for this task. Table 5 describes train and test datasets and Table 6 illustrates obtained metrics.

Summarizing received metrics, we can see, that our model has many false positives (therefore, we obtained small recall). Small precision says, that there are not many false positives (which means that out GAN trained to figure out normal objects good).



```
[3/100][78/79] Loss_D: 0.0108 Loss_G: 7.5132 D(x): 0.0101 D(G(z)): 0.9994 / 0.9994
[4/100][78/79] Loss_D: 1.6283 Loss_G: 0.2784 D(x): 0.1883 D(G(z)): 0.2427 / 0.2427
[5/100][78/79] Loss_D: 0.2791 Loss_G: 1.9524 D(x): 0.1172 D(G(z)): 0.8579 / 0.8579
[14/100][78/79] Loss_D: 1.3863 Loss_G: 0.6937 D(x): 0.5003 D(G(z)): 0.5003 / 0.5003
[15/100][78/79] Loss_D: 1.3863 Loss_G: 0.6932 D(x): 0.5000 D(G(z)): 0.5000 / 0.5000
[43/100][78/79] Loss_D: 1.3863 Loss_G: 0.6932 D(x): 0.5000 D(G(z)): 0.5000 / 0.5000
[44/100][78/79] Loss_D: 1.3863 Loss_G: 0.6932 D(x): 0.5000 D(G(z)): 0.5000 / 0.5000
```

Figure 3. Generator output and Discriminator scores for sequences of 15 images

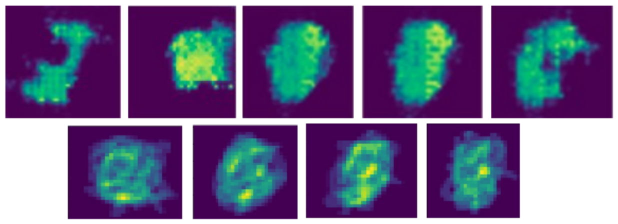


Figure 4.
Top line: Generator samples
Bottom line: sums of 10 (one number from every class) random numbers from training dataset

6. Conclusions

6.1. MAD-GAN

Summing up, we have implemented the MAD-GAN model from the paper using PyTorch, conducted several experiments on sequential data and compared results with authors. Some of the obtained metrics are similar, while some differs noticeably. One of the reasons for metrics difference is that authors did not indicate optimal hyperparameters for metrics calculation in the paper. Also the source code of the paper has several unsolved issues, which lead to non-working code, so our reimplementation of these moments might differ from paper ones.

Table 5. General Information about Single number sequences MNIST Datasets

Item	Dataset	Info
Training size	100000	10000 sequences of ten ”1”s
Testing size	100000	10000 sequences of ten ”1”s and ”7”s, (where 7 is anomaly), with anomaly ratio = 0.1-0.5
$N_{rate}(\%)$	77.64	

Table 6. Metrics for Single number sequences MNIST Datasets

Metric	Value
Accuracy	0.65
Precision	0.88
Recall	0.35
F-1	0.50

6.2. GAN for image sequences

We have also expanded MAD-GAN architecture for working with MNIST image sequences. We have trained a model for single number sequence of ”1”s and tested it on data with ”7”s as anomalies. There is a lot of work for the future research on selection of hyperparameters, tuning architecture, etc., however baseline results were achieved. As for sequences of all 10 MNIST numbers, we could not obtain appropriate model parameters for training. Nevertheless, possible reasons of such results were described and further model tuning could lead to acceptable result.

References

Kdd dataset. URL http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

Swat dataset. URL https://itrust.sutd.edu.sg/itrust-labs_datasets/dataset_info/.

Wadi dataset. URL https://itrust.sutd.edu.sg/itrust-labs_datasets/dataset_info/.

Braei, M. and Wagner, S. Anomaly detection in univariate time-series: A survey on the state-of-the-art. 2020.

Gao, J., Song, X., Wen, Q., Wang, P., Sun, L., and Xu, H. Robusttad: Robust time series anomaly detection via decomposition and convolutional neural networks. 2020.

Geiger, A., Liu, D., Alnegheimish, S., Cuesta-Infante, A., and Veeramachaneni, K. Tadgan: Time series anomaly detection using generative adversarial networks. 2020.

Li, D., Chen, D., Shi, L., Jin, B., Goh, J., and Ng, S. MAD-GAN: multivariate anomaly detection for time series data with generative adversarial networks. volume abs/1901.04997, 2019. URL http://arxiv.org/abs/1901.04997.

Nguyen, H., Tran, K., Thomassey, S., and Hamad, M. Forecasting and anomaly detection approaches using lstm and lstm autoencoder techniques with the applications in supply chain management. volume 57, pp. 102282, 2021. doi: https://doi.org/10.1016/j.ijinfomgt.2020.102282. URL https://www.sciencedirect.com/science/article/pii/S026840122031481X.

Qingsong Wen, Kai He, L. S. Y. Z. M. K. H. X. Robustperiod: Time-frequency mining for robust multiple periodicity detection. 2020.

Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. 2016.

Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U., and Langs, G. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. 2017.

Schlegl, T., Seeböck, P., Waldstein, S., Langs, G., and Schmidt-Erfurth, U. f-anogan: Fast unsupervised anomaly detection with generative adversarial networks. volume 54, 01 2019. doi: 10.1016/j.media.2019.01.010.

Sovilj, D., Budnarain, P., Sanner, S., Salmon, G., and Rao, M. A comparative evaluation of unsupervised deep architectures for intrusion detection in sequential data streams. volume 159, pp. 113577, 2020. doi: https://doi.org/10.1016/j.eswa.2020.113577. URL https://www.sciencedirect.com/science/article/pii/S0957417420304012.

Yoon, J., Jarrett, D., and van der Schaar, M. Time-series generative adversarial networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf.

Zhu, J., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. volume abs/1703.10593, 2017. URL http://arxiv.org/abs/1703.10593.

A. Team member's contributions

Explicitly stated contributions of each team member to the final project.

Mezentsev Gleb (28% of work)

- Reimplementing main algorithm from the paper on PyTorch
- Experimenting with model parameters on single number MNIST sequences
- Preparing the GitHub Repo
- Reviewed and corrected this report

Leshchev Dmitrii (24% of work)

- Reviewing literate on the topic (all Related work section)
- Experimenting with model parameters on single number MNIST sequences
- Preparing Sections 1, 2, 3, 5.2, 6 of this report.
- Making final presentation

Baianov Valerii (24% of work)

- Requested data (WADI and SWaT) from owners
- Experimenting with finding best parameters for MAD-GAN
- Implementing code for obtaining metrics for plots and tables for comparison with the paper
- Preparing Sections 4 and 5.1 of this report

Dubovitskiy Ilya (24% of work)

- Worked with initial paper code and tried to solve issues with it
- Making final presentation
- Preparing final video
- Experimenting with anomaly detection on MNIST sequences
- Reviewed and corrected this report

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

   Students' comment: There was a code of the paper on the github. However, it didn't work (which is also proved by 10+ issues of such problem from other users), so we reimplemented it on PyTorch. Also we were provided with CVAE code for generating sequences, which we used as a base one for our generation functions.

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

4. A complete description of the data collection process, including sample size, is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

9. The exact number of evaluation runs is included.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

10. A description of how experiments have been conducted is included.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

11. A clear definition of the specific measure or statistics used to report results is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

12. Clearly defined error bars are included in the report.

- ☐ Yes.
- ☐ No.
- [✓ Not applicable.]

Students' comment: None

13. A description of the computing infrastructure used is included in the report.

- [✓ Yes.]
- ☐ No.
- ☐ Not applicable.

Students' comment: None

C. Appendix plots

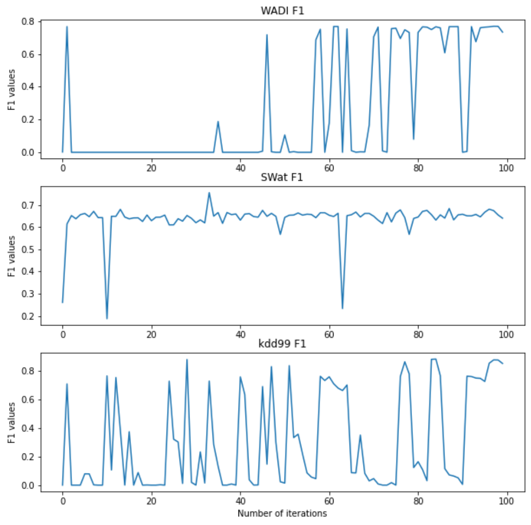


Figure 5. F1 score graphs for three datasets

The volume agreed with responsible TA

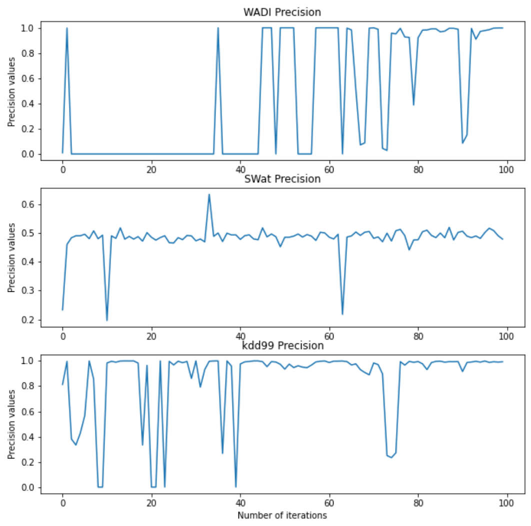


Figure 6. Precision score graphs for three datasets

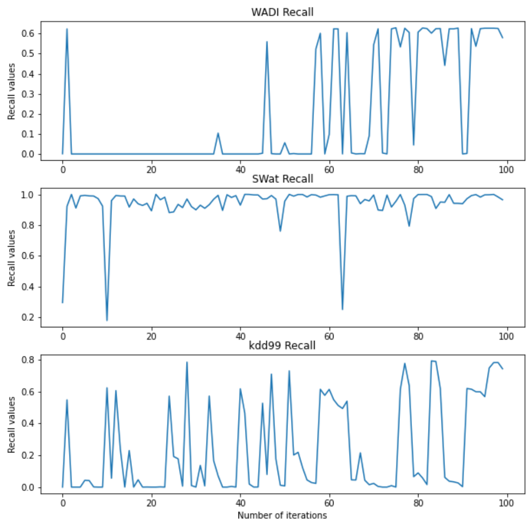


Figure 7. Recall score graphs for three datasets