# FEDERATED UNSUPERVISED ANOMALY DETECTION

HOMEWORK F OR THE COMPLEX FEDERATED MODELS IN MACHINE LEARNING COURSE OF BME (VIMIAV25)

**Máté Baranyi**
Department of Stochastics
BME
baranyim@math.bme.hu

**Attila Kádár**
Department of Networked Systems and Services
BME
attilaka98@gmail.com

**Marcell Nagy**
Department of Stochastics
BME
marcessz@math.bme.hu

June 10, 2021

**ABSTRACT**

Unsupervised anomaly detection in high dimensional data is a difficult problem from many different angles. Density-based algorithms are popular in this framework. We tested some existing algorithms of this type from the point of view of federated learning. We propose and implement federated workflows for the tested methods. The workflows assume horizontally distributed data across many parties. We evaluated the algorithmic modifications on some publicly available datasets, and studied how the federated approach influenced the efficiency of the algorithms on these.

***Keywords*** federated learning · anomaly detection · unsupervised learning

## 1 Algorithms

### 1.1 Autoencoder and some variants

Autoencoder (AE), also known as auto-associative neural network [1] is a feed-forward network trained to recall the input in its output. The first half of the autoencoder approximates the function $f$ that maps the input space to the lower-dimensional space while the second half approximates the inverse function $f^{-1}$ since the autoencoder is trained to have an output as similar to its input as possible. The architecture of a simple fully connected autoencoder can be seen in Figure 1a. In this example, we have 10 neurons on the input and output layers, and two hidden layers consisting of 7 neurons each, meanwhile the encoded layer has 5 neurons. This smallest middle layer corresponds to the reduced dimensional space. The size of the middle layer is either a user specified parameter or adjusted automatically to the smallest size that is still meaningful, defined by the reconstruction error or by other reasonable metrics.

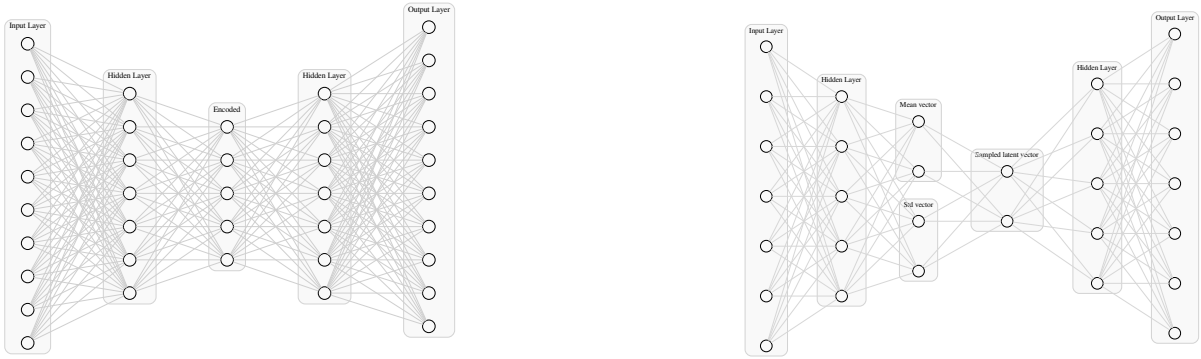
#### 1.1.1 Variational variant

A drawback of the regular autoencoder is that the mapping of the autoencoder does not preserve the $L_2$-distance and the data points are "spread out" in the encoded space. To deal with this, the variational autoencoder (VAE) approach [2] was introduced.

The VAE makes an assumption about the distribution of the latent/encoded representation. If the data vector is denoted by $\mathbf{X}$ and the encoded vector as $\mathbf{E}$, then it is assumed that $\mathbf{X}$ is coming from the distribution $p_\theta\left(\mathbf{X} \mid |\mathbf{E}\right)$. The encoder part tries to learn an approximation of the distribution $q_\phi\left(\mathbf{E}|\mathbf{X}\right)$ with respect to the decoder part which adjusts the distribution $p_\theta\left(\mathbf{X} \mid |\mathbf{E}\right)$ based on a sampling from $q_\phi\left(\mathbf{E} \mid |\mathbf{X}\right)$, and so on, using the error function of Eq. 1. Usually it is assumed that both $p$ and $q$ are factorized[1] multivariate Gaussian distributions (with parameters $\theta$ and $\phi$ respectively). The loss function is usually two-folded, depends on the given $\mathbf{X}$ and the parameters $\theta$ and $\phi$ of the two conditional distributions:

$$\begin{aligned} \mathrm{L}\left(\mathbf{X}, \theta, \phi\right) = d_{KL}\left(q_\phi\left(\mathbf{E} \mid |\mathbf{X}\right)||r\left(\mathbf{E}\right)\right)) \quad &+ \mathbb{E}_{q_\phi(\mathbf{E}||\mathbf{X})}\left(\log p_\theta\left(\mathbf{X} \mid |\mathbf{E}\right)\right) \\ \mathrm{Loss} = \mathrm{Loss}_{\mathrm{KL}} \quad &+ \mathrm{Loss}_{\text{reconstruction}}, \end{aligned} \tag{1}$$

[1] Factorized in the sense that the coordinates are uncorrelated (diagonal covariance matrix), which keeps the calculations tractable since we have less parameters to approximate.




(a) A simple autoencoder.

(b) A simple variational autoencoder.

Figure 1: The AE architectures

where $r(\mathbf{E})$ is the apriori distribution chosen for the latent vector space, usually it is the standard normal distribution. The first part of the loss behaves as a regulator term, the second term actually reduces to the usual MSE. In [3] an additional factor is introduced in the error:

$$\text{Loss} = \beta \cdot \text{Loss}_{\text{KL}} + \text{Loss}_{\text{reconstruction}},$$

where $\beta$ is a scalar. The $\beta$ factor can control which part of the error is more important in the given scenario. Choosing $\beta$ as zero essentially reduces the VAE to a regular AE. If the decoder's output (the other distribution) does not reconstruct the data well (the second part of the loss is large), it means that the decoder parameterizes a likelihood distribution that does not place much probability mass on the true data, a smaller $\beta$ can help in this case.

Since the encoder's output is actually a distribution, the encoded vector will be a sampling from this distribution, thus it will be random. The architecture of a simple variational autoencoder is depicted in Figure 1b.

Autoencoder and its variants have also been utilized as anomaly detection tools. There are two main types: regular AE [1] try to learn a latent low-dimensional representation of the dataset, whereas VAE [2] try to learn the parameters of the data-generating process assumed behind the dataset. In both case the reconstruction error of the model is used as the outlier score.

The deep neural network-based anomaly detection models usually scale well, have online variants, and work in high dimension with a well-defined objection function. Their main drawback is the number of hyper-parameters, most of which require fine-tuning. Most of these models were build around a specific dataset, so their ability to generalize to other datasets depends on the chosen hyper-parameters.

### 1.2 LODA

LODA was introduced in [4]. The model approximates the joint probability in a transformed multidimensional space by using a collection of one-dimensional histograms, where each one-dimensional histogram is constructed on the input space projected onto a randomly generated vector. The authors claim the followings about their model:

- A large number of samples can be processed in real-time.
- It works in environments where the data stream is subject to concept drift and the detector needs to be updated on-line[2].
- It can handle missing data, thus it is practical in domains with sensor outages.[3]
- It can rank features according to their contribution to sample's "anomalousness".
- Hyper-parameter free.[4]

The algorithm works as follows:

[2]Only works if on-line histogram estimation is utilized

[3]True but it also has to be properly implemented.

[4]Actually there are at least 2 parameters, and some other parameter choices feel arbitrary.

1. Input: data samples $\mathbf{X}$ of size $N \times D$ composed of $\left\{x_i \in \mathbb{R}^D\right\}_{i=1}^N$ .
2. Generate a random projection matrix $\mathbf{W}$ of size $K \times D$ composed of vectors $\left\{\omega_i \in \mathbb{R}^D\right\}_{i=1}^K$,
   (a) $\frac{1}{\sqrt{D}}$ elements are coming from $\mathcal{N}(0,1)$, the others are set to zero,
   (b) typically $K \gg D$.
3. Each vector projects the data onto a one-dimensional random space, thus after the projection, the data is transformed into a new $k$ dimensional random/latent space:
   (a) the projected data is $\mathbf{Z} := \mathbf{X}\mathbf{W}^T$ of size $N \times K$;
   (b) these $K$ projected dimensions are assumed to independent of each other.
4. The density in this new space is assumed to be proportional to the one in the original space.
   (a) Utilizing the previous independence assumption the density is estimated in this new space as product of one-dimensional histogram estimates.
5. The output of LODA for a single ($d$-dim) point $x$ is the scaled estimated log-probability of this point in the transformed space:

$$\text{score}(x) := -\log f\left(x\omega_1^T, \ldots, x\omega_k^T\right) \sim -\sum_{i=1}^{K} \log \hat{f}_i(x\omega_i^T),$$

   where $\hat{f}_i()$ denotes the experimental pdf of the $i$th projected dimension obtained by histogram based approximation and we used the independence assumption in the approximate step.

### 1.2.1 Implementation of LODA

We have found 3 publicly available implementations of the algorithm: pyod and pysad (a fork of pyod for streaming data) in Python, and one other MatLab implementation. We work with our own fork of the pyod implementation. The implementation is simple, easy to improve, so we have many parameters:

- K: no. of projected dimensions,
- bins: no. of (equidistant) bins for the histograms.
- manual selection of the number of non-zero elements for the random projection vectors,
- ability to ($L_2$-)normalize the projection vectors to one.

## 1.3 Generative Adversarial Network

Generative Adversarial Networks (GANs) are able to model complex high-dimensional distributions of real-world data sets. Hence, it makes them applicable to solve outlier detection tasks. In recent years, researchers have been studying how to leverage the GANs for anomaly detection in different settings, e.g., in unsupervised, semi-supervised, and supervised learning.

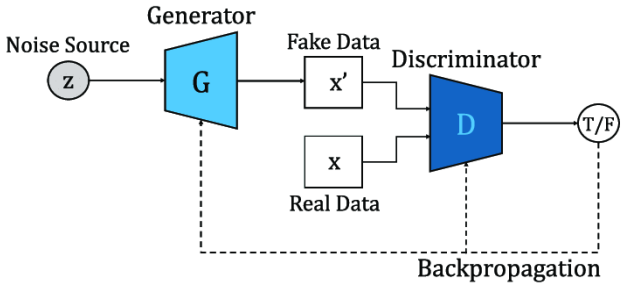



Figure 2: The framework of a simple GAN.

The GANs from a relatively new class of machine learning, the idea and concept have been introduced in 2014 by Goodfellow *et al.* [5]. The idea of GAN is that two neural networks (a generator and a discriminator) contest with each other in a game in the form of a zero-sum game, and the goal of this technique is to learn to generate new data with the same statistics as the training set. The generative network generates candidates while the disciminator network

evaluates them [5]. The generative network learns to map from a latent space to a data distribution of interest, while the discriminator distinguishes candidates produced by the generator from the true data distribution. The generative network's training objective is to increase the error rate of the discriminative network (i.e., "fool" the discriminator network by producing novel candidates that the discriminator thinks are not synthesized (are part of the true data distribution)) [5]. Figure 2 shows the framework of a GAN model.

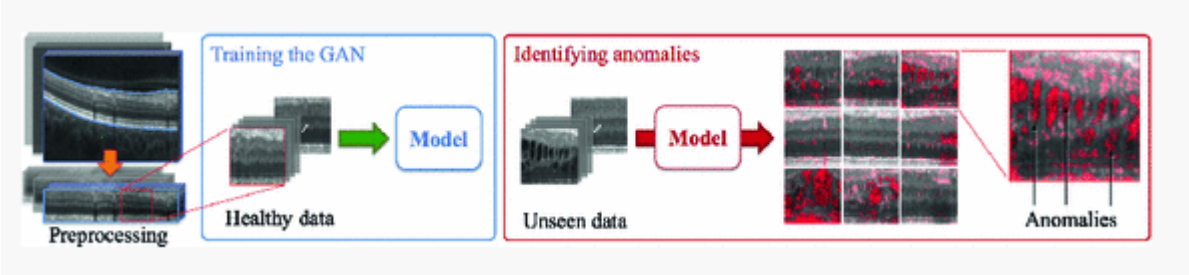



Figure 3: Anomaly detection framework of *AnoGAN*. Generative adversarial training is performed on healthy data and testing is performed on both, unseen healthy cases and anomalous data. The figure is from [6].

The first GAN-based anomaly detection algorithm, called *AnoGAN*, was proposed by Schlegl *et al.* [6], who trained the GAN on the normal datapoints, and then used it on unseen dataset to detect anomalies in images. Figure 3 shows the framework of the *AnoGAN* algorithm. In another closely related work Zenati *et al.* [7] proposed a so-called Adversarially Learned Anomaly Detection (ALAD) methodology which is based on bi-directional GANs. Their method uses reconstruction errors based on the adversarially learned features to determine if a data sample is anomalous or not [7]. Recently, Liu *et al.* [8] proposed a Single-Objective Generative Adversarial Active Learning (SO-GAAL) method for outlier detection, which can directly generate informative potential outliers. The framework of the SO-GAAL method can be seen in Figure 4.

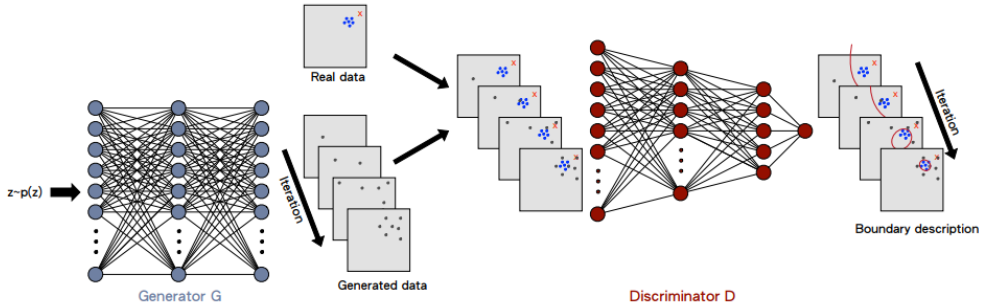



Figure 4: The detection process of the GAN-based outlier detection algorithm introduced in [8]. At the beginning of training, generator $G$ cannot generate a sufficient number of potential outliers around the real data, which causes the discriminator $D$ to describe a rough boundary. But, after several iterations, the generator G gradually learns the generation mechanism of real data and generates an increasing number of informative potential outliers. As a result, the discriminator $D$ can describe a correct boundary around the concentrated data points. The figure is from [8].

In this work, we used the structure of the GAN that has been proposed by Liu *et al.* [8] and their open-source code base (`https://github.com/qarchli/pytorch-gan-for-outlier-detection`) helped us to implement a GAN based anomaly detection algorithm. However, we only adopted the structure and the hyperparameter setting of the generator and the discriminator. We do not generate anomalous datapoints, but we sort of apply the idea of the *AnoGAN* [6], however, we wanted our work to be fully unsupervised, hence the training set contained anomalous points as well. However, we believe that since in real-world datasets the majority of the observations (typically more than 97%) are normal data points, hence the generator will not be able to reproduce the anomalous points after a few epochs of training.

## 2 Federated modifications

### 2.1 Framework of the neural-network-based algorithms

The neural-network-based algorithms (AutoEncoder and GAN) share the same federated learning framework. In this work, we implemented the so-called federated-averaging approach, however, the framework is designed such that it could be easily extended with other federated aggregation methods.

The implementation of our federated averaging process is as follows: the copies of the networks (corresponding to different agents) are initialized exactly the same way, i.e., with the same weights. In every round, where the round is a user-defined parameter, we take the average of the current weights (we believe that this is equivalent to the original federated averaging method, where not the weights but the gradients are averaged). In the models folder[5] all the three approaches (AutoEncoder, GAN, and LODA) have their classes, training, validation, and predict functions, moreover, there is a function and class that manages the data loading. The hyperparameters of the models are defined in the configuration files[6]. These configuration `yml` files can be freely edited.

#### 2.1.1 LODA

Assume that the data samples $\mathbf{X}$ of size $N \times D$ composed of $\left\{x_i \in \mathbb{R}^D\right\}_{i=1}^N$ is distributed horizontally: $\{n_1, \ldots, n_l\}$ data points across $l$ parties. $\sum_{m=1}^{l} n_m = N$, and we have subsamples $\{\mathbf{X}_1, \ldots, \mathbf{X}_l\}$.

The parties share the same projection matrix $\mathbf{W}$. Some differential privacy issues have to be addressed here later on.

Each party $m$ calculates their set of 1D histograms on their projected dataset $\mathbf{X}_m\mathbf{W}^T$:

$$\left\{\hat{f}_1^{(m)}(\cdot), \ldots, \hat{f}_K^{(m)}(\cdot)\right\}. \tag{2}$$

The parties share their histograms with each other for aggregation.

The trivial solution would be: when a party has a datapoint $x$ for evaluation, the aggregated score is calculated as the weighted sum of the individual scores:

$$\text{fed\_score}(x) := \sum_{m=1}^{l} \frac{n_m}{N} \text{score}_m(x).$$

This weighted averaging has the problem that the score (which is only proportional to log-probability) is aggregated from differently scaled histograms, thus every agent will learn scores on a possibly very different range. There are two possible workaround for this:

- The agents agree on a common range to scale there scores into.
  - The problem with this is that a minimal/maximal anomaly score of one agent would not necessary be minimal/maximal for an other agent.
- The agents agree on the parameters of the histogram estimations, and aggregate their results into common histograms.

We have chosen the second approach. The aggregation of multiple histograms (Eq. 2) for one of the projections is not straightforward if the agents choose different number of bins. E.g., aggregating the two histograms of Fig. 5 into a sensible common one is difficult. Even if the number of bins or the width of the bins is the same on two histograms, the possibly different ranges (the underlying samples are coming from) complicate the aggregation. However, if the agents agree on the range and the bin widths of histograms, they can be aggregated easily by stacking, see Fig. 6.

We implemented two rule-of-thumb bin width selectors:

- Rice rule: the number of bins $B = 2\left\lceil \sqrt[3]{N} \right\rceil$, so it is the same in each latent dimension;
- The minimum of the Scott and Freedman–-Diaconis rules (different for each latent dimension $i$) for the bin width:

$$h_i = \min\left(3.49 \cdot \frac{\sigma_i}{\sqrt[3]{N}},\ 2 \cdot \frac{\text{IQR}_i}{\sqrt[3]{N}}\right),$$

[5] `https://github.com/KAttila98/FedLearn/tree/main/models`

[6] `https://github.com/KAttila98/FedLearn/tree/main/configs`

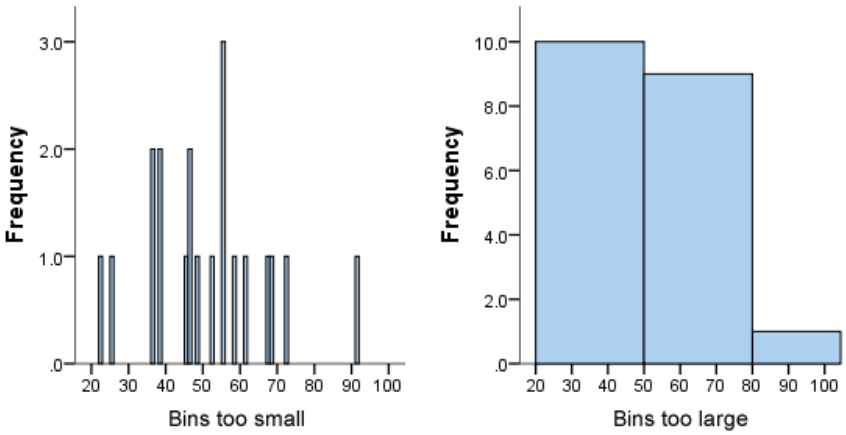



Figure 5: Histograms with different number of bins

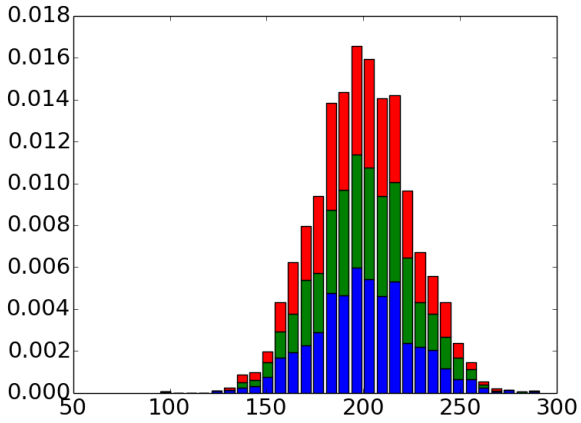



Figure 6: Stacking histograms

where $\sigma_i$ is the sample standard deviation and $\mathrm{IQR}_i$ is the inter-quantile range in the $i$th latent dimension. From the bin widths, we get the number of bins for the $i$th projection as

$$B_i = \left\lceil \frac{\max_i - \min_i}{h_i} \right\rceil,$$

where $\max_i$ and $\min_i$ are the maximum and minimum in the $i$th latent dimension, respectively.

The federated parameters are calculated similarly from the aggregated versions of $N, \mathrm{IQR}, \sigma, \max, \min$.

## 3 Evaluation

We evaluated the algorithms on three real-world benchmark data sets (downloaded from the Outlier Detection Data Sets repository [9], which provides open access to various outlier detection data sets), that are described in Table 1. During the data collection we aimed to select diverse sets, i.e. data sets with different dimensions and proportion of outliers.

### 3.1 LODA on the benchmark datasets

Table 2 show different evaluation metrics on the benchmark datasets' training samples. Note that we have as many training subsamples as many agents (per dataset). The agents' own LODA and the federated version were both evaluated. On these training sets, the agents' LODA sometimes outperforms the federated version. This shows that the LODA in some cases tends to overfit, specially when the sample sizes are smaller. However, on Table 3 we can see the evaluation

Table 1: The downloaded real-world benchmark data sets.

| **Data set** | **#points** | **#dim.** | **#outliers (%)** |
|---|---|---|---|
| Musk | 3,062 | 166 | 97 (3.2%) |
| Satimage-2 | 5,803 | 36 | 71 (1.2%) |
| Shuttle | 49,097 | 9 | 3551 (7%) |

metrics for independent test subsamples (per dataset) where the federated version usually outperforms most agents' own version of LODA; at least in accuracy, but other metrics tend to vary. This has to be further investigated.

Table 2: Validation metrics of LODA agents on their respective training datasets, and the results with the federated LODA after the slashes

(a) Musk

| | **Accuracy** | **Average precision** | **AUC** |
|---|---|---|---|
| LODA0/Fed. | 0.2905/0.9478 | 0.9958/0.1908 | 0.9999/0.7602 |
| LODA1/Fed. | 0.9771/0.9902 | 0.5621/0.89 | 0.9894/0.9962 |
| LODA2/Fed. | 1.0/0.988 | 1.0/0.9203 | 1.0/0.9967 |

(b) Satimage-2

| | Accuracy | Average precision | AUC |
|---|---|---|---|
| LODA0/Fed. | 0.912/0.9866 | 0.9297/0.4756 | 0.9987/0.9774 |
| LODA1/Fed. | 0.9464/0.9866 | 0.9564/0.5481 | 0.9987/0.9873 |
| LODA2/Fed. | 0.9617/0.9981 | 0.9288/0.925 | 0.9878/0.9625 |
| LODA3/Fed. | 0.9856/0.9828 | 0.8621/0.3403 | 0.9925/0.9788 |
| LODA4/Fed. | 0.9962/0.9962 | 0.9521/0.8889 | 0.999/0.9987 |

(c) Shuttle

| | Accuracy | Average precision | AUC |
|---|---|---|---|
| LODA0/Fed. | 0.9287/0.9814 | 0.7743/0.7813 | 0.9915/0.9918 |
| LODA1/Fed. | 0.9348/0.9826 | 0.7671/0.791 | 0.9914/0.9924 |
| LODA2/Fed. | 0.9409/0.978 | 0.7467/0.7751 | 0.9898/0.9915 |
| LODA3/Fed. | 0.9477/0.9817 | 0.7413/0.7839 | 0.9899/0.9923 |
| LODA4/Fed. | 0.9441/0.9808 | 0.7552/0.8032 | 0.989/0.9917 |
| LODA5/Fed. | 0.9554/0.9758 | 0.6995/0.7567 | 0.9865/0.9904 |
| LODA6/Fed. | 0.9604/0.9783 | 0.7358/0.7704 | 0.9878/0.9901 |
| LODA7/Fed. | 0.964/0.9801 | 0.725/0.7751 | 0.988/0.9907 |
| LODA8/Fed. | 0.9787/0.9774 | 0.7541/0.7659 | 0.9892/0.9908 |
| LODA9/Fed. | 0.9869/0.9828 | 0.8946/0.8017 | 0.9945/0.9924 |

### 3.2 GAN on the benchmark dataset

Table 4 shows the performance of the GAN-based anomaly detection algorithm on the Shuttle dataset. Note that the results only show the result of one experiment. We found that the GAN is extremely sensitive to the initialization and its performance is very volatile. Figure 5 shows an example when the performance of the GAN is really low. When the AUC score is below 0.5, it means that is switches the labels of the anomalous and normal points.

### 3.3 Autoencoder on the benchmark dataset

As it can be observed on Figure 6, Autoencoder has produced high performance indicators after one epoch of local training and federated averaging. In this case, the impact of federation is relatively negligible due to the already high local validation metric values. However, it can be seen (especially in terms of Average Precision) that the federated

Table 3: Validation metrics of LODA agents on the common test dataset, and the results with the federated LODA in the last row

(a) Musk

| | **Accuracy** | **Average precision** | **AUC** |
|---|---|---|---|
| LODA0 | 0.1987 | 0.9341 | 0.9974 |
| LODA1 | 0.6352 | 0.0631 | 0.7677 |
| LODA2 | 0.5831 | 0.0478 | 0.6734 |
| FedLODA | 0.9577 | 0.3110 | 0.9206 |

(b) Satimage-2

| | Accuracy | Average precision | AUC |
|---|---|---|---|
| LODA0 | 0.9072 | 0.9333 | 0.9988 |
| LODA1 | 0.9433 | 0.9103 | 0.9980 |
| LODA2 | 0.8883 | 0.4280 | 0.9887 |
| LODA3 | 0.9656 | 0.9444 | 0.9991 |
| LODA4 | 0.9399 | 0.5035 | 0.9928 |
| FedLODA | 0.9794 | 0.2771 | 0.9841 |

(c) Shuttle

| | Accuracy | Average precision | AUC |
|---|---|---|---|
| LODA0 | 0.9304 | 0.7726 | 0.9913 |
| LODA1 | 0.9375 | 0.7596 | 0.9907 |
| LODA2 | 0.9442 | 0.7576 | 0.9906 |
| LODA3 | 0.9503 | 0.7501 | 0.9902 |
| LODA4 | 0.9491 | 0.7504 | 0.9900 |
| LODA5 | 0.9574 | 0.7374 | 0.9893 |
| LODA6 | 0.9658 | 0.7506 | 0.9897 |
| LODA7 | 0.9684 | 0.7273 | 0.9887 |
| LODA8 | 0.9745 | 0.7350 | 0.9891 |
| LODA9 | 0.9760 | 0.7327 | 0.9885 |
| FedLODA | 0.9813 | 0.7780 | 0.9916 |

Table 4: The performance (Accuracy, AUC, and AP) of the federated GAN and its individual components before the federated averaging on the Shuttle dataset.

| | **Accuracy** | **AUC** | **Average Precision** |
|---|---|---|---|
| GAN_0 | 0.9659 | 0.9840 | 0.8552 |
| GAN_1 | 0.9686 | 0.9869 | 0.8649 |
| GAN_2 | 0.9678 | 0.9893 | 0.8912 |
| GAN_3 | 0.9673 | 0.9893 | 0.8558 |
| GAN_4 | 0.9677 | 0.9749 | 0.8243 |
| FedGAN | 0.9652 | 0.9884 | 0.8539 |

version outperforms the local performance of several clients. Figure 7 shows that the performance of AE decreases after multiple local training epochs. According to our hypothesis, the reason for this is that, as the number of epochs increases, the AE can reproduce anomalous points with a smaller reconstruction error, and thus perceive them as normal points. In this case, the effect of federated learning is amplified (see AUC and Average Precision).

Table 5: An example for the low performance the federated GAN and its individual components before the federated averaging on the Shuttle dataset.

| | **Accuracy** | **AUC** | **Average Precision** |
|---|---|---|---|
| GAN_0 | 0.8522 | 0.0897 | 0.0491 |
| GAN_1 | 0.8564 | 0.0985 | 0.0466 |
| GAN_2 | 0.8569 | 0.1072 | 0.0648 |
| GAN_3 | 0.8536 | 0.0874 | 0.0479 |
| GAN_4 | 0.8590 | 0.0957 | 0.0415 |
| FedGAN | 0.8531 | 0.0950 | 0.0409 |

Table 6: Performance the federated AE and its individual components before the federated averaging on the Shuttle dataset after one training epoch.

| | **Accuracy** | **AUC** | **Average Precision** |
|---|---|---|---|
| AE_0 | 0.9648 | 0.9953 | 0.8461 |
| AE_1 | 0.9623 | 0.9942 | 0.8456 |
| AE_2 | 0.9631 | 0.9951 | 0.8541 |
| AE_3 | 0.9639 | 0.9948 | 0.8480 |
| AE_4 | 0.9651 | 0.9955 | 0.8629 |
| FedAE | 0.9623 | 0.9943 | 0.8508 |

## 4 Summary

In this work, we developed, more specifically made three algorithms able to detect anomalies in real-world data in a federated manner. We modified the so-called LODA (Lightweight on-line detector of anomalies) algorithm and two neural network algorithms: an AutoEncoder and a Generative Adversarial Network. We have found that the best-performing model was the AutoEncoder, however, it is important not to run the learning process for too long, otherwise the model will learn the representation of the anomalous points as well. Note that it also holds for the GAN network, i.e., if it learns for too long, it learns how to generate anomalous points, in other words, it also learns the distribution of the whole dataset including the anomalous points as well. The GAN could perform really well, as the average precision of Table 4 shows, however, we found that the performance is very unstable. On the other hand, the federated LODA provides a robust good performance in all scenarios.

Table 7: Performance the federated AE and its individual components before the federated averaging on the Shuttle dataset after three training epochs.

| | **Accuracy** | **AUC** | **Average Precision** |
|---|---|---|---|
| AE_0 | 0.9423 | 0.9475 | 0.6568 |
| AE_1 | 0.9407 | 0.8384 | 0.6375 |
| AE_2 | 0.9443 | 0.9581 | 0.7188 |
| AE_3 | 0.9419 | 0.9350 | 0.6636 |
| AE_4 | 0.9468 | 0.9743 | 0.7387 |
| FedAE | 0.9443 | 0.9658 | 0.6953 |

## References


[1] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):11, 1991.

[2] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *arXiv:1312.6114 [cs, stat]*, 5 2014. arXiv: 1312.6114.

[3] Irina Higgins, Loïc Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *ICLR*, 2017.

[4] Tomáš Pevný. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, 102(2):275–304, February 2016.

[5] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[6] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International conference on information processing in medical imaging*, pages 146–157. Springer, 2017.

[7] Houssam Zenati, Manon Romain, Chuan-Sheng Foo, Bruno Lecouat, and Vijay Chandrasekhar. Adversarially learned anomaly detection. In *2018 IEEE International conference on data mining (ICDM)*, pages 727–736. IEEE, 2018.

[8] Yezheng Liu, Zhe Li, Chong Zhou, Yuanchun Jiang, Jianshan Sun, Meng Wang, and Xiangnan He. Generative adversarial active learning for unsupervised outlier detection. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1517–1528, 2019.

[9] Shebuti Rayana. ODDS library, 2016.