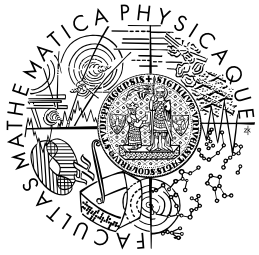

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Ondřej Měkota

Anomaly Detection Using Generative Adversarial Networks

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis:	RNDr. Jiří Fink, Ph.D.
Study programme:	Computer Science
Study branch:	General Computer Science

Prague 2019

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date .....

I would like to thank my advisor RNDr. Jiří Fink, Ph.D. for his patience, time and his valuable comments, and suggestions. I would also like to thank my family for their support.

Title: Anomaly Detection Using Generative Adversarial Networks

Author: Ondřej Měkota

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Jiří Fink, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Generative adversarial networks (GANs) are able to capture distribution of its inputs. They are thus used to learn the distribution of normal data and then to detect anomalies, even if they are very rare; e.g. Schlegl et al. (2017) proposed an anomaly detection method called AnoGAN. However, a major disadvantage of GANs is instability during training. Therefore, Arjovsky et al. (2017) proposed a new version, called Wasserstein GAN (WGAN).

The goal of this work is to propose a model, utilizing WGANs, to detect fraudulent credit card transactions. We develop a new method called AnoWGAN+e, partially based on AnoGAN, and compare it with One Class Support Vector Machines (OC-SVM) (Schölkopf et al. (2001)), k-Means ensemble (Porwal et al. (2018)) and other methods. Performance of studied methods is measured by area under precision-recall curve (AUPRC), and precision at different recall levels on credit card fraud dataset (Pozzolo (2015)). AnoWGAN+e achieved the highest AUPRC and it is 12% better than the next best method OC-SVM. Furthermore, our model has 20% precision at 80% recall, compared to 8% precision of OC-SVM, and 89% precision at 10% recall as opposed to 79% of k-Means ensemble.

Keywords: anomaly detection, generative adversarial networks, neural network, deep learning

Contents

Introduction

The need for anomaly detection can be seen in many aspects of our lives. In medical science, researches and doctors want to identify ill patients using various indicators. For example, trying to diagnose cancer from tissue screening can be done using outlier detection methods. A tissue of patients who have cancer will be considered anomalous and the rest normal. Automotive industry is another field where outlier detection is essential. Engineers want to know whether there is a malfunction or irregularity in some part of an assembled car. Internet companies are frequently faced with problems concerning their networks caused by intruders, those also can be revealed using anomaly detection methods.

Often, anomalies are rare and therefore it is difficult for supervised machine learning algorithms to capture their distribution. For that reason, unsupervised methods are utilized. They either learn from datasets consisting of only normal samples or from all, unannotated samples. Both of those methods can work, one that learns from normal examples is usually able to distinguish anomalies more accurately.

Banks need to be able to detect fraudulent credit card transactions. According to Mehrota et al. [1], credit cards of millions of people have been compromised in 2013. Unauthorized access to credit cards costs billions of dollars annually [1]. Problem is that actual frauds do not occur very often among all transactions, this means supervised algorithms would be hard to utilize. Thus it is necessary to use an unsupervised solution.

With the prevalence of deep learning in the last couple of years, scientists started to explore the possibility of deploying deep learning and neural networks in the area of anomaly detection [2]–[5]. We chose to study anomaly detection methods using Generative adversarial networks (GANs) because they show good results in imagery data [2] and we wanted to try them on non-visual data. In this thesis, we propose an unsupervised model based on GAN for detecting fraudulent credit card transactions.

Goals of this thesis are:

- Study methods described in articles by Schlegl et al. [2] and Zenati et al. [3].
- Propose alternative training procedure of encoder used in [3] with better stability.
- Set suitable hyperparameters so that studied methods are able to perform fraud detection on Credit Card Fraud Dataset (CCFD) [6].
- Compare the performance of studied methods with the performance of Isolation Forest [7], k-Means ensemble [8] and One Class Support Vector Machines (OC-SVM) [9].
- Analyze results and discuss the advantages and disadvantages of compared methods.

Contribution of this thesis is improving the method described in article [2], by switching GAN for Wasserstein GAN and introducing an encoder to latent

space; hence, making it more precise and much more faster during evaluation. Our method overcomes the instability of [3] by training the encoder independently of the training of the generator and the critic.

Main **result** of this thesis is introducing a new method for detecting anomalies. We show that our model (AnoWGAN+e) achieves higher area under precision-recall curve (AUPRC) than k-Means ensemble, Isolation Forest and OC-SVM, which is the state-of-the-art to the best of our knowledge. Our method has substantially higher precision at some recall levels, see table 5.2.

Structure

The thesis is divided into five chapters. Chapter 1 describes machine learning, neural networks, and Generative adversarial networks, which are the main subject of our work. Furthermore, it provides the reader insight into some areas of statistics which are necessary to understand the underlying principles of anomaly detection methods.

Chapter 2 shows literature review of anomaly detection methods – One Class Support Vector Machines [9], k-Means ensemble [8], Isolation Forest [7], and GAN-based methods [2], [3], which are further discussed in Chapter 3.

Chapter 3 contains a description of our model anomaly detection model, called *AnoWGAN*.

Chapter 4 contains information about CCFD, about conducted experiments, particular setting of hyperparameters of our model to detect fraudulent credit card transaction in CCFD and implementation details.

Chapter 5 is about measurements of performance of studied methods, their comparison and discussion about the results.

1. Background

In this chapter, we discuss some basic techniques, algorithms, and tools necessary to understand methods used for anomaly detection in this thesis. Firstly we talk about machine learning and neural networks, what they are and what kind of tasks can they solve. Next section's concern are statistical tools whose knowledge is essential to understand anomaly detection methods used in this thesis. Finally, in the last section, we show what Generative adversarial networks are, what problems they have and how to solve those problems.

1.1 Machine Learning

Mitchell [10] defines machine learning as "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." We do not provide formal definition of experience E, tasks T and performance measure P but rather informally explain what they mean.

The task is usually either *classification* or *regression*. But many other, not so extensively used, tasks exist [11]. In classification, one tries to find a mapping from some feature space $\mathbb{R}^n$ to k categories [11]. Regression is a task where the objective is finding a mapping from $\mathbb{R}^n$ to $\mathbb{R}$.

In order to be able to evaluate a model, we need a performance measure or a loss function. This can be for example accuracy, precision or error rate in case of classification tasks [11]. For regression, we usually utilize some continuous distance between the predicted value and the real value.

Experience is the data that learning algorithm is allowed to see during training. Learning algorithms can be categorized to two categories: supervised and unsupervised.

Unsupervised algorithms are only provided a dataset containing features that are generated by an underlying distribution. We usually aim to learn this underlying distribution [11].

Supervised methods also have access to *label* of class, when performing classification, or to *target* when performing regression.

The division between these categories is not clear and sometimes it raises a call for another category: semi-supervised algorithm, where some examples do have a label or a target and some do not. Definitions of these categories vary in literature [12].

1.1.1 Neural Networks

Neural networks, more specifically *deep* neural networks, are a class of machine learning algorithms. The aim of neural network is to find approximation of function $y = f^*(\boldsymbol{x})$, which maps inputs $\boldsymbol{x}$ to the output y, with parametrized function $y = f(\boldsymbol{x}, \boldsymbol{\theta})$. The inputs are fed through multiple layers of the network: $y = f^{(n)}(\cdots f^{(2)}(f^{(1)}(\boldsymbol{x})))$. All of those functions are parametrized by different parameters.

The more layers network has, the deeper we say it is. Figure 1.1 shows a three-layer neural network. Each edge (which are oriented from inputs to outputs) is associated with *weight* w and each neuron (grey circles in the figure) with *bias* b. Value at the start of an edge is multiplied by the edge's weight and summed with the bias of the neuron at its end. For example, the output of the first neuron in hidden layer (in Figure 1.1) looks like this

$$f^{(1)}(x_1 * w_{11} + x_2 * w_{21} + b_1).$$

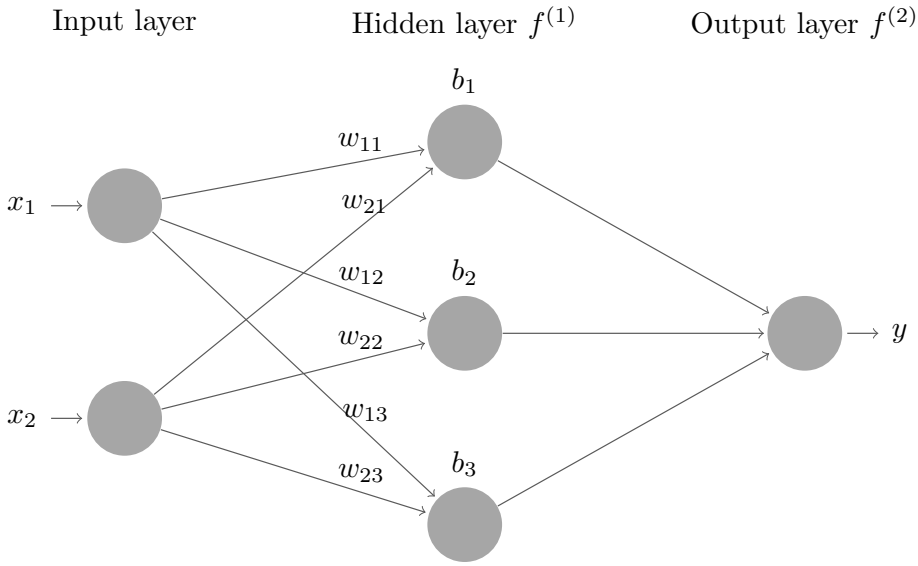

Figure 1.1: Neural network with input layer of size 2, one hidden layer of size 3 and output layer of size 1.

Then the resulting values of all incoming edges of a neuron are summed together and some differentiable *activation function* f is applied to the result. Activation function is usually *sigmoid*, *hyperbolic tangent*, *relu* or *softmax* [11]. This whole process is called *forward propagation* and it can be thought of as applying the approximation of function f to the inputs.

This approximation is, at the beginning of the training process, random as the parameters $\boldsymbol{w}$ and b are randomly initialized. Now, we would like to adjust them in order to get better performance. This is done by defining *loss* function, a performance measure described in the previous section.

Our objective is to minimize the loss function with respect to the input. Natural way to discover the direction in which to minimize the loss function is to find its derivatives with respect to each training parameter. The obtained derivative is then multiplied by *learning rate* $\alpha \in (0, 1)$ and subtracted from the corresponding parameter:

$$\theta \leftarrow \theta - \alpha \frac{\partial L}{\partial \theta},$$

where L is the loss function. The described procedure is the simplest algorithm used for this purpose: *stochastic gradient descent* [11].

The optimization algorithm we use is called Adam [13]. It is based on stochastic gradient descent algorithm but is more complicated as it takes into account

past gradients and second moments of those gradients. The advantage of Adam is that it converges faster to the desired value and it is not as prone to converging to local minima like stochastic gradient descent is.

In previous paragraphs we considered the networks to take one training instance at a time, compute gradients and apply them on parameters using optimizer. This is not always the case. Some number (called *mini-batch* or *batch*) of instances is usually propagated through the neural network, their gradients are acquired, averaged and the descent is applied only once on the parameters [11]. This is an important technique which minimizes variance of the loss function, as our estimate of direction, in which we minimize, is more confident because the gradients are computed using more training samples. Also, it is often infeasible to use all instances in the train set as a batch because it would take the model long time to converge, as the algorithm would have to compute gradients for the whole set and then perform only one training step.

1.2 Statistics

To design good quality loss function, *maximum likelihood principle* is used [11]. Let $p(\boldsymbol{x})$ be a distribution function of random vector $\boldsymbol{x}$, $p(\boldsymbol{x};\boldsymbol{\theta})$ is a distribution of $\boldsymbol{x}$ explicitly parametrized with $\boldsymbol{\theta}$. We call the set $\{p(\boldsymbol{x};\boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Theta\}$, where Θ is a set of all possible values $\boldsymbol{\theta}$ can have, a family of probability distributions. The expectation of function $f(\boldsymbol{x})$ with respect to some continuous probability distribution p is defined

$$\mathbb{E}_{\boldsymbol{x}\sim p}[f(\boldsymbol{x})] = \int p(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x}.$$

Let $p_{data}(\boldsymbol{x})$ be the probability distribution of $\boldsymbol{x}$, p_{data} is unknown data generating distribution that we wish to estimate. Let $\hat{p}_{data}$ be the empirical distribution of $\boldsymbol{x}$ defined by training samples $\mathbb{X}$.Let $p_{model}(\boldsymbol{x};\boldsymbol{\theta})$ be an estimation of $p_{data}(\boldsymbol{x})$, which is the probability of $\boldsymbol{x}$ in distribution p_{data}.

The maximum likelihood principle [11] states that given examples $\mathbb{X}$, drawn independently from data distribution $p_{data}(\boldsymbol{x})$, parameters $\boldsymbol{\theta}$ are estimated as such:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} p_{model}(\mathbb{X};\boldsymbol{\theta}) \tag{1.1}$$

$$= \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\sim\hat{p}_{data}}[\log p_{model}(\boldsymbol{x};\boldsymbol{\theta})], \tag{1.2}$$

Maximum likelihood estimation can be interpreted as minimizing the dissimilarity between two probability distributions, P and Q, measured by Kullback-Leibler divergence (KL divergence), defined:

$$D_{KL}(P \parallel Q) = \mathbb{E}_{\boldsymbol{x}\sim P}[\log P(\boldsymbol{x}) - \log Q(\boldsymbol{x})]. \tag{1.3}$$

Similar divergence is Jensen-Shannon divergence (JS divergence) which is defined:

$$D_{JS}(P,Q) = D_{KL}(P \parallel R) + D_{KL}(Q \parallel R) \tag{1.4}$$

where R is $(P+Q)/2$. Unlike KL divergence, JS divergence is symmetrical:

$$D_{JS}(P,Q) = D_{JS}(Q,P).$$

Another divergence, which we actually use in this work, is called Wasserstein distance or sometimes called Earth-Mover distance [14], see section 1.3.1:

$$W(P,Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\gamma}[|\boldsymbol{x}-\boldsymbol{y}|], \tag{1.5}$$

where $\Pi(P,Q)$ is the set of all joint distributions of $\gamma(\boldsymbol{x},\boldsymbol{y})$. The infimum in (1.5) is intractable but the equation can be rewritten using Kantorovich-Rubinstein duality [15]:

$$K \cdot W(P,Q) = \sup_{\|f\|_L \leq K} \mathbb{E}_{\boldsymbol{x}\sim P}[f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim Q}[f(\boldsymbol{x})], \tag{1.6}$$

where $K \in \mathbb{N}$ and $\|f\|_L \leq K$ means that function f is K-Lipschitz continuous. K-Lipschitz continuousness is defined

$$|f(x_1) - f(x_2)| \leq K \cdot |x_1 - x_2|\,.$$

In order to evaluate and compare trained models we need to choose suitable metric. We use two metrics: precision at different recall levels and area under precision-recall curve (AUPRC). We consider anomalies to be the positive class. Let us define precision:

$$precision = \frac{|TruePositives|}{|TruePositives| + |FalsePositives|},$$

recall as

$$recall = \frac{|TruePositives|}{|TruePositives| + |FalseNegatives|},$$

where the absolute value signifies the number of examples belonging to the set in question (true positives, false positives, ...). PR-curve describes how precision reacts to a change in recall.

Precision-recall curve is dependent on the ratio of positives and negatives, or anomalies and normals respectively. Even though some other metrics, for example receiver operating characteristic (ROC) curve, are class balance independent, they are not as good for evaluation of models trained on highly imbalanced datasets as PR curve is; since even the simplest baseline — always choosing the most common class — achieves a very high score. On the other hand, when computing the quality of such classifier under AUPRC, it completely fails. In [8] they empirically show that PR-curve is better than ROC curve on imbalanced datasets.

1.3 Generative adversarial networks

Generative adversarial networks (GANs) [16] is a machine learning method employing two elements which play min-max game. Informally said the generator is trying to generate instances as similar as possible to the real data and the discriminator is trying to decide whether given example comes from real data or has been generated. The objective function is

$$V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim\hat{p}_{data}}[\log D(\boldsymbol{x};\boldsymbol{\theta}_d)] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[1 - \log D(G(\boldsymbol{z};\boldsymbol{\theta}_g);\boldsymbol{\theta}_d))], \tag{1.7}$$

where $\hat{p}_{data}$ is the empirical distribution defined by training data and $p_{\boldsymbol{z}}$ is the distribution of latent variable $\boldsymbol{z}$ (usually normal or uniform). Parameters of generator and discriminator are $\boldsymbol{\theta}_g$ and $\boldsymbol{\theta}_d$ respectively.

Generator and discriminator then play the following game

$$\min_G \max_D V(D, G), \tag{1.8}$$

i.e., the discriminator is trying to maximize its output on real data and at the same time minimize its output on the generated data. Generator, on the other hand, tries to minimize $1 - D(G(\boldsymbol{z}))$, forcing the discriminator into producing high outputs on generated data.

Training GAN is equivalent to minimizing *Jensen-Shannon* divergence (1.4).

GAN learns *manifold* of its inputs. Manifold $\mathcal{X}$ is a topological space for which the following holds: neighbourhood of every point on $\mathcal{X}$ is homomorphic to euclidean space [11], hence every point has continuous neighbourhood. Learning distribution of the inputs usually means learning the manifold they occupy.

Figure 1.2 depicts a generative adversarial network.

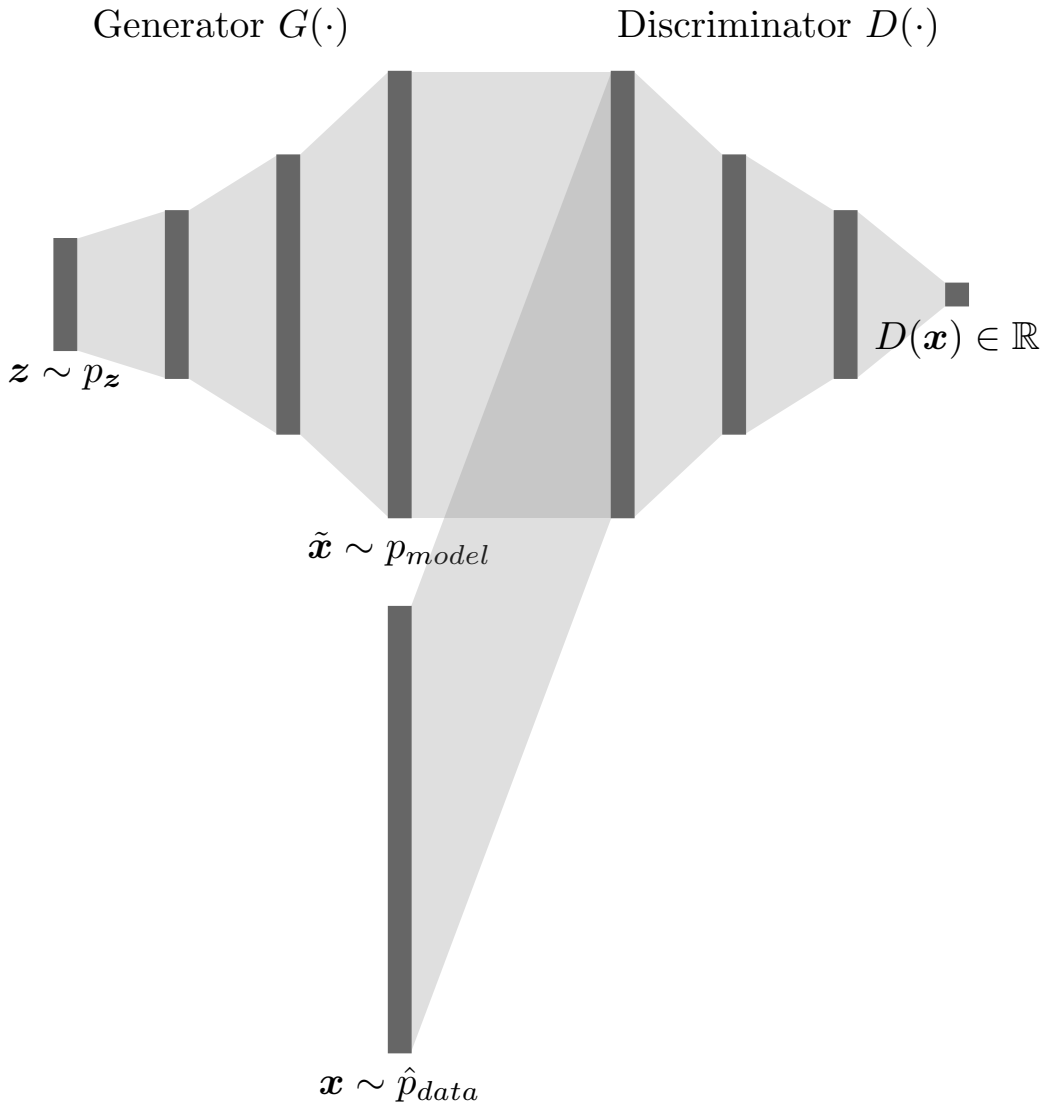

Figure 1.2: Illustration of GAN. The darker grey rectangles represent arrays of neurons and the lighter grey shapes are connections between them.

1.3.1 Wasserstein GAN

GANs suffer from many problems [17], due to their sensitivity to hyperparameters. Such as *mode collapse*, this is state of the network in which generator tends to generate very similar instances. Another problem is *vanishing gradients*, which happens when JS divergence increases and the gradients are getting closer to zero. Furthermore, it can easily happen that the loss will diverge from the optimum.

These problems are addressed in [14] where they propose a new architecture called Wasserstein GAN (WGAN) based on Earth-Mover distance (also called Wasserstein distance). Instead of using JS divergence to measure the divergence between the distribution of real instances $\hat{p}_{data}$ and the distribution of the generated ones p_{model}, Earth-Mover distance (1.6) is utilized. Also, they call the discriminator *critic*.

This approach yields better results because $W(\hat{p}_{data}, p_{model})$ is continuous function on $\boldsymbol{\theta}$ under some mild assumptions, as opposed to JS divergence [14]. $W(\hat{p}_{data}, p_{model})$ is defined as such:

$$W(\hat{p}_{data}, p_{model}) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\boldsymbol{x} \sim \hat{p}_{data}}[f(\boldsymbol{x})] - \mathbb{E}_{\tilde{\boldsymbol{x}} \sim p_{model}}[f(\boldsymbol{x})], \tag{1.9}$$

where $\|f\|_L \leq 1$ means that function f is 1-Lipschitz.

To find function f they have constructed neural network which utilizes weights clipping to some compact space $\mathcal{W}$, which ensures the K-Lipschitz property of f (K is dependent only on $\mathcal{W}$).

Alternative way to enforce functions to be K-Lipschitz is *gradient penalty* introduced in [18]. The new loss function for critic is:

$$L = \mathbb{E}_{\tilde{\boldsymbol{x}} \sim p_{model}}[D(\tilde{\boldsymbol{x}})] - \mathbb{E}_{\boldsymbol{x} \sim \hat{p}_{data}}[D(\boldsymbol{x})] + \lambda \, \mathbb{E}_{\hat{\boldsymbol{x}} \sim p_{\hat{\boldsymbol{x}}}}[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2] \tag{1.10}$$

where $p_{\hat{\boldsymbol{x}}}$ is uniform distribution of points sampled along straight lines between points from real data $\hat{p}_{data}$ and generated data p_{model}. And $\lambda \in \mathbb{N}$ is hyperparameter. The last summand of this loss is the gradient penalty, which ensures that D is 1-Lipschitz [18].

According to the authors of WGAN, it is desirable to train the critic, before training generator, till optimality. Well trained critic is still able to provide useful gradients for generator, whereas regular discriminator's gradient would vanish.

1.3.2 Adversarially Learned Inference

The generator learns a mapping from latent space to data space but there is no obvious ways to get the inverse mapping. Such function is useful for anomaly detection, where we need to know the distribution of our data points in latent space.

There are two articles about learning the mapping from data space to latent space [19], [20]. These articles are using the same procedure to learn the mapping. Additional neural network, which approximates the function data space to latent space, is added to the existing GAN architecture: *encoder*. We denote point from distribution of real samples encoded to latent space $E(\boldsymbol{x})$.

Instead of distinguishing generated instances from real ones, the discriminator tries to distinguish pair $(\boldsymbol{x}, E(\boldsymbol{x}))$ from $(G(\boldsymbol{z}), \boldsymbol{z}))$ where $\boldsymbol{x} \sim \hat{p}_{data}$ and $\boldsymbol{z} \sim p_{\boldsymbol{z}}$.

The resulting game is

$$\min_{G,E} \max_{D} V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim\hat{p}_{data}}[\log D(\boldsymbol{x}, E(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[1 - \log D(G(\boldsymbol{z}), \boldsymbol{z})], \quad (1.11)$$

where D, G, E are neural networks. The objective is to match the distributions generating pairs $(\boldsymbol{x}, E(\boldsymbol{x}))$ and $(G(\boldsymbol{z}), \boldsymbol{z}))$.

2. Related Work

In this chapter, we review various machine learning methods suitable for anomaly detection.

2.1 Anomaly detection methods

Anomaly detection is topic which has been extensively studied. Methods can be categorized in three categories [12]: supervised, unsupervised and semi supervised, see Chapter 1.1.

Supervised methods require annotated datasets to be able to detect anomalies. The need for annotated data is usually unattainable since anomalies are rare. Also, the dataset would be highly imbalanced. Unsupervised anomaly detection methods need only normal instance during training. This is quite feasible since most of the data, in which one wants to find outliers, are not anomalies. Sometimes this discipline is called *novelty* detection because it can be described as detecting novelties, which do not conform to the distribution of already experienced examples.

In the subsequent subsections, we describe papers about anomaly detection [8] and some traditional machine learning methods that can be used for unsupervised anomaly detection. We use k-Means ensemble [8], One Class Support Vector Machines [9], and Isolation Forest [7] as baselines for comparison with our model.

2.1.1 k-Means ensemble

In a recent paper by Porwal et al. [8], a novel outlier detection method is proposed, the authors evaluate it on CCFD and compare it with Isolation Forest (outperforming it) [7].

k-Means [21] is an unsupervised classification algorithm. The idea is that given $k \in \mathbb{N}$ and data $\mathbb{X}$, the algorithm tries to divide $\mathbb{X}$ into k sets (clusters) such that examples in each cluster are similar to each other. Interested reader can find out more about k-Means algorithm in book by James et al. [22].

The ensemble contains multiple k-Means models, each trained with different k. It is necessary to choose suitable values of k for the models. The authors use Silhouette score [23] for this purpose. Silhouette score of a data point is a measure of how much does the data point in question belong to a cluster it has been assigned to [8]. Silhouette score of the whole model is the average of average silhouette scores of all clusters. Final values of k, used in the ensemble have positive Silhouette score.

During prediction, each data point is assigned a vector $(c_1, \ldots, c_L)$ of clusters (from L runs of k-Means). Anomalous behaviour of a data point is scored according to weighted similarity of centroids of the clusters to which it has been assigned by the ensemble, with high values meaning consistent, normal behaviour:

$$score = \frac{\sum_{i=1}^{L-1} \sum_{j=i+1}^{L} (|c_i| + |c_j|) \cos(C_i, C_j)}{\sum_{i=1}^{L-1} \sum_{j=i+1}^{L} (|c_i| + |c_j|)}, \tag{2.1}$$

where C_i is centroid of cluster c_i, $|c_i|$ is the number of samples assigned to cluster c_i, and *cos* is cosine similarity metric.

2.1.2 One Class Support Vector Machine

One Class Support Vector Machines (OC-SVM) is an unsupervised machine learning algorithm based on Support Vector Machines (SVM) [24] which, on the other hand, is supervised. It is used for binary classification. The principle of SVM lies in finding hyperplane separating the dataset into the two classes. When the partition between individual classes is non-linear, use of some non-linear function for transformation of data points is inevitable.

However, when training OC-SVM, instead of separating data points belonging to two classes, we are trying to find hyperplane in n dimensional space such that training examples are on one side and all other points $\boldsymbol{x} \in \mathbb{R}^n$ lie on the opposite side. Anomaly score is then determined by the distance of tested data point from the hyperplane.

Major disadvantage of this algorithm is the quadratic computational complexity (in number of data points) during training. On the other hand, evaluation time is quite quick, when compared to other methods, see table 5.1.

Detailed description of Support Vector Machines (SVM) is out of the scope of this thesis. Reader may get more information on this topic in book by Cristianini [25].

2.1.3 Isolation Forest

Isolation Forest [7] is an unsupervised machine learning algorithm. It is similar to Decision Tree and its ensemble version, Random Forest [10]. Idea of Isolation *tree* is that anomalies require fewer splits in the decision tree than normal instances. Isolation Forest is then an ensemble of such trees.

Anomaly score of a sample depends on the number of edges from the root to a node, to which the sample has been assigned. It is given by

$$s(x, n) = 2^{\frac{\mathbb{E}[h(x)]}{c(n)}},$$

where $\mathbb{E}[h(x)]$ is the average number of edges which x has to traverse from the root to a terminal node during classification; $c(n)$ is the average path length of unsuccessful search in a binary search tree. The number of instances is n.

Advantage of Isolation Forest is its linear computational complexity in the number of training examples.

2.1.4 GAN based methods

First model utilizing GANs to detect anomalies is, to the best of our knowledge, *AnoGAN* proposed by Schlegl et al. [2]. They train model only on normal samples and use iterative mapping to latent space to establish whether tested sample is an anomaly, see Section 3.1.

The method AnoGAN uses for mapping to the latent space is quite inefficient, which is addressed by Zenati et al. [3]. Instead of iteratively adjusting coefficients

of the mapped point in latent space, the authors train a BiGAN (1.11). Subsequently, they use the trained encoder to find the corresponding image in latent space for each data point from real data, resulting in a much more time efficient solution.

Li [4] uses GANs to detect anomalies in multivariate time series. They utilize an *LSTM* [26] recurrent neural network as a model, enabling the architecture to capture dependencies in time.

3. Our anomaly detection model

In this chapter we propose *AnoWGAN*, a model for detecting anomalies and novelties. AnoWGAN uses Improved Wasserstein GAN [18] to learn the distribution of normal examples. In order to find the latent representation of a sample, we train an encoder from data space to latent space.

Our work is based on *AnoGAN* by Schlegl et al. [2], which uses regular GANs [16] to learn the normal data manifold, and instead of encoder, AnoGAN iteratively maps each individual sample to latent space to test whether it is an anomaly.

We have tried to implement method using Adversarially Learned Inference [19] for anomaly detection, proposed by Zenati et al. [3]. It turned out that training Adversarially Learned Inference version of GANs is very difficult, see Section 4.4. We have not been able to stabilize the training on the Credit Card Fraud Dataset.

ALI trains encoder, generator and discriminator concurrently (1.11). Instead, we train the encoder after training of generator and critic.

Section 3.1 describes *AnoGAN*, method developed by Schlegl et al. [2]. In Section 3.2 we propose *AnoWGAN*, a model for detecting anomalies. There are two versions – with and without encoder. The first, which we denote AnoWGAN+e, utilizes encoder to latent space and the other, denoted AnoWGAN–e, relies on iterative mapping. Otherwise, the models are the same. When we describe some feature that they have in common, we denote the model as AnoWGAN.

3.1 AnoGAN

Schlegl et al. [2] introduced new architecture using generative adversarial networks. The principle lies in learning the manifold, on which normal instances (images of eye retina in this article) lie. Since anomalies do not lie on this manifold, it is possible to distinguish them. GANs are mapping points from latent space to data space but not the other way around. To solve this issue, the authors propose a technique for mapping from data space to latent space. Given a data point $\boldsymbol{x}$, coordinates of a random point $\boldsymbol{z}$ from latent space are iteratively adjusted to minimize the dissimilarity of $\boldsymbol{x}$ and $G(\boldsymbol{z})$. They denote $\boldsymbol{z}$ after γ iterations as $\boldsymbol{z}_\gamma$.

Method called *feature matching*, first proposed by Salimans et al. [27], is utilized to enforce the mapped point to lie on the learned manifold. So the *discriminator loss* L_D used for mapping to latent space is following:

$$L_D(\boldsymbol{z}_\gamma) = \sum |\mathbf{f}(\boldsymbol{x}) - \mathbf{f}(G(\boldsymbol{z}_\gamma))| \tag{3.1}$$

where $\mathbf{f}(\cdot)$ is the output of an intermediate layer of the discriminator. An intermediate layer is the last but one layer in the computational graph of a neural network. Discriminator loss enforces the mapped instance $\boldsymbol{x}$ to lie on the learned manifold. This loss is combined with *residual loss* which measures the similarity between generated instances and real data:

$$L_R(\boldsymbol{z}_\gamma) = \sum |\boldsymbol{x} - G(\boldsymbol{z}_\gamma)| \,. \tag{3.2}$$

Final loss is computed as a convex combination of residual and discriminator loss:

$$L(\boldsymbol{z}_\gamma) = (1-\zeta)\cdot L_R(\boldsymbol{z}_\gamma) + \zeta \cdot L_D(\boldsymbol{z}_\gamma) \tag{3.3}$$

where $\zeta \in [0,1]$ is hyperparameter.

3.2 AnoWGAN

Our model uses Improved WGAN to learn the manifold of normal samples. Generator changes its parameters so that it maximizes critic's output on the generated instance:

$$L_G = \min -\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[D(G(\boldsymbol{z}))]. \tag{3.4}$$

Critic on the other hand changes its parameters to minimize its output on generated instance and at the same time maximize the output on real instance. Consequently, because we are training WGAN with gradient penalty, we need to add the penalty to the critic's loss:

$$L_C = \min_D \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[D(G(\boldsymbol{z}))] - \mathbb{E}_{\boldsymbol{x}\sim \hat{p}_{data}}[D(\boldsymbol{x})] + \lambda\, \mathbb{E}_{\hat{\boldsymbol{x}}\sim p_{\hat{\boldsymbol{x}}}}[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2], \tag{3.5}$$

which is equation (1.10), where $\hat{\boldsymbol{x}} = \epsilon\cdot\boldsymbol{x} + (1-\epsilon)\cdot G(\boldsymbol{z})$, $\epsilon \sim U(0,1)$ and $\lambda \geq 0$.

To detect anomalies of one data point $\boldsymbol{x}$, we need to find its image $\boldsymbol{z}$ in latent space. We have two versions of our model to find the image in latent space. The first is similar to AnoGAN, it utilizes iterative mapping, minimizing objective (3.3) with $\zeta = 0$. This procedure is time consuming, see table 5.1.

AnoWGAN+e makes use of another neural network, encoder. The difference of our model and the one proposed by Zenati et al. [3] is that we do not train the encoder concurrently with generator and critic but after. Encoder is trained by minimizing the absolute difference between instance $\boldsymbol{x}$ real data and its projection back to the space of real data but through encoder and already trained generator:

$$L_E = \min_E \mathbb{E}_{\boldsymbol{x}\sim \hat{p}_{data}}[|\boldsymbol{x} - G(E(\boldsymbol{x}))|]. \tag{3.6}$$

The choice to train the encoder independently of generator and critic leads to much better stability of training, see Section 4.4 for instability of training of ALI.

4. Experiments

The focus of this chapter is the practical use of our model (AnoWGAN) to detect fraudulent credit card transactions. In the first section, we explore the Credit Card Fraud Dataset. Section 4.2 describes the neural network, used as our model, and the training process. Section 4.3 describes the implementation of models compared in the next chapter (5) and concrete hyperparameters necessary to reproduce our results. Section 4.4 is about failed attempt to train the ALI model.

4.1 Dataset

There are not many publicly available datasets suitable for anomaly detection. Most publicly available datasets, such as *Arrythmia* or *Thyroid* [28], do not have a sufficient number of instances for GAN to be able to train itself on them.

Dataset[1] which is used in this thesis is Credit Card Fraud Dataset (CCFD) made available by Pozzolo et al. [6], [29]. There are 284 315 normal examples and 492 anomalous ones. This is suitable for the method of our choosing since we only need normal instances during training.

The CCFD is anonymized, using Principle Component Analysis (PCA) [30], so that it could be published. Every data point in CCFD consists of 30 real numbers, e.g. amount; time measured from the first transaction. For the purpose of our task, we omit the first and last feature (time and amount).

#	0	1	...	28	29
0	36462	0.774650	...	0.010446	118.80
1	172213	2.113557	...	-0.038202	1.29
2	127823	1.933172	...	-0.037490	70.00
3	158628	1.800412	...	-0.023028	150.00
4	73543	-0.620267	...	-0.342146	4.52

Table 4.1: Sample of the dataset. Each line is one instance (transaction) processed by PCA. First column is time and the last one is amount.

In Figure 4.1 is the correlation matrix of the remaining features. We randomly sampled 2000 normal instances and 50 anomalous to perform hyperparameter search. The rest of the data is used for training and testing.

[1] Credit Card Fraud Dataset is available at `https://www.kaggle.com/mlg-ulb/creditcardfraud` (visited: 04/07/2019)

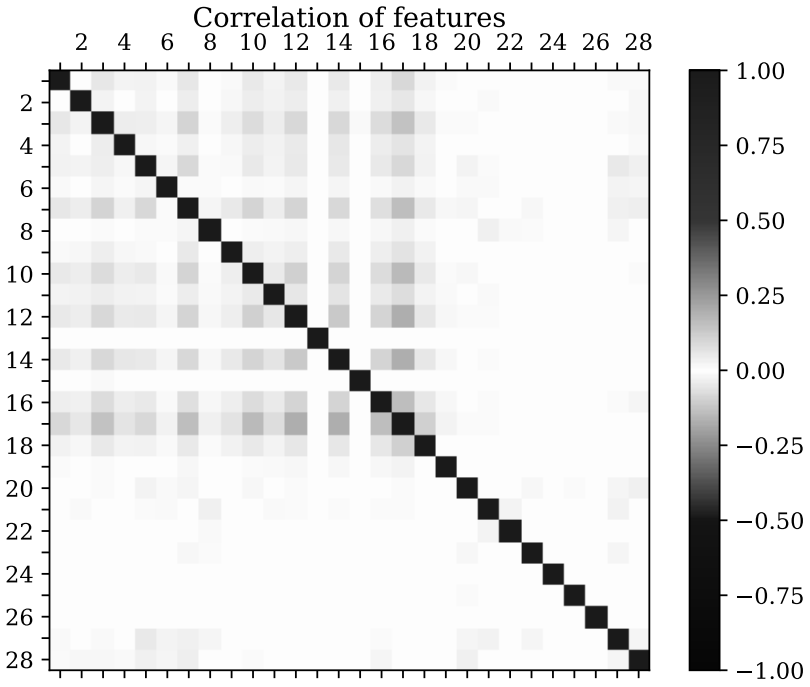

Figure 4.1: Correlation of used features.

4.2 AnoWGAN

In this section, we discuss the model used to detect anomalies in the Credit Card Fraud Dataset. We developed two versions of the model – AnoWGAN–e and AnoWGAN+e. Both of them use Improved WGAN [18] to learn the distribution of normal instances. They differ in mapping to latent space during evaluation – the first model uses iterative mapping and the other trained encoder, see Section 3.2.

4.2.1 Learning normal data distribution

We use a fully connected neural network in both generator and critic. The overall architecture of AnoWGAN used to learn normal data manifold is depicted in Figure 4.2. The generator is slightly *weaker*, consisting of a smaller number of layers, than the critic as we do need to have a very solid critic, one that is able to "criticize" the generated instances as good as possible.

After all hidden layers, there is *Leaky ReLU* activation function. ReLU is abbrevation for *rectified linear unit*, defined as

$$ReLU(x) = max(0, x).$$

Leaky ReLU is parametrized function similar to ReLU with the difference that it allows a small gradient for input values smaller than 0.

$$LeakyReLU(x; \alpha) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{otherwise} \end{cases} \tag{4.1}$$

where $\alpha \in (0, 1)$.

After most of the layers (and their activations), there is a dropout [31]. Dropout ensures that each parameter of the layer is excluded from training (at given time step) with some desired probability and therefore forcing the network to train all the connections between layers. In our case, the probability is 0.2 for the generator layers and 0.3 for critic layers.

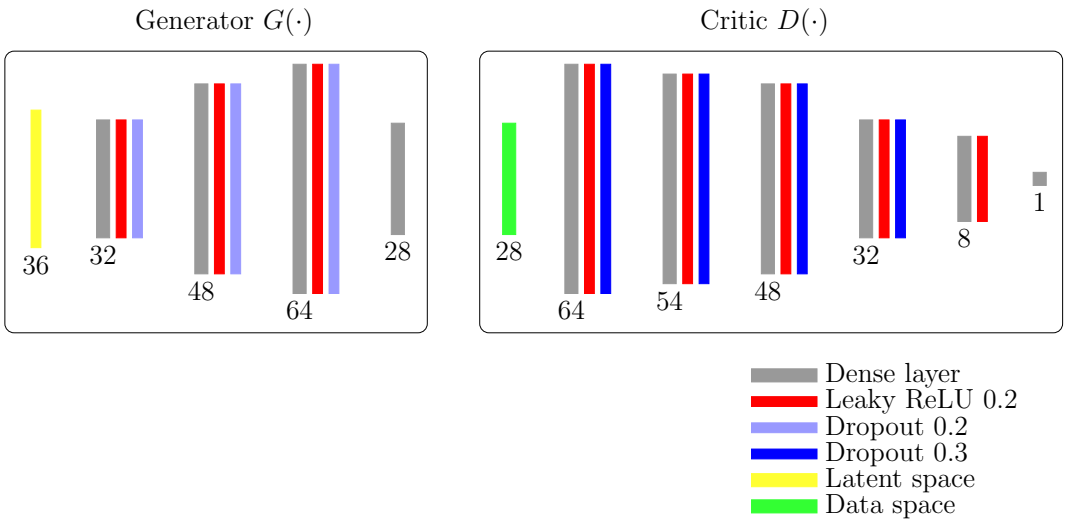

Figure 4.2: Diagram of our model. The numbers under dense layers denote their output sizes. The numbers under latent and data space denote the dimensionality.

Latent space dimension is 36 when the encoder is utilized, and 12 when we use only mapping. Instead of using uniform random distribution for sampling as in [18], we use normal distribution for it helps the model to converge.

The training procedure for learning dataset distribution can be seen in Algorithm 4.1.

4.2.2 Detecting anomalies

In the previous subsection, we described a model which learns the distribution of normal instances. Now we would like to use that distribution to test whether a data point is an anomaly or not. We use two methods to do that.

First is iterative *mapping* to latent space which was proposed by Schlegl et al. [2], see Section 3.1. The second approach we chose, is to train another neural network which learns the mapping from data space p_{data} to latent space p_z, see Section 3.2

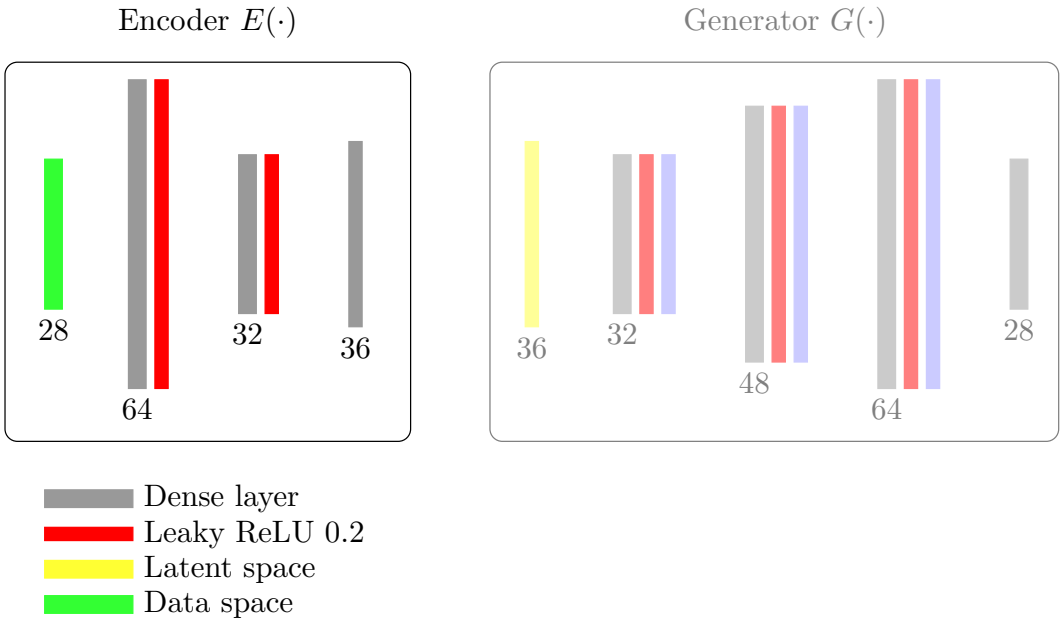

Figure 4.3: Diagram of the encoder. The rest of the network (generator and critic) is the same as in Figure 4.2

The architecture of encoder used to detect anomalies in CCFD and its place within the whole network is depicted in Figure 4.3.

4.2.3 Training and hyperparameters

In this subsection, we describe the training process and hyperparameters of both the model used to learn the distribution of the data and the encoder to latent space.

Generator and critic

At the start of the training, we pre-train the critic using 2000 update steps. Unlike training regular GANs, where it is desirable to balance training of generator and discriminator, in WGANs on the other hand we want the critic to be well trained at every stage. We find this necessary because even in the beginning of the training the generator needs to have a good "feedback" from the critic on the samples it generates, and Wasserstein distance makes it possible for the critic to provide usable gradients even if it is much "better" than the generator.

The initial learning rate of the generator is set to $3 \cdot 10^{-5}$ and critic starts with 10^{-4}.

We train our model for 5 epochs on batches of size 4. At the start of each epoch, we perform 1000 update steps of the critic and 30 update steps of the generator. After every other epoch, we divide the learning rate of the generator and the critic by 1.65 and 1.25 respectively.

For every update of the generator, we update the critic seven times. This is necessary; otherwise, if we would do one update of critic to one update of generator, the generator would become too good with respect to the critic. And

Algorithm 4.1 Pseudocode of training the model to learn the representation of normal instances.

```
# c_lr is critic learning rate
# g_lr is generator learning rate

train critic 2000 times
for e in epochs:
    # lower learning rate every two epochs
    if e > 0 and e is even:
        c_lr /= 1.25
        g_lr /= 1.65

    # pretrain critic and generator
    train critic 1000 times
    train generator 30 times

    while e has not ended:
        # critic iterations increase with epoch
        for c in 1...(7 + 2*e):
            take batch of size 4
            train critic

        train generator
```

Algorithm 4.2 Pseudocode of encoder training.

```
# e_lr is encoder learning rate
restore generator

for e in encoder_epochs:
    # lower learning rate every four epochs
    if e > 0 and e modulo 4 == 0:
        e_lr /= 2

    while e has not ended:
        take batch of size 32
        train encoder
```

it would be easy for the generator to generate samples which the critic would not be able to distinguish from the real samples; however, the generated instances would not be of a good quality (as the critic itself would be weak). Every epoch, the number of critic iterations is increased by two. The training procedure is displayed in Algorithm 4.1.

Encoder

Keeping the parameters of the generator fixed, we train the encoder for 6 epochs with mini-batches of size 32. The critic is not used at all during encoder training. The initial learning rate is set to 10^{-5} and is also decayed, but this time every four epochs by factor of 0.5.

We use t-Distributed Stochastic Neighbourhood Embedding (t-SNE) [32] to visualize the learned manifold 4.4. Algorithm t-SNE is mostly used for dimensionality reduction with the intention to visualize the data in two or three-dimensional space.

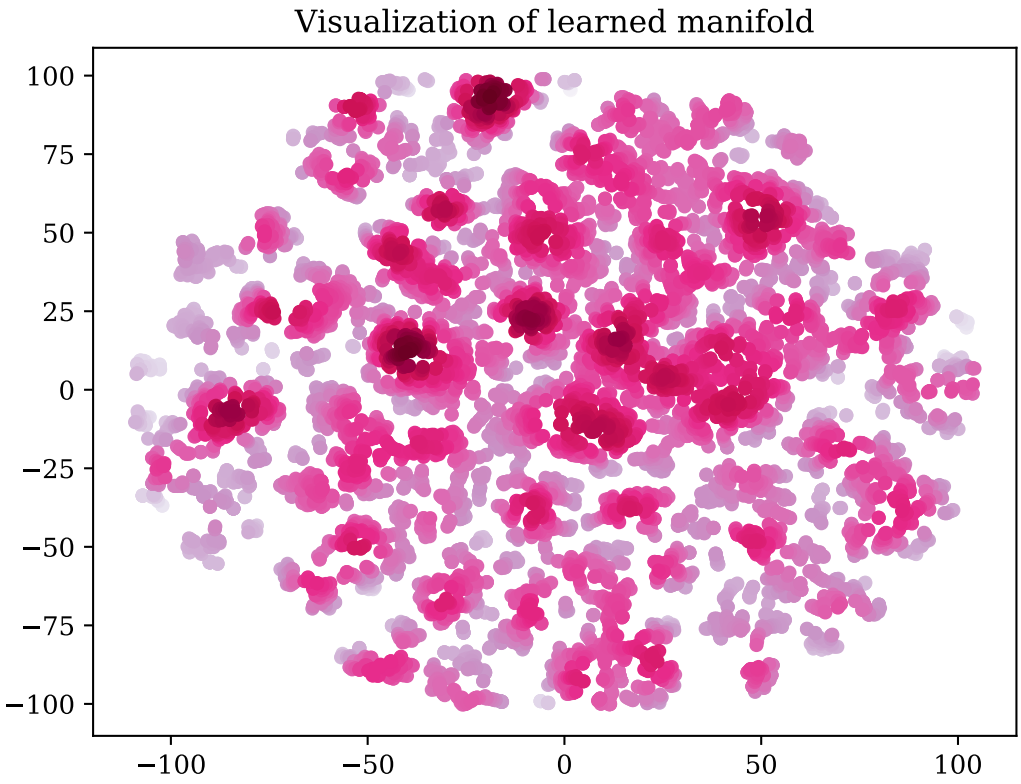

Figure 4.4: Visualization of learned manifold using t-SNE. Twenty thousand *training* samples are mapped to the latent space through the encoder to make this image. There is no reasonable interpretation of axes.

4.3 Implementation

We used a computer with Intel® Core™ i7-6700 CPU and Gentoo[2] operating system to run the experiments.

The model is implemented in Tensorflow 1.12 [33]. Dataset, which is available in comma separated value format, is converted to `.npy` format, which is used by Python library NumPy 1.15.4 [34] for storing numerical arrays. We ran the model in Python 3.6.5. Methods, to which we compare our model (One Class Support Vector Machines and Isolation Forest), are already implemented in Scikit-learn 0.20.2 [35] library for Python. The implemented model is an attachment to this thesis, see Section A.1.

In the next subsections, we describe all hyperparameters used during training of baselines and our model.

4.3.1 Hyperparameters of baselines

We use OC-SVM, k-Means ensemble and Isolation Forest to compare our model with. The results of k-Means ensemble are taken from the original paper [8]. We have implemented the model but the results we have got were inferior to those in the paper (training is sensitive to the order in which the data points are

[2] `https://www.gentoo.org` (visited 04/28/2019)

Algorithm 4.3 Constructor for OC-SVM class in Scikit-learn.

```
OneClassSVM(coef0=0.0, degree=3, \
            gamma=0.007, kernel='rbf', \
            max_iter=-1, nu=0.0005, \
            random_state=41, shrinking=True, \
            tol=0.001)
```

Algorithm 4.4 Constructor for Isolation Forest class in Scikit-learn.

```
IsolationForest(n_estimators=100, max_samples='auto', \
                contamination=0.01, max_features=1.0, \
                bootstrap=False, n_jobs=-1, \
                random_state=41, verbose=0, \
                behaviour="new")
```

presented [8]), so we decided not to use our implementation and state the results from the paper.

OC-SVM has hyperparametes used in kernel[3] on Kaggle website. Hyperparameters of Isolation Forest are also from a kernel[4] on Kaggle website, from which the CCFD comes. As the model is implemented in Scikit-learn 0.20.2 [35] we provide the hyperparameters as a constructor of OC-SVM class in Scikit-learn, see algorithms 4.3, 4.4.

4.3.2 Hyperparameters of AnoWGAN

Hyperparameters used for training and evaluation of our model on the Credit Card Fraud Dataset dataset are presented in Table 4.2. Furthermore, training uses the following hyperparameters.

- After every other epoch, we divide the learning rate of the generator and the critic by 1.65 and 1.25 respectively.
- The encoder learning rate is divided by 2 every four epochs.
- At the start of the training, the critic is pre-trained on 2000 training steps.
- Starting from the second epoch, the critic and the generator are pre-trained on 1000 and 30 training steps respectively.
- The number of critic iterations is increased by 2 every epoch.
- At the end of the last epoch, during normal data distribution learning, the generator is trained for 10 more steps.

[3] `https://www.kaggle.com/neoyipeng2018/one-class-svm-and-data-leakage` (visited: 04/08/2019)

[4] `https://www.kaggle.com/rgaddati/unsupervised-fraud-detection-isolation-forest` (visited: 04/15/2019)

Hyperparameter	value
Batch size	4
Batch size – encoder	32
Initial learning rate G	$3.0 \cdot 10^{-5}$
Initial learning rate D	10^{-4}
Initial learning rate E	10^{-5}
Initial critic iterations	7
Epochs	5
Epochs – encoder	6
Penalty coefficient λ	12
Mapping coefficient ζ	0.0
Dropout – generator	0.2
Dropout – critic	0.3
Latent space dimensionality – encoder	36
Latent space dimensionality – mapping	12
Mapping iterations	70
Mapping learning rate	0.09
Adam first momentum β_1	0.1
Adam second momentum β_2	0.9

Table 4.2: Hyperparameters used during training and evaluation.

When splitting dataset to train, development and test parts, we use random seed 42 (for NumPy). Tensorflow uses random seed 88, so does NumPy when sampling the dataset. When evaluating we set NumPy random seed to 42.

4.4 Efficient AnoGAN

We have tried to implement anomaly detection technique from [3]. However, BiGANs showed themselves to be very unstable during training and we have not been able to stabilize it on the CCFD.

In Figures 4.5 and 4.6 we show loss functions for different sets of parameters. These three models differ in the number of the discriminator iterations (how many training steps discriminator takes per one generator step) and batch sizes. All models have learning rates of both the discriminator and the generator set to 10^{-5}, which was empirically chosen. The models were trained for 10 epochs.

Though higher batch size leads to lower variance of the loss, it is obvious the loss functions do not show signes of convergence; for the generator tries to minimize the same loss function the disciminator maximizes. It is natural that at the beginning of the training losses of generator and discriminator are symmetrical. However, unlike what we observe in the Figures 4.5 and 4.6, we would expect the loss functions not to stay symmetrical and reach *Nash* equilibrium [36]. It is however not proven that it will converge, moreover it has been proven that in some cases (on some types of data) GANs do not converge [36].

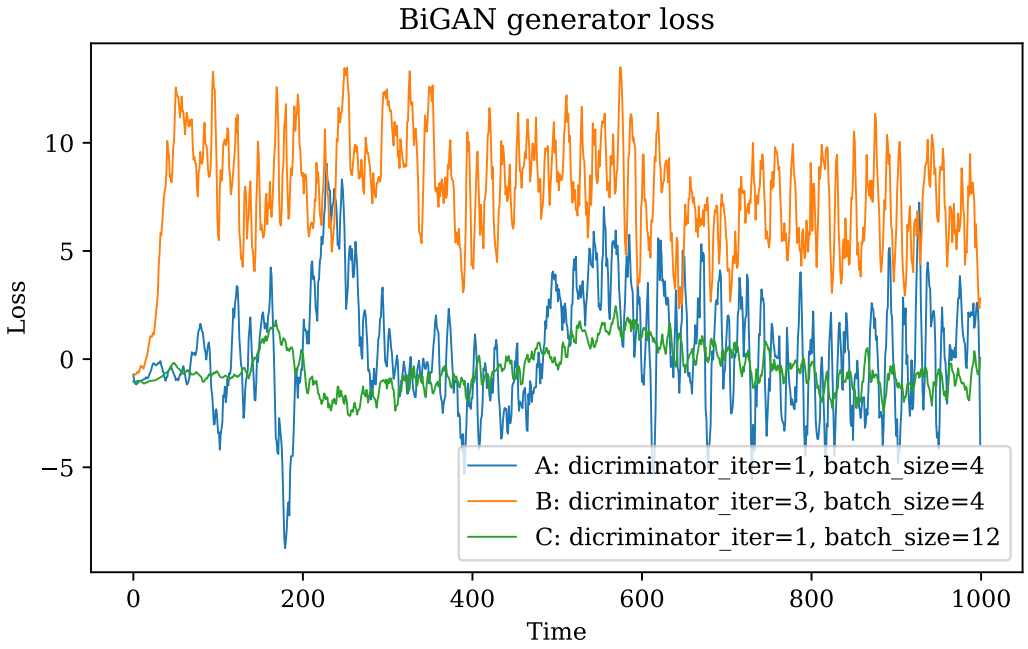

Figure 4.5: BiGAN generator loss.

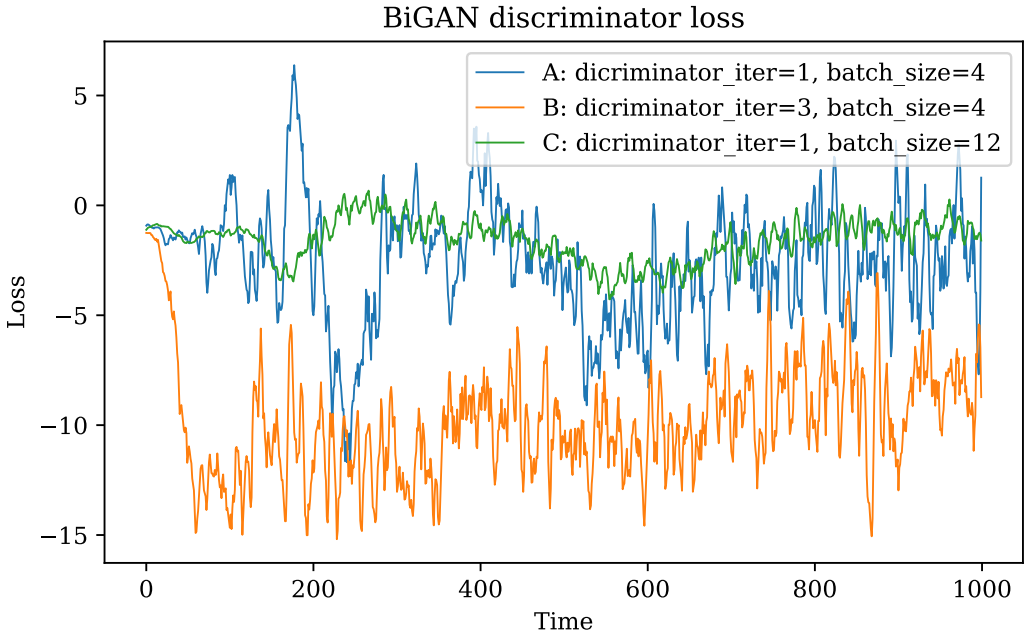

Figure 4.6: BiGAN discriminator loss.

5. Evaluation

This chapter presents results of performance measurements of AnoWGAN whose architecture is described in Section 4.2.

In the first section of this chapter, we compare our model, both with encoder and without it, to k-Means ensemble [8], OC-SVM algorithm [24], and Isolation Forest [7]. In the second section, we analyze received results.

AnoWGAN+e has the highest average AUPRC of all compared methods, see Table 5.2 and Figure 5.1. Table 5.1 shows the difference in performance, measured by AUPRC and evaluation time of studied methods. Table 5.2 compares precision of tested methods at different recall levels.

5.1 Results

We use area under precision-recall curve (AUPRC) and precision at various recall levels as evaluation metrics. It is important to keep the ratio of normal and anomalous instances almost the same during evaluation since PR curve depends on it.

Efficient AnoGAN (in Table 5.1) is the model proposed by Zenati [3], which we have not been able to stabilize, see Section 4.4. We use it as a GAN baseline.

Method	Average AUPRC	Variance	Evaluation time (s)
AnoWGAN+e	**0.4625**	0.00378	9.96
One Class SVM	0.4113	0.00475	**1.03**
AnoWGAN–e	0.2706	0.00320	433.10
k-Means ensemble[1]	0.2231	0.00144	—
Isolation Forest	0.1827	0.00261	2.27
Efficient AnoGAN	0.1196	0.00073	10.02

Table 5.1: Evaluation of compared methods.

The dataset, with which we work, is highly imbalanced — approximately 577 normal examples to 1 anomalous. We have 9735 normal examples and 442 anomalous ones for testing. To keep the ratio of normal and anomalous instances as close as possible to the original one, we make predictions on all the normal instances and 17 anomalous, arriving at ratio 572 to 1. This is repeated 26 times for different anomalous instances and the area under PR curves is averaged.

We run 29-fold cross validation. Each time the model is trained on a different subset of data and evaluated on a test set (disjoint from corresponding train set). Sets of normal data of the test sets (from the 29 runs) are disjoint. Resulting AUPRC is the average of AUPRCs from these runs.

Evaluation time has been measured on a sample consisting of 10117 examples. The measurement is conducted on a personal computer with CPU Intel® Core™

[1]Results for this method are taken from the paper [8] proposing it. The variance is from 10 runs, each with different ordering of dataset.

i7-6700. We have run five such evaluations and averaged the measured times, the variance of these runs is negligible: it is lower than one percent.

To illustrate the difference between studied techniques, we average precision from the 29 runs at each recall level, making precision-recall curve. Linear interpolation is used to smooth the curve 5.1.

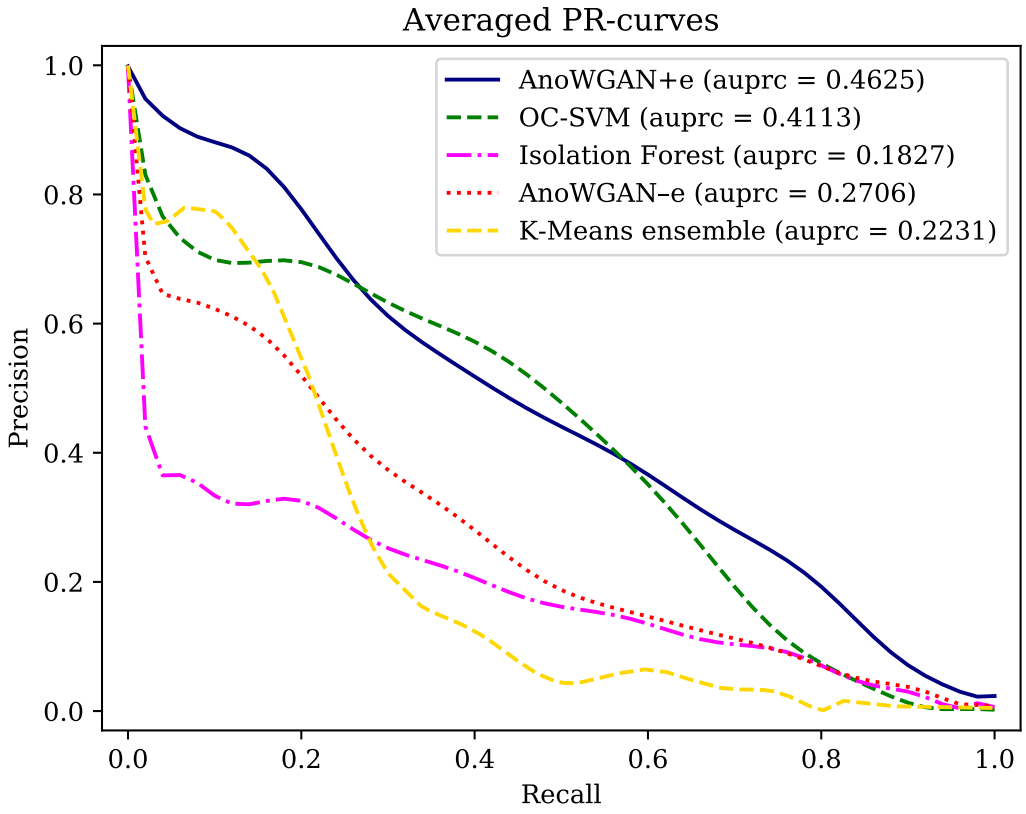

Figure 5.1: Graph of averaged precision-recall curves.

Our method (AnoWGAN+e) achieved the highest precision of all methods at recall levels 0 – 0.2 and 0.6 – 0.9, see table 5.2. Although k-Means ensemble has low AUPRC, its precision is better than OC-SVM's at lower recall levels but it has been outperformed by AnoWGAN+e in every metric, see table 5.2 and figure 5.1.

Method \ Recall	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
AnoWGAN+e	**0.89**	**0.78**	0.59	0.53	0.45	**0.35**	**0.30**	**0.20**	**0.05**
OC-SVM	0.70	0.69	**0.62**	**0.59**	**0.48**	0.32	0.23	0.08	0.01
Isolation Forest	0.34	0.32	0.24	0.22	0.17	0.13	0.11	0.07	0.02
AnoWGAN–e	0.62	0.51	0.34	0.31	0.20	0.14	0.12	0.07	0.03
k-Means ensemble	0.79	0.61	0.21	0.10	0.05	0.05	0.04	0.01	0.00

Table 5.2: Precision at different recall levels.

In Figure 5.2, there is a visualization of normal and anomalous data points mapped into latent space.

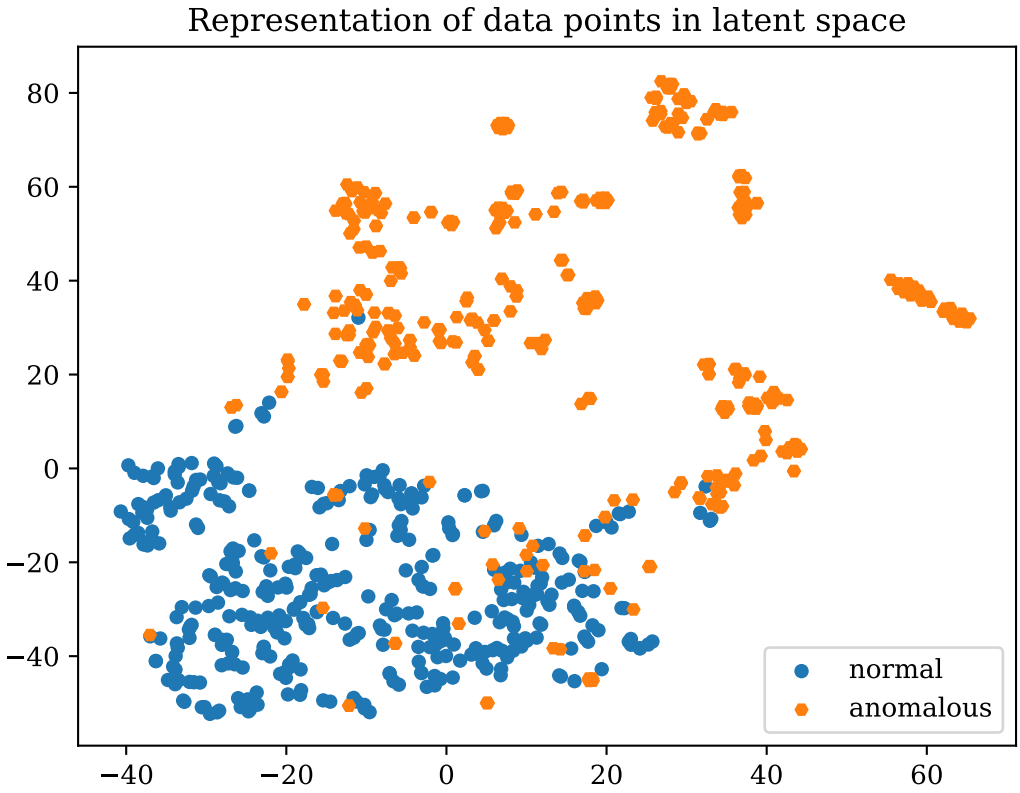

Figure 5.2: Visualization of mapped points (using encoder) to latent space. THe dimensionality of the latent space is reduced using t-SNE. There is no reasonable interpretation of axes.

5.2 Discussion

The model we propose, AnoWGAN+e, has the highest average AUPRC from all compared methods, surpassing the state-of-the-art OC-SVM (to the best of our knowledge). It also outperforms other models in precision at some recall levels, especially from 10% to 20% and from 70% to 80%. Both of these intervals can be very useful for real world applications; for example banks may want to detect a portion of credit card frauds with high level of certainty, where the interval from 10% to 20% comes into use (at 10% recall almost 90% of samples marked as anomalies actually are anomalies).

A model with high recall (but lower precision) also has an important place in fraud detection. Consider 80% recall, then our model detects anomalies with 20% precision. When we take into account that there are 492 anomalies in the dataset, then this means that we can detect 80% of them by manually examining less than 2000 transactions.

Porwal et al. [8] claim they can "identify 40% of fraud cases with high precision" (0.1 precision). However, we outperform their method by identifying 80% of fraud cases even with double precision, see table 5.2.

AnoWGAN–e did not perform very well. We tried running the evaluation with twice the number of iterations but the improvement of AUPRC was insignificant, considering doubling the evaluation time. Its evaluation time is, as expected,

slower than the times of the rest of the algorithms.

We believe that the difference in AUPRC of AnoWGAN with and without encoder is caused by the fact that encoder learns the mapping of normal examples very well and does not experience anomalous ones during training, resulting in a bigger gap between the normal and the anomalous. Whereas mapping each individual sample relies only on the generator. In other words, the encoder compensates for imperfect generator. We can see that **utilizing encoder to latent space is essential for achieving good results**.

We feel obliged to say that the time measurements are probably influenced by the Python frameworks used by the compared methods – Scikit-learn [35] and Tensorflow [33]. Some of the models might run faster on GPU that CPU, but we wanted the measurements to be comparable.

Conclusion

This thesis focuses on anomaly detection, utilizing Generative adversarial networks. Our goal was to study GAN-based anomaly detection methods and see if they can detect fraudulent credit card transactions in Credit Card Fraud Dataset.

We proposed a model employing Wasserstein GANs for learning the distribution of normal examples, and an encoder (in the form of deep neural network) for mapping from data space to latent space. Our model shows better stability than the one proposed by Zenati et al. [3].

AnoWGAN+e achieved the highest average AUPRC and exceeded all other models in precision at most of the recall levels, making it the new state-of-the-art (instead of OC-SVM) on CCFD. Our model has almost 90% precision at 10% recall, which is 10% more than the next best method (k-Means ensemble).

At 80% recall, our model detects anomalies with 20% precision (more than two times more than the next best method). There are 492 anomalies in the dataset, which means **we can detect** 80% **of frauds by manually examining less than** 2000 **transactions**.

As expected, evaluation time of AnoWGAN–e is quite long and therefore it shows the importance of utilizing encoder to latent space. The encoder also significantly improved the model's ability to detect anomalies.

In 2019, new articles about GANs have been emerging ([37]–[39]) but we did not manage to study them nor use them for anomaly detection.

The training procedure of WGANs and hyperparameter selection has been exhaustive and the model still showed instability. In future work, we believe it is crucial to improve the stability of training of GANs and make them more robust to hyperparameter change. It would also be interesting to know how do the studied methods compare on lower (or higher) dimensional datasets and how many samples are necessary to obtain a usable GAN-based model.

Bibliography

[1] K. G. Mehrotra, C. K. Mohan, and H. Huang, *Anomaly Detection Principles and Algorithms*. Springer, 2017, ISBN: 978-3-319-67526-8.

[2] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," in *Information Processing in Medical Imaging*, Springer, 2017, pp. 146–157, ISBN: 978-3-319-59050-9.

[3] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, *Efficient GAN-based anomaly detection*, 2018. arXiv: 1802.06222v1.

[4] D. Li, D. Chen, J. Goh, and S.-k. Ng, *Anomaly Detection with Generative Adversarial Networks for Multivariate Time Series*, 2018. arXiv: 1809.04758.

[5] R. Chalapathy, A. K. Menon, and S. Chawla, "Anomaly Detection using One-Class Neural Networks," *CoRR*, vol. abs/1802.06360, 2018.

[6] A. D. Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi, "Calibrating Probability with Undersampling for Unbalanced Classification," in *2015 IEEE Symposium Series on Computational Intelligence*, IEEE, 2015.

[7] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," IEEE, 2008. DOI: 10.1109/ICDM.2008.17.

[8] U. Porwal and S. Mukund, "Credit card fraud detection in e-commerce: An outlier detection approach," 2018. arXiv: 1811.02196.

[9] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Computation*, 2001. DOI: 10.1162/089976601750264965.

[10] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997, ISBN: 978-0-07-042807-2.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[12] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, 2009. DOI: 10.1145/1541880.1541882.

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference for Learning Representations, 2015*, 2014.

[14] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning*, 2017, pp. 214–223.

[15] C. Villani, *Optimal transport: old and new*, ser. Grundlehren der mathematischen Wissenschaften. Springer, 2009, ISBN: 978-3-540-71049-3.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, Curran Associates, Inc., 2014, pp. 2672–2680.

[17] M. Arjovsky and L. Bottou, "Towards Principled Methods for Training Generative Adversarial Networks," in *5th International Conference on Learning Representations, 2017*, 2017.

[18] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., 2017, pp. 5767–5777.

[19] V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. C. Courville, "Adversarially learned inference," in *5th International Conference on Learning Representations, 2017*, 2017.

[20] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial feature learning," in *5th International Conference on Learning Representations, 2017*, 2016.

[21] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, 1982. DOI: 10.1109/TIT.1982.1056489.

[22] *An introduction to statistical learning: with applications in R*, Springer, 2013, ISBN: 978-1-4614-7137-0.

[23] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, 1987. DOI: 10.1016/0377-0427(87)90125-7.

[24] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, 1995. DOI: 10.1007/BF00994018.

[25] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000. DOI: 10.1017/CBO9780511801389.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, 1997. DOI: 10.1162/neco.1997.9.8.1735.

[27] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems 29*, 2016, pp. 2234–2242.

[28] D. Dheeru and E. Karra Taniskidou, *UCI machine learning repository*, 2017. [Online]. Available: http://archive.ics.uci.edu/ml (visited on 01/02/2019).

[29] A. Dal Pozzolo, "Adaptive machine learning for credit card fraud detection," PhD thesis, Université Libre de Bruxelles, 2015.

[30] I. T. Jolliffe, *Principal Component Analysis*. Springer New York, 1986, ISBN: 978-1-4757-1904-8.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[32] L. Maaten van der and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[33] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. [Online]. Available: `https://www.tensorflow.org/`.

[34] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array," *Computing in Science & Engineering*, 2011. DOI: `10.1109/MCSE.2011.37`.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[36] L. Mescheder, A. Geiger, and S. Nowozin, "Which training methods for GANs do actually converge?" In *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, 2018, pp. 3481–3490.

[37] H. He, H. Wang, G.-H. Lee, and Y. Tian, "Bayesian modelling and monte carlo inference for GAN," in *International Conference on Learning Representations*, 2019.

[38] Y. Yazıcı, C.-S. Foo, S. Winkler, K.-H. Yap, G. Piliouras, and V. Chandrasekhar, "The unusual effectiveness of averaging in GAN training," in *International Conference on Learning Representations*, 2019.

[39] P. Mertikopoulos, B. Lecouat, H. Zenati, C.-S. Foo, V. Chandrasekhar, and G. Piliouras, "Optimistic mirror descent in saddle-point problems: Going the extra(-gradient) mile," in *International Conference on Learning Representations*, 2019.

List of Figures

List of Abbreviations

ALI Adversarially Learned Inference. 14–16

AUPRC Area under precision-recall curve. 3, 7, 26–30

BiGAN Bidirectional GAN. 13

CCFD Credit Card Fraud Dataset. 2, 3, 11, 14, 16, 17, 19, 23, 24, 30

CPU Central Processing Unit. 22, 26

GAN Generative adversarial network. 2–4, 7–9, 12–14, 16, 19, 24, 30, 34

IF Isolation Forest. 2, 3, 11, 12, 22, 23, 26, 27

JS divergence Jensen-Shannon divergence. 6, 9

KL divergence Kullback-Leibler divergence. 6

OC-SVM One Class Support Vector Machines. 2, 3, 11, 12, 22, 23, 26–28, 30

PCA Principle Component Analysis. 16

ROC receiver operating characteristic. 7

SVM Support Vector Machines. 12

t-SNE t-Distributed Stochastic Neighbourhood Embedding. 21, 28, 34

WGAN Wasserstein GAN. 9, 14, 15, 17, 19, 30

A. Attachments

A.1 First Attachment – AnoWGAN+e scripts

The ZIP file contains directory with scripts which can be used to reproduce results of AnoWGAN+e. The folder structure is following:

```
anowgane
├── README ..... Manual on how to run the scripts
├── Makefile ..... Makefile to run the scripts
├── ad_wgan.py ..... Contains neural network and training procedure
├── convert.py ..... Used for converting dataset
├── convert.sh ..... Used for converting dataset
├── convert2.py ..... Used for converting dataset
├── credit_card_dataset.py ..... Dataset handler
├── requirements.txt ..... Python3 requirements
├── runner.sh ..... Script used for training/evaluating
└── utils.py ..... Module used by ad_wgan.py
```

A.1.1 Usage

The scripts are intended to run on Unix operating system. For implementation details see section 4.3. To use the scripts follow this procedure:

1. Download the dataset[1] and save it as `creditcard.csv` to the same folder as the rest of scripts.
2. Call `make requirements` to install necessary Python packages.
3. Call `make convert` to convert and split the dataset.
4. Call `make encoder` to run the training and evaluation. This action runs training on four different portions of dataset at once. It takes about 2 hours on Intel® Core™ i7-6700 CPU and uses about 1.5 GB of disk space.
5. Call `make clean` remove files created during converting and training.

[1] `https://www.kaggle.com/mlg-ulb/creditcardfraud` (visited: 04/07/2019)