SECURITY OF GENERATIVE ADVERSARIAL NETWORKS

by

Kyrylo Rudavskyy

Bachelor of Science, Ryerson University, 2020

Bachelor of Mathematics, University of Waterloo, 2009

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the program of

Computer Science

Toronto, Ontario, Canada, 2022

Author's Declaration For Electronic Submission Of A Thesis

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

SECURITY OF GENERATIVE ADVERSARIAL NETWORKS

Master of Science, 2022

Kyrylo Rudavskyy

Computer Science

Ryerson University

Abstract

The overarching goal of this work is to explore the security landscape of Generative Adversarial Networks(GANs). In recent years, their adoption started to gain traction, and they are now used in many critical domains. Security is paramount in many of these fields. Since GANs are a system of two or more neural networks, security weaknesses in one of its components can be exploited against the system. This is the attack vector considered here. Specifically, this research evaluated the threat potential of an adversarial attack against the discriminator part of the system. Such an attack aims to distort the output by injecting maliciously modified input during training. The attack was empirically evaluated against four types of GANs, injections of 10% and 20% malicious data, and two datasets. The targets were CGAN, ACGAN, WGAN, and WGAN-GP. The datasets were MNIST and F-MNIST. The attack was created by improving an existing attack on GANs. The lower bound for the injection size turned out to be 10% for the improvement and 10-20% for the baseline attack. It was shown that the attack on WGAN-GP can overcome a filtering defence for F-MNIST. Furthermore, it was demonstrated that differentially private GANs are likely impossible to defend using current countermeasures.

Acknowledgements

I want to thank my supervisor, Dr. Ali Miri, for his guidance and support. His patient mentorship helped me overcome many challenges and stay focused on the goal in these difficult pandemic times.

I also want to thank Dr. Nariman Farsad and his machine learning lab. He helped me resolve complex problems and provided equipment. Without his kind disposition, this thesis would not have been possible.

Finally, I want to thank my wife, Irenes, for her encouragement and support.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

A *Generative Adversarial Network (GAN)* was first proposed by Goodfellow *et al.* in 2014 [24]. It is a machine learning technique whose goal is to learn the distribution of a set of data [36]. This is accomplished similar to a two-player game, where the players are neural networks. One player - called the *generator* - transforms a sample from the normal distribution into a sample that resembles real data. The other player - called the *discriminator* - tries to assess if a sample is real or fake based on its knowledge of the real data. After a sufficient number of iterations, the generator will produce samples that are hard to distinguish from the real ones. Thus, it will learn to transform a normal distribution into the data distribution.

GANs have a variety of applications in the real world. Alqahtani *et al.* [2] provide a survey of the more popular applications of GANs. The authors describe how GANs improve quality, reconstruct missing details, increase resolution, generate video and images, manipulate facial attributes, and more in the visual domain. Similar applications are shown in the audio domain.

The authors also provide noteworthy examples in the medical domain [2]. For instance, GANs were proposed to generate prescriptions for incurable diseases by learning from an existing database of drugs. Another proposal was to generate new medications by following

a similar strategy but using a database of biochemical data instead of a drugs database.

1.1 Synthetic Data

A promising application of GANs is synthetic data generation. Because GANs are capable of learning the distribution of a dataset, they can generate fake or synthetic data. This application is motivated by various necessities: insufficient data [51], imbalanced data [18], privacy [10], etc. Since the application of synthetic data is considered central to this thesis, several notable case studies will be discussed.

It is well-known that data augmentation - such as image rotation - can improve model generalization. In [51] the authors propose to augment the dataset size by generating synthetic data. Because a neural network's learning potential is not asymptotic with respect to the dataset size [21], data augmentation can theoretically provide additional gains in model performance.

Another well-known challenge in machine learning is learning from imbalanced datasets. Such datasets possess classes that are much smaller than others. In contrast, similar class size is a desirable property for machine learning tasks.[1] Moreover, on the extreme side, an imbalanced class can represent anomalies and, as such, is very difficult to augment. Authors in [18] note that synthetic data can help increase the size of such classes. Notably, Arora and Shantanu [5] highlight that synthesizing anomalies can improve the training of Intrusion and Detection Systems, which heavily rely on such data points.

The potential of GANs in the privacy-protection domain has recently received considerable attention. Recent results demonstrate that most existing measures are vulnerable to privacy-violation attacks [1, 14]. This motivates the need to develop new privacy-protecting technologies, such as GAN-generated synthetic data. However, in order to provide formal privacy guarantees, a technology called Differential Privacy (DP) was incorporated into these

[1] `https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data`

algorithms [20, 35]. The fact that the level of privacy provided by these algorithms can be mathematically quantified makes it possible to objectively refer to them as private.

1.2 Social Impact

It is worth mentioning that an unfortunate choice of the word "adversarial" carries a sentiment of an antisocial intent. The term adversarial strictly refers to how internal components of a GAN interact with each other. Although all advanced technology can transform society by the mere virtue of its existence, GANs are not *a priori* an offensive technology. However, they may be used that way, and to highlight the fact, a few examples of malicious usage will be provided.

Deepfakes[2] are an excellent example of a GAN-based technology that recently caused a widespread debate regarding its destructive potential. Deepfake usually refers to a video or image of a person generated by a GAN. The subject can appear to act or say almost anything, and the technology itself is very accessible [57]. On the social level, deepfakes can create a significant political impact. On the individual level, they may be used to inflict considerable distress.

Another notable example of malicious use is password guessing. Authors in [5] describe how GANs can be applied to that purpose. The generative ability of GANs is exploited to create plausible passwords by learning from a real-world dataset, thus replacing brute force or other approaches. These, and many other examples, leave the debate regarding social impact to a degree open. However, what is not open to discussion is that malicious usage was not the creators' original intent, unlike what the name of the technology may imply.

[2] `https://www.youtube.com/watch?v=T76bK2t2r8g`

1.3 Research Goals

It is clear from the above that GANs can be used in mission-critical systems. Alternatively, one may wish to disrupt a GAN in case of potential malicious usage of the latter. In both cases, the security of these algorithms is of interest. Thus, the overarching goal of this research is to explore the security of Generative Adversarial Networks.

It is a well-known fact that neural networks are susceptible to attacks [40, 29]. Some researchers even think that circumstances enabling attacks on neural networks are an innate characteristic of the deep learning process [28]. Since neural networks play a key role in most GANs, it is reasonable to assume that security problems from neural networks will affect them.

Particularly worrisome are *adversarial attacks*. These attacks were created to manipulate the output of a neural network by modifying the input [40]. Since GANs are a cleverly designed system of two or more neural networks, an adversarial attack works by targeting one of these networks. In fact, designing GANs that are resilient to such attacks is an active area of research [34, 6, 56, 60]. A possible impact of such an attack is low-quality output, or synthetic data, that does not resemble the original.

What is less understood is the feasibility of such attacks in the real world. The reason is that scholars often focus on defending either a single component of a GAN or a narrow case, such as defending a cycle-consistent GAN against itself. This is the first gap that my research is trying to close. Specifically, the following questions motivate my work:

- Given sample datasets, what amount of data must be modified to form a successful injection of malicious data?
- How to modify data to make it malicious?
- Do these modifications work across different variations of GANs?
- Is it possible to conceal the modifications from a human inspection?
- How effective is the attack in terms of its impact on the generated data?

- Is it possible to counter these attacks?
- Will the attack or the countermeasure work for private versions of GANs?

In short, I want to investigate the impact of adversarial attacks on synthetic data, which was not comprehensively examined before. Thus, the **first goal** is to design a real-world attack on Generative Adversarial Networks. The attack will be referred to as *Monkey-Wrench Attack (MWA)*. It will be demonstrated empirically that such an attack is possible and feasible under specific conditions. The attack works at training time via an injection of malicious samples into a dataset. It assumes a certain knowledge about the training process, but it does not require direct access to model parameters. Later we will show that this assumption, commonly referred to in the literature as a white-box attack, is not a significant obstacle for an attacker. Finally, the aim of the attack is a malfunctioning GAN that outputs distorted samples.

In designing MWA, two possible opportunities were explored to improve an existing proposed attack in the literature, which is my **second goal**. One was to make the attack more effective by forcing the GAN to produce lower-quality synthesis. The direct result of the latter is synthetic samples that resemble original data much less. Another was to make the samples more stealthy with respect to either a human or machine detector.

Stealthy, in this case, refers to the ease of detection. An automated (i.e. machine) detector may find adversarial samples with higher True Positive Rate (TPR) at the cost of a lower False Positive Rate (FPR). In this case, detection is considered easier. For a human operator, the situation is different. A person may come across adversarial samples during routine checks of data. If it is clear from the visual observation that the samples may carry a payload, then human detection is considered easy.

The answer to the first two goals helped me accomplish a **third goal**, which was to establish whether a defence strategy based on altering GAN algorithms is necessary. Such strategy is referred to here as *invasive* because it requires changing the algorithm. Doing so may introduce undesirable consequences, such as computational burden or uncertainty

of convergence. Moreover, these consequences are a death knell to the private versions of GANs. Any additional computation will likely deplete the privacy budget.

1.4 Findings

My work will show that adversarial attacks on GANs are challenging to execute in practice. This is likely because GANs have a degree of innate resilience to noisy data [9]. Of the four GAN variants, only one proved to be vulnerable across datasets. In addition, the attack required modifying 10-20% of input data.

However, in conditions that more closely resemble a real-world application of GANs, the attack proved to be more successful. The GAN version that was more vulnerable is one of the most advanced architectures proposed to date [36]. Moreover, the modifications of input data evaded detection on the more sophisticated dataset. Advanced GAN architectures and sophisticated datasets are more likely to be employed in applications.

In the setting mentioned above, the second goal was successfully achieved. An adversarial crafting process was devised that improved attack performance over the baseline approach. Moreover, a lower bound on the injection size was established for both cases. Finally, my adversarial samples were harder for a human operator to identify as malicious.

The stealth of adversarial samples is significant because they produce unexpected results and might be spotted during production. However, if the operator fails to identify the malicious intent behind these irregularities, the operator may fail to attribute a system malfunction to them. This will lead to the operator forgoing additional security measures and putting the data into production.

Finally, it will be investigated whether private GANs can be protected with a non-invasive strategy. As will be shown, the invasive strategy will introduce computational overhead that will probably make it inapplicable to the private versions of GANs. The implication is that private GANs may be defenceless against attacks such as MWA.

The above high-level conclusions rest on a series of contingent results. Most of the latter will be derived empirically and substantiated in Chapter 4. Below is a list of findings that, to the best of my knowledge, were not produced before:

F.1 The improved attack is likely to outperform the existing attacks when the loss function is complex and the dataset is sophisticated. Performance is measured by observing higher output distortion for equal size injections.

F.2 If the above conditions are met, the improved attack will likely overcome the countermeasure applied herein with greater success than the existing attack. Success is defined the same way as above.

F.3 My version of the attack required an injection of at least 10%, while the baseline had a lower bound in the 10-20% range.

F.4 It is likely possible to visually conceal the improved attack from a human operator.

F.5 Private GANs may be more resilient to an attack and more responsive to the countermeasure considered here.

In addition, the following observations confirmed previous results that are important to this work:

O.1 GANs often demonstrated resilience to noisy data [49, 9].

O.2 Since compromising a discriminator may not be sufficient to compromise a GAN, attack on the whole system may need to be studied (i.e. generated data) [34].

1.5 Thesis Outline

Following this introduction, the next chapter will provide a literature review of specialized technologies employed in this thesis and other closely related works. In Chapter 3, the reader will find a detailed description of the above attack and information regarding its implementation for empirical evaluation. Chapter 4 will describe and analyze the experiments to

empirically support the findings above. Finally, Chapter 5 will conclude the thesis with a discussion of future research directions.

Chapter 2

Literature Review

This chapter will explain the foundational technologies used in this work. Even though *neural networks* are a well-known technology, a short review will be provided because they are central to this research. This will be followed by *Stochastic Gradient Descent (SGD)* and its extension - *Stochastic Gradient Langevin Dynamics (SGLD)*. SGLD can provide a clue to the resilience of private GANs observed in **F.5**.

Next, we will focus on GANs, starting with the original formulation. Two families of GANs will be discussed: *Wasserstein* and *Conditional*. Original formulations of both will be reviewed, followed by their recent extensions: *Wasserstein GAN with Gradient Penalty (WGAN-GP)* and *Auxiliary Conditional GAN (AC-GAN)*. Additionally, a private version of WGAN and its underlying privacy-protecting technology - Differential Privacy - will be discussed. Mentioned GANs are targets of the attack proposed in this thesis. Having established the theoretical foundations of the target algorithms, we will review methods of evaluating synthesis quality, attacking neural networks, and defending them. These methods are an integral part of the empirical evaluation mentioned before.

2.1 Neural Networks

This section will provide a short review of a Fully-Connected Multi-Layer Network [37]. Assuming the network consists of r layers, m neurons, n data points, d features, a weight vector $w \in \mathbb{R}^{d\times 1}$, a vector of biases b, an input sample $x \in \mathbb{R}^{d\times 1}$, a label y and an activation function $\sigma(\cdot)$, then the first layer can be defined as $W^{[1]}X + b^{[1]}$ where

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{d\times n} \qquad Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \dots & z^{[1](n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{m^{[1]}\times n}$$

$$W^{[1]} = \begin{bmatrix} — & {w_1^{[1]}}^T & — \\ — & {w_2^{[1]}}^T & — \\ & \vdots & \\ — & {w_m^{[1]}}^T & — \end{bmatrix} \in \mathbb{R}^{m^{[1]}\times d} \qquad b^{[1]} = \begin{bmatrix} | & | & & | \\ b^{[1]} & b^{[1]} & \dots & b^{[1]} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{d\times n}$$

with [ ] brackets representing layers and () brackets - samples. For r layers the system becomes

$$\begin{aligned} Z^{[1]} &= \sigma(W^{[1]}X + b^{[1]}) \in \mathbb{R}^{m^{[1]}\times n} \\ Z^{[2]} &= \sigma(W^{[2]}Z^{[1]} + b^{[2]}) \in \mathbb{R}^{m^{[2]}\times n} \\ &\dots \\ Z^{[r]} &= W^{[r]}Z^{[r-1]} + b^{[r]} \in \mathbb{R}^{1\times n} \end{aligned}$$

Now, let's define a loss function (note $m^{[r]} = 1$)

$$J = \frac{1}{2}\left(Z^{[r]} - y\right)^2$$

and optimize it using the SGD method. This involves updating the paramaters $\theta = \{W, b\}$ for a number of iterations

$$\theta := \theta - \alpha \nabla J(\theta) \tag{2.1}$$

where α is a learning rate. SGD is commonly performed on random batches of data with the first-order term scaled proportionally to their size (i.e. $\frac{set\ size}{batch\ size}$). Finally, a technology - called *back-propagation* - is used to calculate the gradient efficiently by exploiting the capacity of matrix algebra to parallelize on GPUs. Note that besides SGD, many first-order optimization techniques can solve this problem [47].

Regularization and Stochastic Gradient Langevin Dynamics

Often a phenomenon called *overfitting* occurs in Neural Networks where the test error is larger than the training error. Goodfellow *et al.* [25, p.222] describe how to remedy it using a regularization method. They use a common approach of adding a squared L_2 norm to the cost function

$$J = \frac{1}{2}\left(Z^{[r]} - y\right)^2 + \frac{\alpha}{2} W^T W$$

where α modulates regularization strength. Also, other norms can be used.

The authors also described a Bayesian approach of adding noise to the weights. A more precise description is that the mean and variance of a Normal distribution of the weights is learnt [8]. This approach is similar in spirit to a less known technique called *Stochastic Gradient Langevin Dynamics (SGLD)*. The technique was first introduced by Welling and Teh [53].

SGLD consists of adding $N(0, \epsilon_t)$ to the first-order term, such that ϵ_t anneals under specific conditions and is twice the magnitude of the learning rate. The annealing conditions are

$$\sum_{t=1}^{\infty} \epsilon_t = \infty \quad and \quad \sum_{t=1}^{\infty} \epsilon_t^2 < \infty$$

As training progresses, the noise outgrows the first-order term and makes the algorithm

switch from optimization to sampling. In Bayesian terms, this switch makes sure that "posterior landscape" [52] is explored and, so, SGLD approximates a correct distribution. This, of course, works only under specific assumptions and conditions which are not relevant to this work.

2.2 Generative Adversarial Networks

As mentioned earlier in Chapter 1, GANs were invented by Goodfellow *et al.* in their seminal paper *Generative adversarial nets* [24]. This section will commence with the architecture presented in their paper. GANs were quickly extended by Mirza and Osindero with the introduction of the Conditional GAN (CGAN) [41]. This extension allows the generative model to be conditioned by a class label to generate samples from a chosen class. CGANs were further improved by Odena, Olah and Shlens [42] in 2017 with a variant called Auxiliary Conditional GAN (AC-GAN).

Another significant lineage of GANs was started by Arjovsky *et al.* [4] in 2017 with the introduction of the Wasserstein GAN (WGAN). Unlike the conditional family of GANs, WGANs cannot generate a sample from a specified class. However, their application domain is more general because conditional families lean towards the image domain. Moreover, WGANS have theoretical and practical advantages that will be discussed in this chapter. Finally, a notable improvement is Wasserstein GAN with Gradient Penalty (WGAN-GP) proposed by Gulrajani *et al.* [26] in 2017. Above GAN architectures, except the original, were used extensively in my research and will be described below.

2.2.1 Original GAN

Let P_X represent an unknown data distribution, then discriminator, D, is trained to assign a probability of sample belonging to P_X, while generator, G, is trained to create samples that resemble the original data. This process estimates data distribution, P_X, by implicitly

creating a generator distribution, Q_X. Finally, a GAN builds G by learning to transform samples from the normal distribution, P_Z. The process is expressed by a two-player minimax game with the following objective function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim P_X}\left[\log D(x)\right] + \mathbb{E}_{z\sim P_Z}\left[\log\left(1 - D(G(z))\right)\right] \tag{2.2}$$

Solving it minimizes Jensen–Shannon Divergence (JSD) between the distributions P_X and Q_X [22]. For a full algorithm, please refer to Algorithm 1 in the appendix.

Let us look at Equation 2.2 in more detail, maximizing V over D maximizes the probability of D detecting a true sample - represented by the first term - and a fake sample - represented by the second term. The reasoning behind the first term is self-evident. To maximize the second term, $D(G(z)))$ must be as close to zero as possible because $D \in [0, 1]$ (ref. Fig. 2.1). A small $D(G(z))$ implies good detection of fake samples. At the same time,

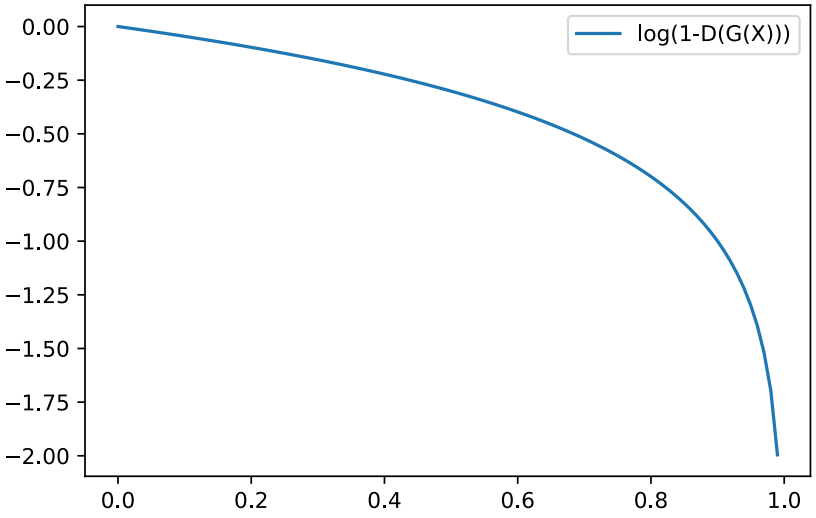


Figure 2.1: Second Term of GAN Objective Function

V is minimized over G. This happens when the second term is minimized. The second term is minimized when $D(G(z)))$ is close to one, which implies that D fails to detect a fake sample. Finally, once the loss function is constructed, **gradient descent** is used to solve

the optimization.

Another way to derive equation (2.2) is with a binary cross entropy function [39]

$$L(\hat{y}, y) = -ylog(\hat{y}) - (1-y)log(1-\hat{y})$$

where y is true label and $\hat{y}$ is a prediction. Then,

$$L(D(x), 1) = log(D(x))$$
$$L(D(G(z)), 0) = 1 - log(D(G(z)))$$

This approach has interesting information-theoretic properties that can help gain a deeper understanding of GANs.

2.2.2 Conditional GAN

Mirza and Osindero [41] propose a straight-forward extension to GAN by adding a class label or other auxiliary information as input to D and G. Then, equation (2.2) becomes

$$\min_G \max_D V(D,G) = \mathop{\mathbb{E}}_{x,y\sim P_{XY}} [\log D(x|y)] + \mathop{\mathbb{E}}_{z\sim P_Z, y\sim P_Y} [\log (1 - D(G(z|y)|y))] \tag{2.3}$$

and now we are minimizing JSD between joint distribtuions P_{XY} and Q_{XY} [22]. This, in turn, estimates $P_{X|Y}$ with $Q_{X|Y}$.

To add new information, Mirza and Osindero simply concatenated the inputs of G and D with the class label. In turn, they noted that GAN formalism is expressive enough to allow a rich representation of auxiliary data for a more complex generative process. That means that instead of concatenation, a more elaborate scheme can be devised to encode the necessary information.

Auxiliary Condition

Odena, Olah and Shlens further extended the CGAN by adding a third component. This component determines a sample's probability of belonging to a certain class [42]. Now, D, similar to the original GAN, returns the probability of a sample being real and the probability that x belongs to a certain class. So, $P(S|x), P(C|x) = D(x)$ where $S \in \{real, fake\}$ and C is the set of classes. They divide the objective function into two parts: log-likelihood of the correct source, L_S, and correct class, L_C, s.t.

$$L_S = \mathbb{E}\left[\log P(S = real|X_{real})\right] + \mathbb{E}\left[\log P(S = fake|X_{fake})\right]$$
$$L_C = \mathbb{E}\left[\log P(C = c|X_{real})\right] + \mathbb{E}\left[\log P(C = c|X_{fake})\right]$$

These two equations are combined into discriminator and generator losses s.t. $L_S + L_C$ is maximized to train D, and $L_C - L_S$ is maximized to train G.

It is worth mentioning that there are other variations of conditional GANs proposed in the literature (see [22]). In this thesis, I was interested in GAN architectures that also had a private version developed. As such, my investigation will only focus on CGAN and AC-GAN within the conditional family.

2.2.3 Wasserstein GAN

A breakthrough in GANs came with the introduction of WGANs in 2017. Arjovsky *et al.* [4] recognized the importance of GANs for learning distributions and approached the problem from a highly theoretical perspective. They realized that the original choice of Jensen–Shannon Divergence was subpar and proposed to replace it with a different measure of similarities between distributions - Wasserstein or Earth Mover Distance (EM).

This measure is smoother and remedies many theoretical shortcomings of the original GAN [3]. Smoother means that it is continuous everywhere and differentiable almost everywhere. Clearly, certain assumptions must be met for this to hold, but they usually do not

present a significant obstacle.

The primary benefit of the Wasserstein distance is the introduction of convergence properties into the process. As a result, WGAN continues to improve with additional epochs, while the original may not. In addition, WGAN remedies mode collapse and vanishing gradients problem. Mode collapse refers to the low diversity of samples generated by a GAN, and vanishing gradients stop the algorithm from improving.[1] These make the original GAN effectively obsolete.

If we are estimating the real distribution, $\mathbb{P}_r$, with $\mathbb{P}_\theta$, then Wasserstein distance can be expressed[2] as

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \max_{||f||_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)] \tag{2.4}$$

where $f : \mathcal{X} \to \mathbb{R}$ is 1-Lipschitz for a compact metric space $\mathcal{X}$. In our case $\mathcal{X}$ is a space of images $[-1, 1]^d$, but can also be $[0, 1]^d$. Function, f, can be K-Lipschitz, if multiplying W by K.

Now, let f_w represent the discriminator and g_θ the generator. Let the distribution $\mathbb{P}_\theta$ belong to $g_\theta(Z)$ for $Z \sim N(0, 1)$ and g_θ be continuous. Then, there is a solution to (2.4) and

$$\nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{z \sim P_z}\left[\nabla_\theta f_w(g_\theta(z))\right] \tag{2.5}$$

Finally, we must clip the weights to enforce the Lipschitz condition and backpropagate through 2.4 and 2.5 to solve the optimization. The objective function can also be expressed as

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim \mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z \sim P_z}[f_w(g_\theta(z))] \tag{2.6}$$

where f_w is a parameterized function and $\mathcal{W}$ is a compact space.

This section briefly described the Wasserstein GAN. For a complete algorithm, please refer to Algorithm 2 in the appendix.

[1] `https://developers.google.com/machine-learning/gan/problems`

[2] using Kantorovich-Rubenstein duality

Gradient Penalty

Gulrajani *et al.* [26] observed that the main problem with WGANs is weight clipping. Authors demonstrated that improper clipping can either lead to vanishing or exploding gradients. Also, clipping incentivizes the generator to learn simple functions, thus losing representational capacity. To solve these problems, the authors enforced the Lipschitz condition with a gradient penalty instead of clipping. The idea here is that under normal WGAN assumptions, bounding the weight norm is enough to enforce the Lipschitz constraint. Authors achieve the constraint by adding the following penalty to (2.6)

$$\lambda \mathop{\mathbb{E}}_{x\sim\mathbb{P}_h} \left[(||\nabla_x f(x)||_2 - 1)^2\right]$$

where λ is a constant.

Imposing the penalty everywhere can be computationally prohibitive, so the authors impose it on a cleverly chosen subset. They sample from $\mathbb{P}_h$ which they describe as: "We implicitly define $\mathbb{P}_{\hat{x}}$ sampling uniformly along straight lines between pairs of points sampled from the data distribution $\mathbb{P}_r$ and the generator distribution $\mathbb{P}_g$." [26, p.4] This is a slightly convulated way of saying that if $u \sim U(0,1),\ x \sim \mathbb{P}_r,\ y \sim \mathbb{P}_\theta$, then a sample from $\mathbb{P}_h$ can be constructed as $u \cdot x + (1-u) \cdot y$. Note that their notation $\mathbb{P}_{\hat{x}},\ \mathbb{P}_r,\ \mathbb{P}_g$ corresponds to $\mathbb{P}_h,\ \mathbb{P}_r,\ \mathbb{P}_\theta$ above. For a full algorithm, please refer to Algorithm 4 in the appendix.

2.3 Differential Privacy

This section aims to introduce another GAN architecture, but unlike the previous ones, this architecture is intended for datasets with sensitive information. On a high level, privacy refers to protecting information belonging to an individual (or thing) that participated in a database. But, this definition fails to explain what it means to protect.

One possibility is to deny third-parties access to data. To accomplish this, access to

models trained on data must also be denied. The reason is that privacy attacks exist that can extract information from trained models, including GANs [58, 55]. As a result, most machine learning algorithms would be impossible to apply. For this reason, a different definition of privacy will be used.

In 2006, Cynthia Dwork introduced the concept of *Differential Privacy (DP)*, which blossomed into a rich field of scientific endeavour [17, 15, 16]. This concept formulates a mathematical measure for privacy loss. Let us consider two datasets, D and D', that differ by one entry. For example, an individual is present in D and not in D'. Now, let f be a function performed on a dataset and $\mathcal{M}$ be an anonymization mechanism that obfuscates the output of f. Then for $\mathcal{M}$ to be (ε, δ)-DP

$$P[\mathcal{M}(D) = c] \leq exp(\varepsilon) \cdot P[\mathcal{M}(D') = c] + \delta$$

where c is some output of $\mathcal{M}$ and $1 - \delta$ is the probability that $(\varepsilon, \delta = 0)$-DP will hold.

There are many ways to create an anomyzation mechanism, $\mathcal{M}$. One way is to add a specially crafted Laplacian random variable to each coordinate of the output of f. To do that, we must find what Dwork calls the *sensitivity* of function f, which is

$$\Delta f = \max_{D,D'} ||f(D) - f(D')||_1$$

giving us

$$Y \sim Lap\left(scale = \frac{\Delta f}{\varepsilon}\right)$$

for $\delta = 0$ and centered at zero. Alternatively, it is possible to add a Guassian with $\mu = 0$ and $\sigma^2 = \Delta f \ln(1/\delta)/\varepsilon$.

Two important properties of DP will be used extensively in this thesis: *composition* and *postprocessing*. Let us begin with the composition property [16, p.49, 52]. Suppose that k function calls or queries are performed and each one is (ε, δ)-differentially private. Then,

together, the calls are $(\varepsilon', k\delta + \delta')$-differentially private for $\varepsilon, \delta, \delta' > 0$ and

$$\varepsilon' = \sqrt{2k \ln(1/\delta')} \cdot \varepsilon + k\varepsilon(e^{\varepsilon} - 1)$$

It is also possible to go in the opposite direction. If the total budget, $(\varepsilon', k\delta + \delta')$, is known and $0 < \varepsilon' < 1, \delta' > 0$ then per query budget, ε, is

$$\varepsilon = \frac{\varepsilon'}{2\sqrt{2k \ln(1/\delta')}}$$

Finally, the postprocessing property guarantees that properly anonymized outputs are immune to any processing post factum [16, p.19].

DP may seem a bit low-level on the first look because it does not quantify risk directly. For example, learning the definition and the mechanics still does not clearly convey the probability of a privacy-violation event. However, the low-level nature of the definition makes it easy to apply in various fields, such as machine learning. This was taken advantage of during the development of a private WGAN, which will be discussed below. On the other hand, by building a mathematical layer of abstraction above the definition, it is possible to quantify the probability of a privacy-violation event on a per-case basis [32].

2.3.1 Private GAN

A differentially private version of WGAN was proposed by Xie *et al.* in 2018 [55]. Privacy is achieved by adding a specially crafted Gaussian to each gradient of the Wasserstein distance w.r.t. the discriminator weights. This is done when updating the discriminator. In addition, the weights of these gradients must be clipped in a specific way. If these two main conditions are met, then the new WGAN is well on its way to guaranteeing (ε, δ)-DP.

The following steps are necessary to achieve DP. First, the activation function of the discriminator, σ, and its derivative must be bounded by $B_\sigma, B_{\sigma'}$. Also, every data point

must be bounded. Second, the clipping coefficient must be calculated such that

$$c_p \leq \frac{1}{m \cdot B_{\sigma'}}$$

where m is the number of nodes in the biggest hidden layer. This guarantees that the mentioned norm is bounded by a constant

$$c_g = 2c_p B_\sigma B_{\sigma'}^2 \sum_{k=1}^{H-1} m_k m_{k+1}$$

where H is the number of layers above the input layer, and m_k is the number of nodes in a layer. Finally, a per iteration noise scale can be defined as

$$\sigma_n = 2q\sqrt{n_d \log \frac{1}{\delta}}/\varepsilon$$

where n_d is the total number of discriminator iterations, and q is the sampling probability (i.e. batch size/number of samples). With these pieces in place, it is possible to create a Differentially Private Wasserstein GAN (DP-WGAN). For a full algorithm, please refer to Algorithm 3 in the appendix.

2.4 Fréchet Inception Distance

Fréchet Inception Distance (FID) was introduced by Heusel *et al.* [27] in 2017 specifically for evaluating a GAN's ability to generate images. FID determines how similar two sets of images are. Thus, generated images are compared with the originals to evaluate a GAN. The higher the score, the lower is the similarity.

This metric is considered superior for the following reasons. The authors observe that FID reflects the human perception of visual similarity between images. Moreover, the metric is sensitive to mode collapse and resilient to noise [7]. Mode collapse was discussed in Section

2.2.3 and noise refers to undesirable artifacts in images. In fact, many papers cited in this thesis used this metric to evaluate GANs.

The main idea behind the metric is to calculate the Frechet distance between two distributions derived from the original and synthetic data. Rafael Valle describes this process well in [50]. The tricky part is building the distributions. The author notes that Heusel *et al.* do not derive the distributions directly from the data. Instead, they pass the images through part of a pre-trained Inception [48] model. That is, they take the output from an intermediate layer.

Then these two groups of activations, from original and synthetic images, are used to create the distributions. This is done by fitting multivariate Gaussians using the means and covariances of the two groups. Finally, the Frechet Distance is calculated between these two distributions. The formula for doing this is

$$FID = ||\mu_r - \mu_s||_2^2 + trace\left(\Sigma_r + \Sigma_s - 2(\Sigma_r\Sigma_s)^{\frac{1}{2}}\right)$$

where $\mathcal{N}(\mu_r, \Sigma_r)$ represents the distribution derived from the real data, and $\mathcal{N}(\mu_s, \Sigma_s)$ from synthetic.

2.5 Projected Gradient Descent

Projected Gradient Descent (PGD) was chosen in this thesis because it is considered to be a strong attack. Madry *et al.* [38] provide strong evidence that PGD is a universal first-order optimization attack. If a network is resilient to PGD, then it is resilient to all attacks based on first-order optimization. Schöttle *et al.* [46] state that PGD is a state-of-the-art attack. Moreover, this conclusion is affirmed by authors in [11, 34, 59].

However, an extension of PGD exists whose authors claim an improvement [12]. They provide a comparison where they contrast their PGD variant's performance against the original's. In some cases, the difference is significant. However, this was not the case in the

context of this thesis. The authors reported a median 3% improvement on the dataset that intersects this thesis (e.g. MNIST). Considering that this extension is not yet established in the field and the improvement is small, this thesis will use the original PGD.

Madry formulated optimization-based attacks and countermeasures as follows [38]. Let δ lie in l_∞-ball, $\mathcal{S}$, around a sample x from distribution $\mathcal{D}$ with corresponding label y and let θ be model parameters, then

$$\min_\theta \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y) \right] \tag{2.7}$$

The inner optimization produces adversarial perturbations δ, which is the PGD attack proposed by the author. On the other hand, the outer optimization is a countermeasure to PGD.

Attack strength is varied by the ϵ parameter. This parameter controls the magnitude of δ. In the case of images, the bigger the magnitude, the higher the intensity a set of altered pixels can attain. This makes an attack more noticeable to the naked eye. The other way to modulate strength is by bounding $\mathcal{S}$ with a weaker norm (i.e. l_2), which will make it less noticeable.

2.6 Classifier Defense

Samangouei *et al.* [45] propose a method called Defense GAN (Def-GAN) to make classifiers resilient against adversarial input attacks, such as PGD. This method will be used extensively in this thesis. Def-GAN, as the name suggests, is based on GANs and works as a preprocessing step. Assuming that the defender possesses enough clean data to train a GAN, they can use that GAN to filter out adversarial samples.

Specifically, instead of giving input, x, of unknown origin to the classifier, the GAN is used to construct a sample similar to x. This is accomplished by solving the following

minimization problem

$$\min_{z \sim N(0,1)} ||G(z) - x||_2^2 \tag{2.8}$$

which yields a solution z^*. Now, instead of giving x to the classifier, we replace it with $G(z^*)$, which effectively projects x onto the range of G. Authors [45] base their approach on the logic that if the GAN has enough capacity, then

$$\mathop{\mathbb{E}}_{x \sim data} \left[\min_{z \sim N(0,1)} ||G(z) - x||_2 \right] \to 0$$

Moreover, the authors observed that since clean images should lie closer to the range of G than malicious ones, their method can be used as a *detector* of adversarial samples

$$||G(z*) - x||_2^2 \underset{no\ attack}{\overset{attack}{\gtrless}} \theta \tag{2.9}$$

where $\theta > 0$.

2.7 GAN Defenses

This section will explore existing research on defending GANs. In 2019, Liu *et al.* [34] proposed a method to defend a discriminator of an AC-GAN against adversarial attacks, such as PGD from Section 2.5. The main idea behind this work is to train the discriminator on adversarial samples. They create adversarial samples using PGD (ref. Sec. 2.5). As a result, these samples will contain the original image plus adversarial noise. This is often referred to as *adversarial training*. The authors claimed their approach defends the discriminator against adversarial input and improves convergence and quality.

However, from a practical standpoint, their defence fails at stronger levels of PGD. For instance, at PGD_∞ 0.04 and 0.08, discriminator accuracy drops from approximately 81% to 57% and 30%, respectively. Moreover, the authors tested their method on two datasets, and the cited result is the better case. Considering that this thesis requires a PGD strength

closer to 1.0, the above defence is insufficient.

While Liu *et al.* proposed a method to defend the discriminator against adversarial input, Xu *et al.* [56] devise a method to protect a GAN against adversarial *output* of the discriminator. Their adversary samples perturbations from a finite set with a certain probability. So, a perturbation ψ_i will be chosen from a set $\{\psi_1, \psi_2, \ldots, \psi_n\}$ with probability p_i. The result will be $\psi_i(D(x))$.

Moreover, they constrained the adversary such that $\psi_i(D(x)) > 0.5$. The authors defend against this adversary by passing the discriminator and the generator through functions chosen from a special set $\mathcal{H}$. These function must satisfy two criteria:

1. strictly increasing and differentiable in [0,1]
2. $f(x) = -f(1-x), \forall x \in [0,1]$

Barring a few smaller assumptions, they consider the discriminator formally resilient.

Zhou and Krähenbühl [60] propose a method similar in spirit to that of Liu *et al.* However, instead of adversarially training the discriminator, they offered to adversarially regularize the discriminator. This involves adding a penalty term to the GAN's objective function. As a result, the discriminator will be robust to perturbed generator outputs.

The authors further elaborated on what they mean by robustness. For a bounded norm of perturbation, the difference in the expectation of perturbed and clean output(*i.e.* $D(G(z)) - D(G(z) + \delta)$) will also be bounded. Despite adding new assumptions, their method empirically outperforms the closest competitors. However, the authors note that GAN's convergence may be negatively affected. Moreover, when visually comparing samples generated by WGAN-GP and their method, convergence seems subpar for the latter.

The last defence strategy to be covered in this section was proposed by Bashkirova *et al.* [6] in 2019. This case is different because it protects the GAN against itself. Bashkirova considered the case of cycle-consistent GANs. Such GANs map samples between domains. For example, a satellite image of a neighbourhood can be translated to a city map(i.e. Google Maps).

Cycle-consistent GANs have two generators and two discriminators. One generator translates from some domain A to B, and the other does the opposite. Similarly, each discriminator evaluates a sample's domain membership. The loss function contains cycles of translation to a given domain and back(i.e. A to B to A). Sometimes, these cycles can generate samples that fit the definition of an adversarial attack. Thus, Bashkirova calls this phenomenon a self-adversarial attack. This attack is similar to the one considered in Liu's work discussed above. However, the adversarial samples are not crafted with a PGD. Instead, these samples are byproducts of the synthesis process.

To counter this specific threat, the authors suggest a variation of *adversarial regularization* that we saw in Zhou and Krähenbühl's approach above. It works as follows. Let us refer to the two generators as F and G, then the penalty term is

$$||F\left(G(x) + \mathcal{N}(0, \sigma)\right) - x||_1$$

where $\mathcal{N}(0, \sigma)$ is a low-amplitude Gaussian. By applying such a penalty to both the generator and the discriminator, Bashkirova improved domain translation. However, their work extends Zhou and Krähenbühl's from above and, as such, inherits its shortfalls.

The technologies discussed in this chapter will be used in the rest of the thesis. The chapter started with neural networks because they are central in all sections of this chapter. However, the main subject of the thesis is a GAN because it is the target of the attack proposed here. The discussion of this generative technique started with the original GAN. Even though the original is obsolete, it was included to create a model of the technology from which other GANs can be derived. Specifically, CGAN, ACGAN, WGAN, WGAN-GP, and DP-WGAN were studied because they are the targets of the proposed attack. The discussion of GANs was complemented by a method to evaluate their performance, which is the FID score.

Having covered GANs, the chapter proceeded to study security concepts related to them. First, a way to attack a neural network was presented. This was followed by a countermeasure. Finally, the chapter closed with methods to defend not just a single neural network but a system of them, which is a GAN.

The next chapter will unite concepts presented in the current one to lay a theoretical foundation of the thesis' main focus - the security of GANs. Then, an attack on GANs will be proposed, and a way to counter it using existing techniques will be presented. Following that, we will look at empirical results and future research directions.

Chapter 3

Monkey-Wrench Attack: Theory and Implementation

This chapter will begin with a theoretical construct of a possible attack on GANs, which I will refer to as Monkey-Wrench Attack (MWA). Following that, the reader will be introduced to the data used for this thesis. Then, with an understanding of the attack and the data, we will proceed to the implementation details of MWA in the context of four GAN variants: CGAN, ACGAN, WGAN and WGAN-GP. Once a theoretical and practical framework is built-up, a discussion of attack detection strategies will be possible. Finally, the chapter will conclude by discussing the above within the privacy paradigm.

3.1 Attack Description

A GAN can be viewed as a "student-teacher" system, and MWA works by nudging the learning vector in the wrong direction. Forcing the teacher - the discriminator - to assign a surplus of value to bad examples will provide wrong training directions to the student - the generator. Alternatively, the goal can be achieved by doing the opposite - making the teacher assign too little value to legitimate examples. In this thesis these two types of bad examples

are referred to as *Early Epoch Decoy (EED)*[1] and *Downgrade Decoy (DGD)*, respectively. They can also be seen as two variations of MWA.

Starting with the first type, decoys are created by replacing a subset of data with bogus information while keeping the labels intact. For example, one could simply replace these samples with normal noise. However, normal noise is easy to detect, so a different source of bogus data was used.

Instead of normal noise, I used samples generated by a GAN that was trained for only an epoch or two. In this thesis, these samples are referred to as *EEDs*. For this purpose, a GAN is trained using similar settings as the victim GAN. As a result, adversarial samples will share similarities with the original data, making them harder to detect or recognize as malicious. *Moreover, by overemphasizing their value to the teacher, the hope is that the system will treat premature convergence as a desirable goal.*

The discriminator will be nudged to overestimate the value of decoys using the following technique. Once decoys are created, a PGD attack will be applied to this subset of data. The discriminator is essentially a classifier, so theoretically, it is possible to modify samples with PGD such that the discriminator will classify them according to the attacker's needs. In this case, the discriminator will classify decoys as more likely to be real.

In the *downgrade* variation the opposite occurs. PGD is applied such that the discriminator assigns little value to a subset of legitimate data. The concept of attacking a GAN by applying PGD to a subset of the training data is not new and was mentioned in Section 2.7. However, to the best of my knowledge, the *early epoch* approach had not been tried before.

Once malicious samples are produced, they are combined with the rest of the dataset and given to a victim GAN for training. If the above hypothesis holds, the victim will produce either more malformed samples or a larger number of them. In both cases, the generating process will be compromised.

Finally, MWA can be considered a **white-box attack** because the attacker assumes

[1]Sometimes the term "Early Stop Decoys" is used instead of "Early Epoch Decoys"

knowledge of GAN hyperparameters and network architectures but not direct access to the weights. Typically, a black-box attack assumes little knowledge of the above [40]. The attack schematic can be seen in Fig. 3.1. The following sections will focus on the implementation.

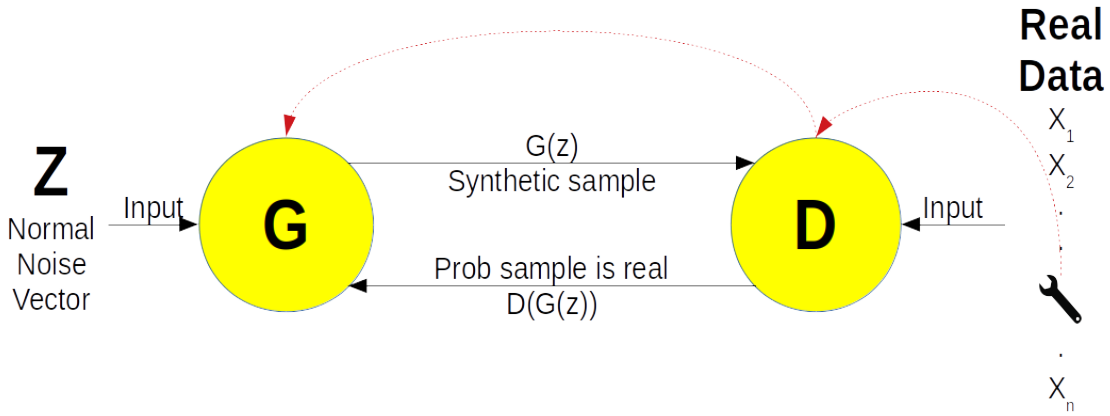


Figure 3.1: MWA Topology.[2]

3.2 Data

This thesis used two datasets: Modified National Institute of Standards and Technology Database (MNIST) [31] and Fashion-MNIST (F-MNIST) [54]. MNIST is a popular dataset, and it has been used extensively in the literature, particularly on GANs. This dataset consists of 70k black-and-white images of handwritten digits (i.e. 0-9). Each digit is a tensor of 28 pixels high and wide. An individual pixel can have an integer intensity anywhere between 0 and 255. As recommended by Goodfellow *et al.* [25, p.419] for vision datasets, images are normalized to [-1,1] with a mean of 0.5 and standard deviation of 0.5. Finally, the authors divided this dataset into 60k training and 10k testing samples. Same subsets will be used here as well.

F-MNIST was designed as a "direct drop-in replacement" [54, p.1] for MNIST. The authors note that MNIST is being challenged in the community as a reliable set due to

[2]Wrench Icon By Estelle DB - Own work, CC BY-SA 4.0, `https://commons.wikimedia.org/w/index.php?curid=57721428`.

its simplicity and other factors. Because of the popularity, the authors decided to create a better dataset that is easy to substitute. Thus, F-MNIST was created. It is identical in all respects except the value of the pixels. Instead of digits, the set contains pictures of fashion items from an online store. The number of classes is also the same, but instead of 10 digits, there are 10 clothing categories. This dataset was pre-processed the same way as MNIST.

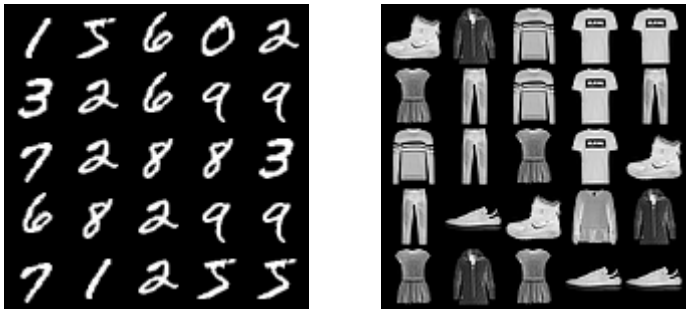


Figure 3.2: MNIST(left) and F-MNIST(right)

3.3 Decoys

As described in Section 3.1, the first step of MWA is to create decoys. Downgrade decoys are taken directly from the training data. Early epoch decoys, on the other hand, need crafting. They were produced by creating four types of GANs - CGAN, AC-GAN, WGAN, WGAN-GP - using training data described in Section 3.2.

The main hyperparameters[3] used to train the GANs are in Table 3.1. As mentioned earlier, decoys are generated by a GAN that stopped training at an early epoch. This epoch can be found in the table under the "Decoy Epoch." Remaining hyperparameters can be found in Table 3.2. The neural network architectures of the discriminator and the generator for these GANs were borrowed from [33]. One can see a description of the networks in Appendix B.

To save computational power, only one class out of the ten available was used with CGANs and ACGANs. Since these two variants are conditional, it is possible to specify

[3]Terms *hyperparameters* and *parameters* are used interchangeably in this thesis.

Type	Epochs	Batch Size	Decoy Epoch	Target Class
CGAN	100	125	1	6
ACGAN	50	250	1	6
WGAN	100	125	1	n/a
WGAN-GP	50	62	1	n/a

Table 3.1: GAN Training Parameters

which class one prefers to generate. Thus, malicious samples were created and evaluated only with one class. The "Target Class" column in Table 3.1 refers to this class. However, GANs themselves were created from the entire training dataset.

	Optimizer	Learning Rate	b_1	b_2	Latent Dimension	Critic Iterations	Clipping Value
CGAN	Adam	2×10^{-4}	0.5	0.999	100	parallel	n/a
ACGAN	Adam	2×10^{-4}	0.5	0.999	100	parallel	n/a
WGAN	RMSprop	5×10^{-5}	0.5	0.999	100	5	0.01
WGAN-GP	Adam	2×10^{-4}	0.5	0.999	100	5	0.01

Table 3.2: Secondary GAN Training Parameters

PyTorch-GAN package by Erik Linder-Norén [33] was used as part of my implementation. As the name implies, the package is based on PyTorch[4] - a GPU-based, machine learning framework [44]. Linder-Norén's package was adapted to interface with the main experiment, other external libraries and support multi-GPU computation.

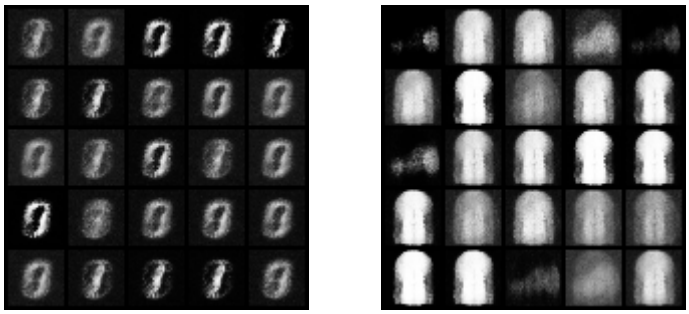


Figure 3.3: Examples of MNIST(left) and F-MNIST(right) Decoys Without PGD

[4]https://pytorch.org/

3.4 Adversarial Perturbations

At this stage of the attack, decoys are modified such that the discriminator will either overestimate or underestimate them. This is accomplished by adding adversarial perturbations using PGD described in Section 2.5. The goal is to create a specially crafted noise - referred to as *adversarial perturbations* - and add it to the decoys. These perturbations change the discriminator's valuation. Since PGD requires a model to craft samples, a fully trained discriminator from the decoy production step (ref. Section 3.3) was used. Visual examples of the result can be seen in Figure 3.4.

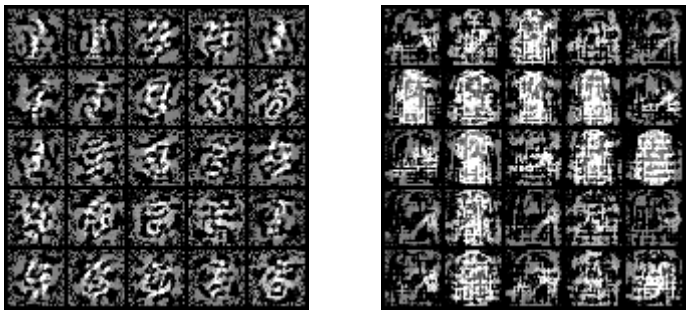


Figure 3.4: Examples of MNIST(left) and F-MNIST(right) Decoys with PGD

The biggest challenge at this stage was providing an appropriate loss function to PGD as stipulated in Equation 2.7. Binary Cross Entropy (BCE) was chosen for WGAN, WGAN-GP, and CGAN. The target was either 1.0 (early stop) or 0.0 (downgrades), and the input was whatever validity the discriminator assigned, passed through a Sigmoid function. Again, since the discriminator can be viewed as a binary classifier (i.e. real or fake), BCE is an appropriate choice. The mathematical expression for BCE[5] is

$$l(x, y) = mean(L) \quad L = \{l_1, ..., l_{batch\ size}\}^T, \quad l_n = -w_n[y_n \log x_n + (1 - y_n) \log(1 - x_n)]$$

where x represents the input probabilities and y the target probabilities.

An important note is that the discriminator in Wasserstein architectures provides a score

[5] `https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html`

of a point being real instead of probability [4]. Passing this score through a Sigmoid function and treating the result as a probability significantly reduces the architecture's theoretical complexity and practical capacity. Nevertheless, this approach was sufficient for this research while leaving open an intriguing research question!

The loss function for performing PGD on the AC-GAN discriminator was constructed differently. The reason is that the discriminator is composed of two neural networks that share the same base network. One network measures data validity, while the other predicts the label. However, both must be used during the PGD step because the networks are intertwined. For this reason, the loss function was similar to the one used in the actual AC-GAN, which is

$$\text{BCE(predicted validity of data, target validity of data)}\cdot 0.5 + \text{CE(predicted labels, real labels)}\cdot 0.5$$

where BCE is Binary Cross Entropy loss, CE is Cross Entropy loss, and only target validity of data is chosen according to the goal (decoy/downgrade).

This is slightly different from the actual discriminator loss used in AC-GAN. That loss applies the above to both real and synthetic samples and then takes an average. However, for MWA, applying PGD only to the real ones is sufficient. Besides the loss, other PGD hyperparameters were chosen as in Table 3.3. For completeness, a targeted version of PGD was used.

Strength(ϵ)	Norm	Clipping	Iterations	ϵ-step	Targeted
1.0	∞	[-1.0, 1.0]	100	0.01	True

Table 3.3: PGD Training Parameters

The code was based on the Advertorch library created by BorealisAI [13]. Like the GAN library, the code needed programming to interface with the main experiment and other external packages.

3.5 Detection

This section proposes a method for defending against MWA. The countermeasure used is based on Def-GAN described in Section 2.6. However, in that section, Def-GAN was primarily considered in its capacity as a detector and not as a pre-processing step. The reason is that pre-processing is sound when defending a classifier. Since the target is a GAN, using projections of data is not appropriate. This would entail building a GAN from synthetic data, which is a less reliable approach. The reason is that estimating data distribution is a more difficult learning task than classification because the latter is a subprocess of the former.

Def-GAN relies on a clean generator to detect adversarial samples. To source the generator, the same GAN was used as the one for the decoys and PGD steps above, Sections 3.3 and 3.4 respectively. This decision was made to reduce computational complexity.

However, using the same GAN possibly created a new problem. Since GANs used for crafting decoys and building the detector were the same, they were trained on the same data. Ideally, the detector would have been built on a different subset of data. Exposing the data used for the creation of decoys to the detector risks having it learn more than it would have under normal operation. Moreover, the detector usually would be trained on a subset of the data and not the whole of it. As a result, detection figures might be more optimistic than they should be.

Detection performance was measured by creating ROCs and AUC scores using the following procedure:

1. Randomly select at most 1000[6] points from training and adversarial sets.
2. Label clean and adversarial points accordingly (i.e. 0,1).
3. Combine the above into a single dataset.
4. Produce 10^7 candidates for z^* using SGD described in Section 2.5.

[6]Conditional GANs may look at a single class that may not have enough samples for a larger draw.
[7]This number was chosen due to computational constraints.

a) SGD performed over n = 200 iterations with a learning rate (α) of 10.

b) Learning rate decayed according to $\alpha_{t+1} = \alpha_t \cdot 0.1 \cdot \exp(1/\lfloor 0.8 \cdot n \rfloor)$.

c) Used Mean Squared Error (MSE) loss throughout.

5. Calculate losses for all z^* candidates and find the one with the smallest loss.
6. Randomly sample 50 θ's from [0,1].
7. Calculate $MSE(G(z^*),\ x) > \theta$ for all θ to produce ROC/AUC.

Code from [43] was used for the implementation of the above.

3.6 Transferability

It may be argued that the white-box assumption is a significant drawback of the MWA. This section will argue to the contrary. By now, the reader should understand the amount of information an attacker must possess about a victim GAN. This includes the type of GAN, loss functions, the architecture of the discriminator and the generator, training hyperparameters and more. Some of these can be expected to be publicly available information.

For example, it is relatively simple to guess the GAN variant that the victim may use. GANs possess unique sets of advantages and drawbacks that make them specialized. Thus, if the training data is available, it is possible that the target GAN variant can be guessed. Otherwise, knowing the target GAN type is the biggest hurdle for an attacker.

Once the target GAN variant is established, it is possible to overcome the other limitations of the white-box approach. Madry *et al.* [38] utilized a well-known phenomenon called *transferability*. When a neural network architecture is not available, it is possible to craft adversarial samples on a surrogate model. Moreover, Madry *et al.* demonstrated that transferability is also applicable for PGD.

Miller *et al.* [40] observed that the biggest obstacle to achieve transferability is targeting the attacks. Since GANs are mostly a case of binary classification, targeting is not an issue. The reason is that there is only one possible target - the opposite. This effectively makes

MWA an *untargeted* attack. Thus, it is likely that the white-box limitation can only represent a minor hurdle for an attacker.

3.7 Private GAN

This section describes the implementation of MWA within the privacy paradigm. To accomplish this, a private version, called DP-WGAN, was used. This variant was described in Section 2.3.1. To train DP-WGAN one must set the privacy parameters. Moreover, privacy parameters directly depend on the number of epochs, batch sizes, discriminator network architectures and more. When using these settings from a non-private version, a DP-WGAN will not converge because it will add too much anonymization noise.

The following steps were mitigated to save the privacy budget. First, the network architecture of the discriminator was significantly simplified to accommodate the privacy requirement. In addition, a different activation function was chosen. As a result, the sigmoid function has a lower bound of its derivative, saving the privacy budget. Furthermore, smaller and fewer layers were chosen for the same reason. To see the new network architecture, please refer to Appendix B. Finally, fewer epochs with smaller batch sizes were used. This was also done to save the budget.

All these changes reduced the GAN's capacity to represent data. To accommodate this, the dataset was reduced to only two classes. Moreover, the chosen classes are easy to discern, thus simplifying the GAN's task. The classes are one and zero. A listing of training parameters can be seen in Table 3.4. The code used for DP-WGAN is from [19].

Epochs	Batch Size	Critic Iter.	Decoy Epoch	ϵ	δ	σ	Clipping Coef.	Optimizer	Learning Rate
50	16	5	2	20	10^{-5}	0.007	0.01 (0.1 non-dp)	RMSprop	5×10^{-5}

Table 3.4: DP-WGAN Training Parameters

Experiments involving the private GAN often required a *non-private counterpart*. To stay consistent, the same architecture was used as above. Also, the settings were the same

as in Table 3.4 except that the sigma (σ) parameter was set to zero to build a non-private version of this GAN.

Clipping Parameter and Noise Scale

As was described in Section 2.3.1, to set the clipping value, c_p, three values must be determined: bounds on the activation function, its derivative and the largest layer of the critic barring the first hidden one. The sigmoid function is bounded by one, and its derivative is bounded by a 1/4 [25, p.83]. Thus, the clipping coefficient was set as $c_p = 0.01$.

Now, to calculate c_g the total number of critic iterations must be established. After removing most classes, the size of the dataset is 1.2×10^3. Each epoch length is $n_{data}/(nb\_iter_{critic} \times n_{batch}) = 1.2 \times 10^3/(5 \times 16) = 150$, thus number of total critic iterations is 150×5. By substituting these values into formulas from Section 2.3.1 and keeping in mind that $\sigma = \sigma_n \times c_g$, we get the values in in Table 3.4.

Detection

Def-GAN detector operates using the same settings and procedure as in the non-private version above (ref. Section 3.5). There is only one difference. A separate GAN was built for the detector. This concludes the discussion on the implementation of MWA and the countermeasure within the private paradigm.

This finalizes the description of the methodology employed in this thesis. Having provided a full description of the attack and the data, it is now possible to evaluate its performance. The next chapter will provide a thorough description of the experiments and the subsequent analysis that was used to validate the findings provided in Section 1.4.

Chapter 4

Experiments and Analysis

This chapter will present the experiments used to explore the security landscape of GANs. It will start with a description of the former, followed by an analysis. First, the null hypotheses will be used to establish success conditions for MWA. Then, the second hurdle for MWA will be to overcome detection. In the process, the findings introduced in Chapter 1 will be substantiated. Finally, the chapter will close with an analysis of private GANs.

4.1 Experiment Description

All experiments were divided into two main categories: private and non-private. Non-private category used four GAN architectures: CGAN, AC-GAN, WGAN, and WGAN-GP. Only one architecture was used in the private category - DP-WGAN. Since WGAN was already studied extensively in the non-private part of the experiment, a simplified approach was taken for its private counterpart. DP-WGAN will be discussed in a separate section at the end of the chapter.

Experiment Parameters

Four GAN variants were trained on both datasets containing 10% and 20% of malicious decoys constructed according to MWA from above. Each combination of architecture, dataset,

percentage, and decoy type constitutes a *parameter set* for a single *experiment*. To produce viable statistics, each experiment was repeated *50 times*. The parameters are summarized in Table 4.1.

GAN Architecture	CGAN, AC-GAN, WGAN, WGAN-GP
Dataset	MNIST, F-MNIST
Decoy Type	Early Stop, Donwgrade
PGD Strength	0.0*, 1.0
Amount of Decoys	10%, 20%

Table 4.1: Experiment Parameters: a total of 48 combinations.
*Early Stop Only

Attack Performance

Once a GAN was trained for a set of parameters, it generated 2048 samples. In addition, another batch was created containing an equal number of randomly chosen, equivalent training samples. Equivalence here refers to maintaining label correspondence for conditional architectures.

These two batches were used to measure the FID score of the generated samples. It was done according to the methodology introduced in Section 2.4. FID score was used to quantify an attack's performance for a set of parameters from above. For clarity, a higher FID score indicates larger dissimilarity and, thus, a better attack performance.

Decoy Detectability

The final step of a single experiment is to measure the detectability of decoys. This was accomplished according to the procedure described in Section 3.5. Detections were made with a Def-GAN at various thresholds. From this, a Receiver Operating Characteristic (ROC) curve and its corresponding Area Under the Curve (AUC) scores where derived.

Summary

The steps necessary to execute a single experiment are summarized in the workflow below:

1. Create a clean GAN
2. Create decoys
3. Enhance decoys
4. Mix decoys with data
5. Create a poisoned GAN
6. Generate synthetic samples
7. Evaluate synthesis quality (FID)
8. Measure decoy detectability (AUC/ROC)

On a practical level, the experiments amounted to 48 experiment combinations $\times$ 50 trials for a total of 2400 runs. Given that each combination of a GAN and dataset required a clean GAN, an additional 400 ($4 \times 2 \times 50$) runs were necessary. In addition, each GAN is composed of at least two neural networks. Thus, a total of *5800* neural networks were trained. The experiments were conducted on a Ubuntu server with CUDA. Total runtime lasted approximately two weeks using the hardware from Table 4.2.

Type	Component	Qty
GPU	NVIDIA Quadro RTX 8000 with 48601 MB	8
CPU	AMD EPYC 7402 24-Core	2
Memory	1 TB	
Storage	2 TB	

Table 4.2: Hardware Specifications

4.2 Null Hypothesis

To evaluate whether MWA had any effect on a GAN, two null hypotheses must be rejected:

H.1 The attack had no effect on a GAN.

H.2 Comparable results can be achieved with a simple injection of random, bogus data.

To address the first hypothesis, FID scores of an attack - parameterized as described above - will be compared to the FID scores of a clean GAN. A similar approach will be taken for

the second null hypothesis. The difference is that instead of using a clean GAN, a substitute will be trained on data containing bogus samples. In this case, bogus samples will be EEDs before PGD. For simplicity, it will be referred to as a *bogus GAN*.

For an easy comparison, notch boxes are created from FID scores generated by a single combination of experiment parameters over 50 trials. If notches do not overlap, this means the difference between the two groups is statistically significant (0.05 p-value) [30]. However, to dismiss a null hypothesis, a stricter measure was taken than the standard p-value.

This difference was not sufficient to manifest visually on the generated samples during experiments. For this reason, the measure was tightened. To dismiss a null hypothesis, the entire boxes must not overlap. Moreover, such an approach guarantees statistical significance.

4.3 Attack Analysis

This section will look at the success and failures of MWA with respect to the null hypotheses. First, attack performance will be examined across families of GANs. Then, the relationship between attack performance and GAN architectures will be examined within their respective families. Next, GAN's reactions to injections of noisy data will be studied. Finally, the link between a successful attack and a compromised discriminator will be analyzed. Unless otherwise indicated, conclusions will be drawn by studying experimental results presented in Figure 4.1. For visual reference of MWA's impact on synthetic data please see Figures D.1 and D.2.

Interpreting Results

First, we need to understand how to interpret the results presented in Figure 4.1. Each row of this figure corresponds to the results for a certain GAN indicated accordingly. The four sets of boxplots on the left refer to GANs trained on MNIST and the four on the right to FMNIST. Now, let us turn to the contents of a set of boxplots. For clarity, there are eight

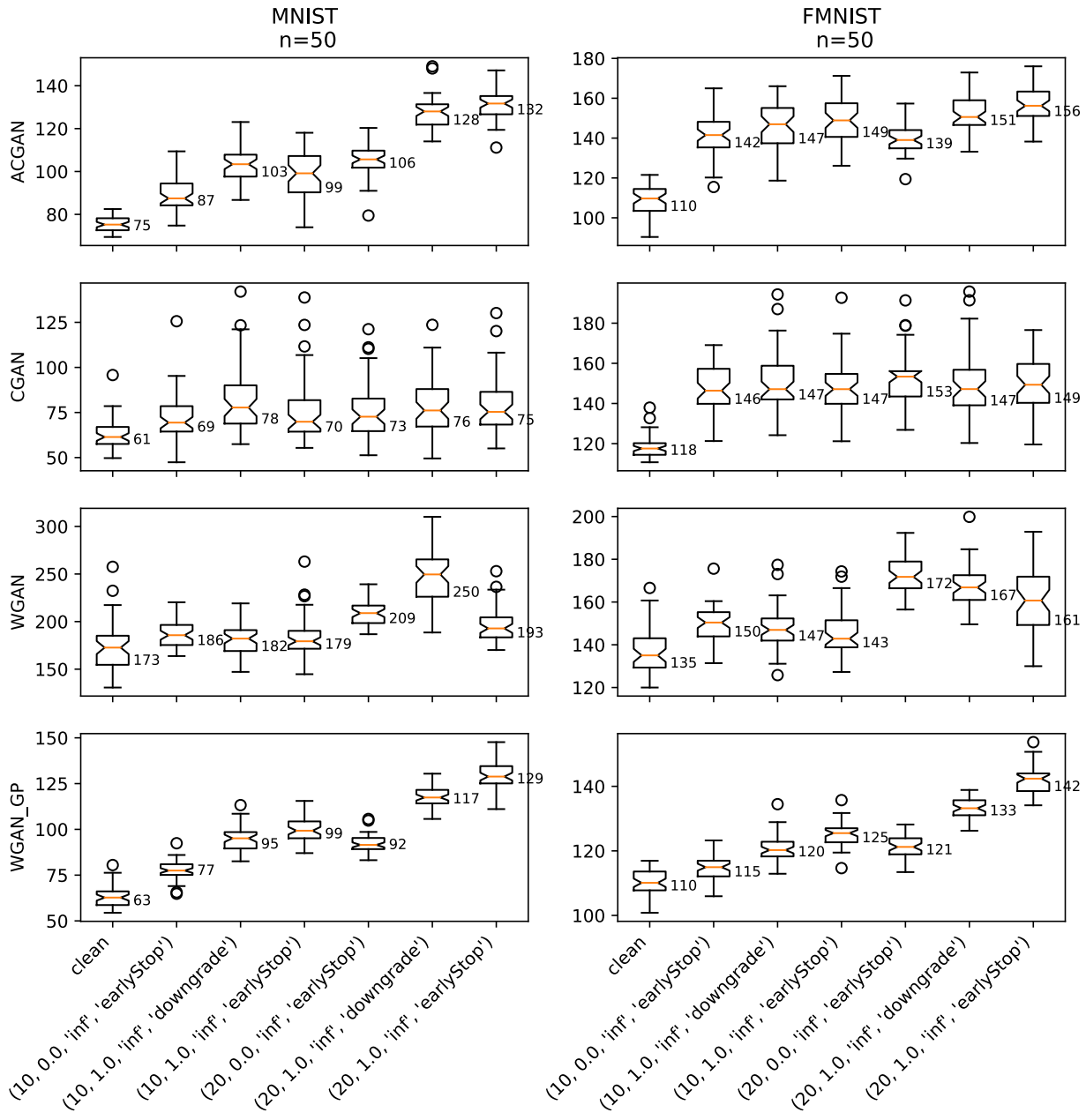


Figure 4.1: FID Scores of attacks on GANs. The tuples on the x-axis represent experiment parameters: injection size (%), PGD strength (ϵ), PGD norm, decoy type. The numbers on the boxplots indicate the medians. The columns with PGD strength of 0.0 represent the bogus GANs needed to reject **H.2**. The clean column refers to **H.1**. A higher score implies **less** similiarty.

such sets in the figure.

The first boxplot in any set corresponds to the "clean" label on the x-axis. If a boxplot with FIDs from an attack does not overlap the first boxplot, it means that the attack rejected

H.1. For example, the third boxplot in the upper-left set - AC-GAN trained on MNIST - contains FIDs of samples generated by a GAN that was trained on data with a 10% injection of DGDs. Since the third boxplot does not overlap the first, we conclude that the first null hypothesis was rejected.

To reject the second null hypothesis, we must compare the results from a given attack to the bogus GAN. The boxplots of the bogus GANs can be found in the second and fifth columns. Going back to our example, we can see that the attack boxplot does not overlap the second boxplot in the set. This means that the attack rejected the second null hypothesis. It is important to note that boxplots of attacks containing 20% of injections must not overlap boxplots of bogus GANs containing an equal or lesser amount of random bogus data. Finally, a successful attack will reject both null hypotheses.

Performance Summary

Now that we know how to interpret results in Figure 4.1 we can determine which attacks were successful. Starting with AC-GAN trained on MNIST, we can see that a 10% injection of DGDs increased the FID scores of the generated data enough to pass both null hypotheses. At 20% of injections, attacks containing both types of decoys rejected the null hypotheses.

AC-GAN trained on F-MNIST failed to reject **H.2** for all cases but one - a 20% injection of EEDs. AC-GANs trained on data containing 10% injections of both decoys clearly fail to reject **H.2**. The reason is that the boxplot of FIDs from this attack overlaps the boxplot of FIDs from the bogus GAN with a 10% injection. Less obvious is that attack containing 20% of DGDs fails **H.2**. The reason is its boxplot does not overlap a bogus GAN containing 20% of noise (fifth column), but it does overlap a bogus GAN containing 10% of noise (second column). EEDs clear the above hurdle.

Turning to CGAN, we can see that the attack failed for all cases. For example, a boxplot made of FID scores from samples produced by CGAN that was trained on MNIST containing a 10% injection of decoys (both types) overlaps the boxplot of the bogus GAN with 10% of

noise. That means this attack failed to reject **H.2**. The same happens for all other attacks on CGAN.

An attack on WGAN was successful only once. WGAN trained on MNIST containing 20% of DGDs produced FID scores whose boxplot cleared both null hypotheses. For all other cases, the attacks failed on WGAN.

WGAN-GP proved to be most vulnerable. All attacks on this GAN rejected both null hypotheses. We can see that boxplots of FIDs for all attack combinations do not overlap neither the FIDs of the clean boxplot nor the boxplot corresponding to a bogus GAN. Thus, **H.1** and **H.2** were rejected respectively.

In total, out of the 32 attack combinations, 13 passed both null hypotheses and can be considered successful. In the rest of the cases, the attack is considered failed. The results are summarized in Table 4.3.

GAN	ES 10%	ES 20%	DG 10%	DG 20%
AC-GAN		M/F	M	M
WGAN				M
WGAN-GP	M/F	M/F	M/F	M/F

Table 4.3: Cases where both null hypothesis were rejected. ES - Early Stop Decoys, DG - Downgrade Decoys, M - MNIST, F - FMNIST.

GAN Families

The first observation is that the Wasserstein family of GANs is more vulnerable to both early epoch and downgrade variations of the attack. For the Wasserstein family, EEDs succeeded four times, while for the conditional family, they succeeded twice. Similarly, for DGDs, the attack succeeded five times with the Wasserstein family and only twice with conditional. In total, the Wasserstein family was vulnerable nine times, while the conditional - four times. As was noted in Chapter 2, Wasserstein GANs use a more sophisticated measure to construct the loss function. This supports the part of finding **F.1** that relates loss function complexity to attack success.

Architecture Dependence

My next observation is that success varies within each family. In the Wasserstein family, the WGAN-GP architecture was successfully attacked eight times, while the WGAN only once. In addition, WGAN saw zero successes of the EEDs, while WGAN-GP produced four. Likewise, in the conditional family, the AC-GAN architecture saw four successes, while the CGAN - zero. Of the four successes, two were for EEDs.

Again, as noted in Chapter 2, AC-GAN is a more advanced version of CGAN and WGAN-GP of WGAN. In both cases, advancement stems from improvements of the loss functions. Also, in both cases, these improvements made the loss functions more complex. This further supports the claim in finding **F.1** that attack success improves with loss function complexity.

Data Dependence

Target data can be a significant factor in the success of the attack. The attack succeeded eight times on MNIST and five on F-MNIST. However, of the five F-MNIST successes, three were EEDs. Moreover, EEDs succeeded consistently across datasets, while downgrades succeeded more often on MNIST. This implies that the claim in finding **F.1** relating the improved attack's success to dataset complexity is correct.

Noisy Data

GANs show a tendency to recover from injections of noise. For CGAN trained on MNIST, the performance of GANs trained on datasets containing both 10% and 20% of bogus data is the same as GANs trained on clean data. The same here means that the bogus GANs fail **H.1**. The fact that this GAN could recover from such large amounts of noisy data is surprising! Similarly, WGAN trained on MNIST with 10% of injections, and WGAN-GP trained on F-MNIST with 20% also recovered. Likely, the high number of iterations induced by the expectation of the objective function averages out the noise. This supports observation **O.1**.

Discriminator

As was shown in Section 2.7, many works in this field focused on making the discriminator more robust by altering the algorithm. However, as was discussed earlier, such invasive approach may not be warranted. My results show that when the number of modified samples is not higher than 20%, a compromised discriminator does not always signify a compromised GAN. Although counter-intuitive, it may be a better strategy to defend the GAN as a whole and not its weakest component - the discriminator.

While the attack succeeded in less than half of the cases, the discriminators were compromised in all cases. This can be seen from Figures D.4, D.5, D.6, and D.7. Clearly, these figures show a result from only one iteration. Thus, one might be looking at a compromised discriminator of an outlier, while the median discriminator is not showing signs of being compromised. For this reason, discriminators from multiple iterations were analyzed, and all were compromised! Thus, a compromised discriminator may not be sufficient to compromise a GAN, as stated in observation **O.2**.

4.4 Detection Analysis

Having established the conditions for a successful attack, it is important to establish whether it is possible to prevent it. The defensive strategy chosen here is based on filtering out malicious samples from the training data. The precise mechanics of this process were described in Section 3.5. AUC scores for detection results can be seen in Figure 4.2.

A general observation is that adding adversarial perturbations from PGD significantly raises the detectability of samples. This can be seen across architectures, decoy types and datasets. The one exception is AC-GAN. If for other architectures the difference is approximately 0.3-0.5, then for AC-GAN, the difference is approximately 0.2 and 0.1 for MNIST and F-MNIST, respectively. The difference is still statistically significant but much smaller than for other architectures.

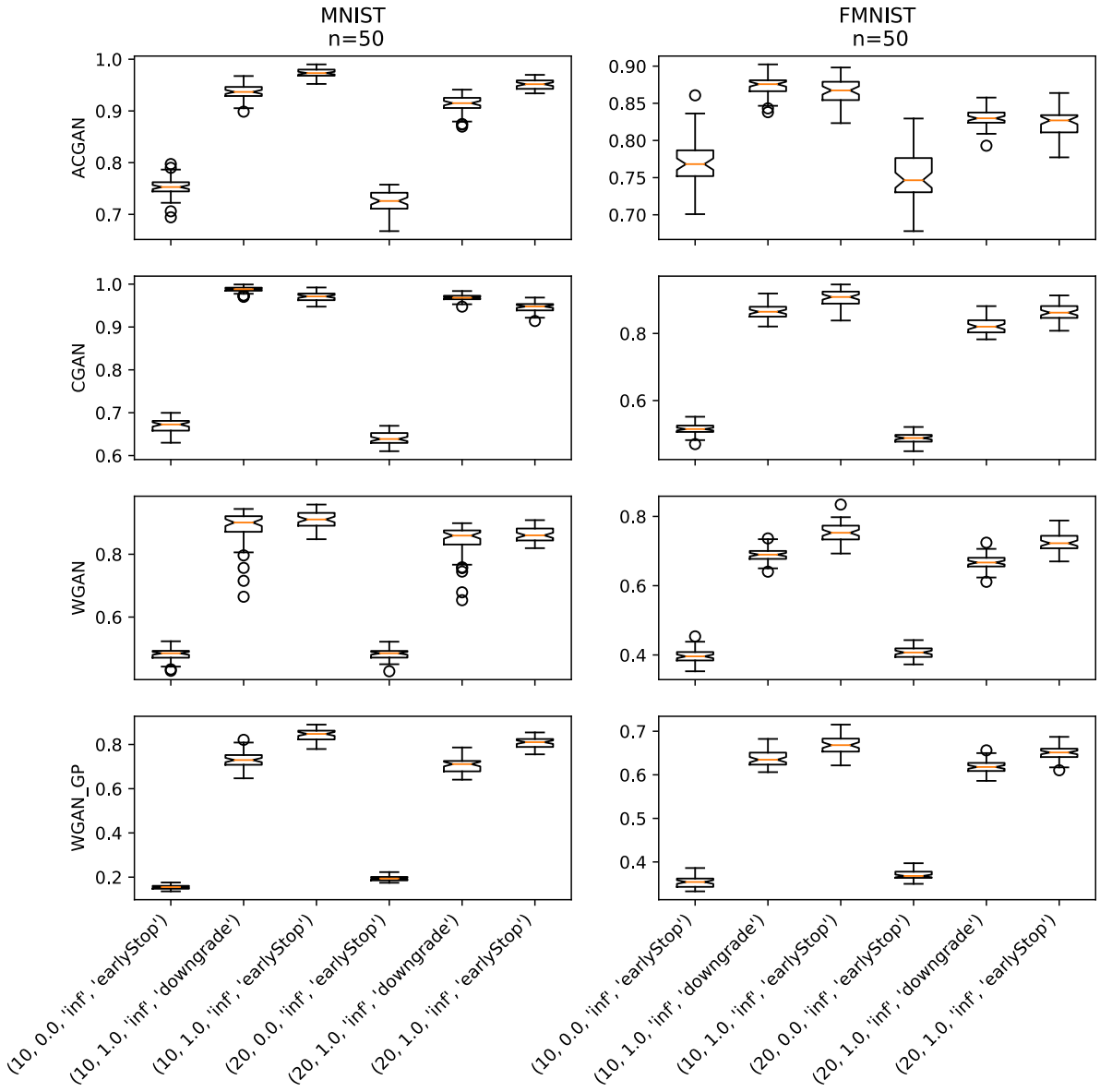


Figure 4.2: AUC Scores of attacks on GANs. The tuples on the x-axis represent the injection size (%), PGD strength (ϵ), PGD norm and decoy type.

What remains unclear is whether the AUC increase will translate into better detection. From the table, we can see that AUCs are above 0.8 for most cases. The exception is the Wasserstein family, especially when trained on F-MNIST. Hypothetically, to evade detection, the AUC should be below 0.8 and the difference with the bogus GAN approximately 0.3. However, ROC curves will be analyzed below to further examine detection.

Another general observation is that for most cases EEDs have comparable AUCs to

their downgrade counterparts. Even though the difference is large enough to be considered statistically significant in most cases, in reality, the difference can often be considered small. The significance of the size of AUC depends on the underlying domain. In the context of this work, a less than 0.05 difference is small, while $0.05 - 0.1$ is big.

From Table 4.4 we can see that in most cases, the difference between medians is less than 0.05. The exceptions are WGAN trained on F-MNIST and WGAN-GP trained on MNIST. However, it is significant that the difference in AUCs is often big in cases that pass the null hypotheses. Thus, it is not possible to conclude that decoys are the same in terms of detection, and further analysis will be provided below.

GAN	Dataset	AUCs (DG - ES)
AC-GAN	FMNIST	0.0
AC-GAN	MNIST	-0.04
CGAN	FMNIST	-0.04
CGAN	MNIST	0.02
WGAN	FMNIST	**-0.05**
WGAN	MNIST	0.0
WGAN-GP	FMNIST	-0.03
WGAN-GP	MNIST	**-0.1**

Table 4.4: Differences between median AUCs of the two decoys. ES - Early Stop Decoys, DG - Downgrade Decoys.

ROC Curve Analysis

Analysis of AUC scores allowed us to draw general conclusions. However, to paint a precise picture of detectability, we should look at the ROC curves. Curves were made only for cases that rejected both null hypotheses. They can be seen in Figure 4.3.

An important observation is that ROC curves in Figure 4.3 come from a single iteration and not the average. This may be a serious problem if we are analyzing a curve that is not representative of its particular case. However, notch boxes in Figure 4.2 show that AUCs have a tight spread around the median. Moreover, the figure with the ROC curves indicates an AUC of each curve. Thus, its deviation from the median can be established.

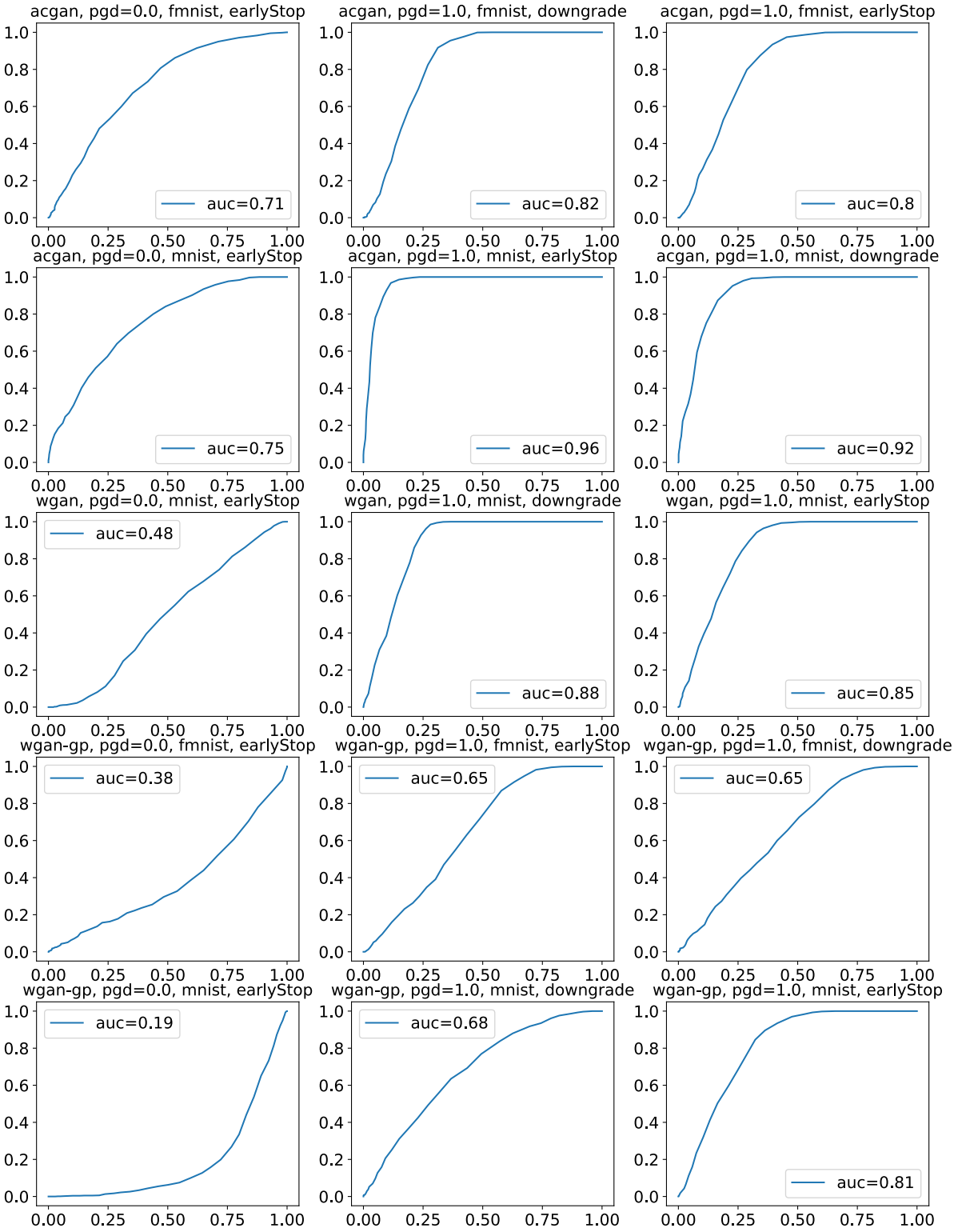


Figure 4.3: ROC Curves. Labels refer to GAN type, PGD strength, dataset and decoy type. Bogus GANs have PGD strength of 0.0 (left column).

ROC curves plot the relationship between detected malicious samples and false alarms. The technical term for the first one is True Positive Rate (TPR) and False Positive Rate (FPR) for the second one. In our case, FPR implies the portion of the dataset that must be abandoned to eliminate a number of decoys that stems from a corresponding TPR.

It is impossible to establish a commonly acceptable FPR because it depends on the case. For example, in some situations, eliminating 10% of the data is unacceptable, while 50% might be dispensable in others. However, to proceed with the analysis, I will consider a 20-30% sacrifice of the dataset to be the threshold of an acceptable loss of data.

Another important criterion to establish is the fail condition of the detection. *The countermeasure fails if the performance of a filtered attack rejects **H.2** and **H.1** as before.* However, since the noise in **H.2** can also be removed, for a fair comparison, it will be filtered using the same technique as the normal decoys.

Starting with AC-GAN, we can see that at 20-30% FPR, a TPR of approximately 80% is achieved for the decoys on F-MNIST (approx. 100% on MNIST). This means that 80% of decoys will be eliminated on F-MNIST. However, a TPR of approximately 60% was achieved for the GAN trained on data containing bogus samples. Bogus GANs are marked in the figure by showing PGD with a strength of 0.0.

WGAN trained on MNIST also achieved a high TPR. For both decoys, the rate is above 80% within the acceptable FPR threshold. However, TPR for the bogus GAN is much lower at 20%. This indicates that the filtered result will likely fail **H.2**.

WGAN-GP trained on MNIST demonstrated a TPR of approximately 60% and 80% for downgrades and EEDs, respectively. On the other hand, the TPR for the bogus GAN is extremely low at less than 10%. This implies that **H.2** will be difficult to overcome in this case.

Finally, WGAN-GP trained on F-MNIST demonstrated a TPR of approximately 40% for both decoys. The TPR for the bogus GAN is low again at around 20%. In this case, **H.2** is likely to be rejected. Now, let us look at the FID scores adjusted for these TPRs.

To visualize the effect of the countermeasure on the attack, a figure similar to Figure 4.1 will be created. However, this figure will contain FIDs that we would have received if we filtered the data before training the GAN. These numbers were simulated as follows.

Re-training the GANs on smaller numbers of decoys was computationally prohibitive. For this reason, an approximation was made. FID scores in Figure 4.1 exhibited approximately a linear relationship to the amount of decoys. This applies to both types of decoys and bogus data. Thus, the new FID scores in Figure 4.4 were simulated by exploiting this linear relationship. FID scores were linearly projected to lower size injections. New sizes were determined using TPRs averaged over iterations.

Starting with AC-GAN trained on MNIST, attacks based on both types of decoys now fail both null hypotheses. Attacks on AC-GAN trained on F-MNIST also failed to reject **H.2** after the countermeasure. Similarly, CGAN now fails both null hypotheses on MNIST and **H.2** on F-MNIST.

Finally, the situation with WGAN is similar. Before the countermeasure, it rejected the null hypotheses only for the case when trained on MNIST with 20% downgrade decoys. After the defence, this case and the others are well below passing both null hypotheses on MNIST. *Thus, AC-GAN, CGAN and WGAN were successfully defended.*

For WGAN-GP, the result was different. The attack failed on MNIST but succeeded on F-MNIST. For MNIST attacks containing both types of decoys clearly fail **H.2**. The same applies to F-MNIST when using 10% of DGDs. However, when using EEDs or a higher number of DGDs, the attack rejects both null hypotheses. *Therefore, the attack is successful for WGAN-GP on F-MNIST at 10% for EEDs and at 20% for both types of decoys.*

Two implications stem from the above. First, the attack survived the countermeasure only for the most advanced GAN [36] and on the more sophisticated dataset. More, EEDs survived twice, while downgrades only once. This proves finding **F.2** that my proposed attack is likely to evade detection for more complex loss functions and datasets.

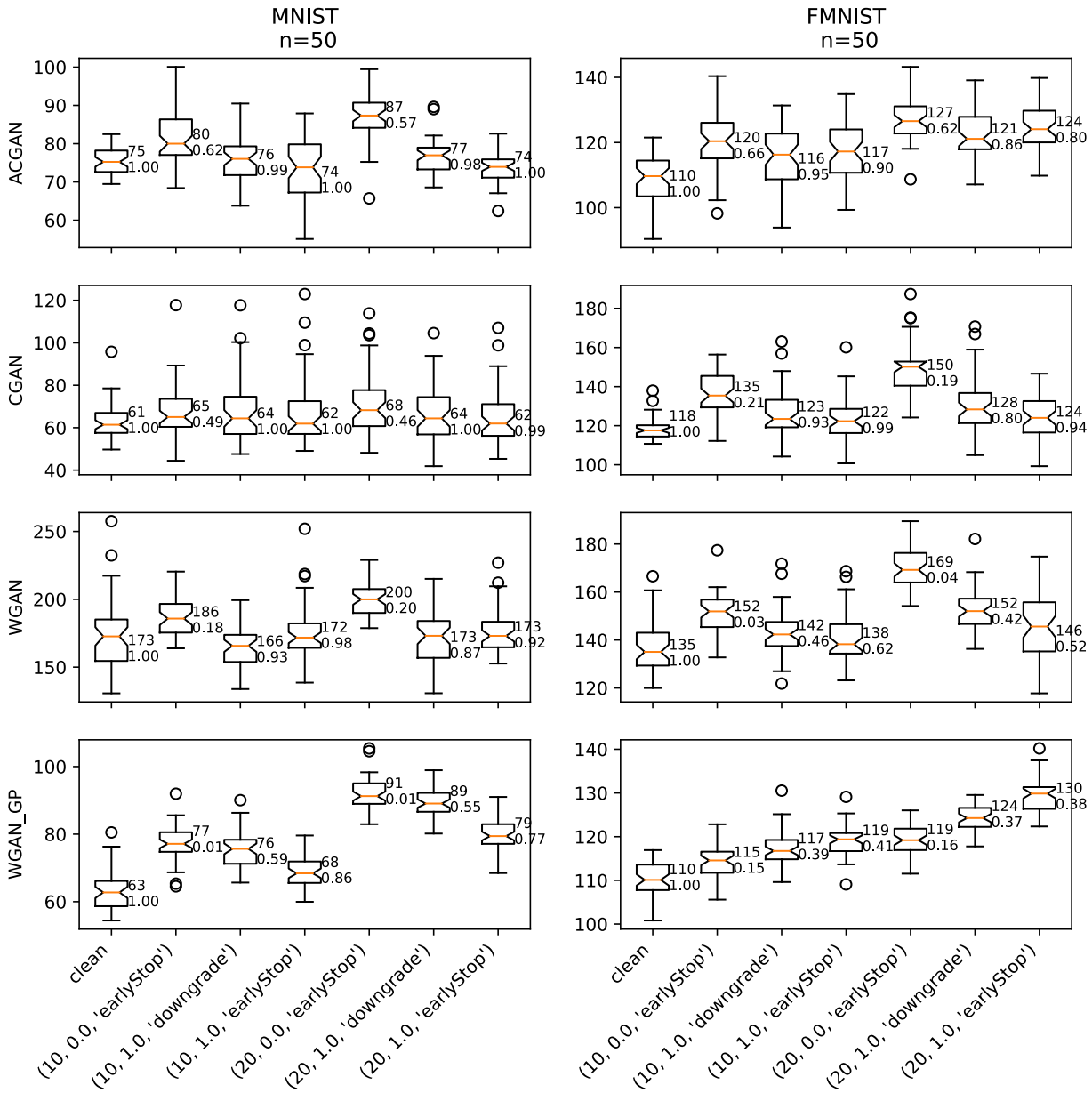


Figure 4.4: FID Scores After Filtering. The tuples on the x-axis represent the injection size (%), PGD strength (ϵ), PGD norm and decoy type. The numbers on the boxplots indicate the medians and the TPRs. The columns with PGD strength of 0.0 represent the bogus GANs needed to reject **H.2**. The clean column refers to **H.1**. A higher score implies **less** similiarty.

The Size of The Wrench

As we saw, most attacks on GANs employed herein are preventable. However, this is not true for WGAN-GP when the dataset is F-MNIST. More precisely, this attack defeated

the defences when 20% of the training data was malicious, and at 10% it succeeded only for EEDs. Judging from FID dynamics in Figure 4.4, 10% is the lower bound for EEDs. For downgrades, the lower bound is somewhere in the 10-20% range. Together, these facts support finding **F.3**.

4.5 Decoy Comparison

Regardless of whether the underlying data is tabular or visual, the reader should establish a sense of what the malicious samples are like. In the case of tabular, one would examine the features of malicious tuples. With sufficient domain knowledge, one can establish a sense of the distance these samples have from the underlying data. Moreover, by analyzing these samples, an insight can be gained into possible avenues for attack or defence. Again, assuming a sufficient degree of expert knowledge in the data or security domains.

In the case of visual data, the situation is similar. By inspecting visual patterns exploited by the attack, one can deduce possible improvement strategies. Alternatively, expert knowledge in the security domain can be applied on a higher level. Strategies for defending communication channels, data collection practices, pre-processing and more can be inspired.

For these reasons, a complete set of malicious samples will be presented. EEDs before the application of PGD are given in Figure C.1. The same decoys after PGD can be seen in Figure C.2. For the downgrade version of decoys please refer to Figure C.3.

Human Observer

Now let us consider visual inspection as a defensive tool. As was mentioned, MWA with downgrade decoys corresponds to efforts from earlier literature. However, when closely examined, these decoys resemble original data very closely (ref. Fig. C.3). Individual MNIST digits can be discerned and recognized. Similarly, with clothing items. The main difference is the presence of a strange noise.

A seasoned security professional would realize this noise is malicious. Adversarial perturbations are not a new phenomenon since this discussion started in 2014 at the latest [23]. Moreover, the noise always follows a similar pattern. This indicates a lack of randomness and a possible presence of intent.

The situation with EEDs is different. Often the original data cannot be discerned at all (ref. Fig. Figure C.2). In some cases, the decoys appear as random noise. In others, they are just strange blobs. Moreover, by tweaking these decoys, they can be made even more stealthy. Thus, I conclude that it is likely possible to conceal EEDs from a human observer. This supports finding **F.4** which states that it is likely possible to visually conceal an attack from a human operator.

However, a separate study is required to formally prove the degree of concealment. Such a study could work like the following. First, decoys must be crafted to maximize concealment. Then, security experts should be asked to recognize the presence of malicious intent. From this, a formal conclusion can be drawn.

Decoy Performance

Having analyzed the qualitative difference between decoys, it is time to look at the quantitative side. For the purpose of this comparison, only cases that pass both null hypotheses will be examined. In addition, if only one type of decoy passed the test, then its other equivalent will be included. From Figure 4.5 we can see that EEDs outperformed downgrades in all cases, but two: (1) AC-GAN trained on MNIST with 10% decoys, and (2) WGAN trained on MNIST with a 20% injection.

It is possible to conclude that EEDs outperform downgrades quantitatively with standard statistical significance [30]. But is statistical significance enough to claim an improvement in this domain? When EEDs outperform downgrades, they do so with a FID increase of 5% on average. Which is the first argument to support finding **F.1** that existing attack proposals were improved upon.

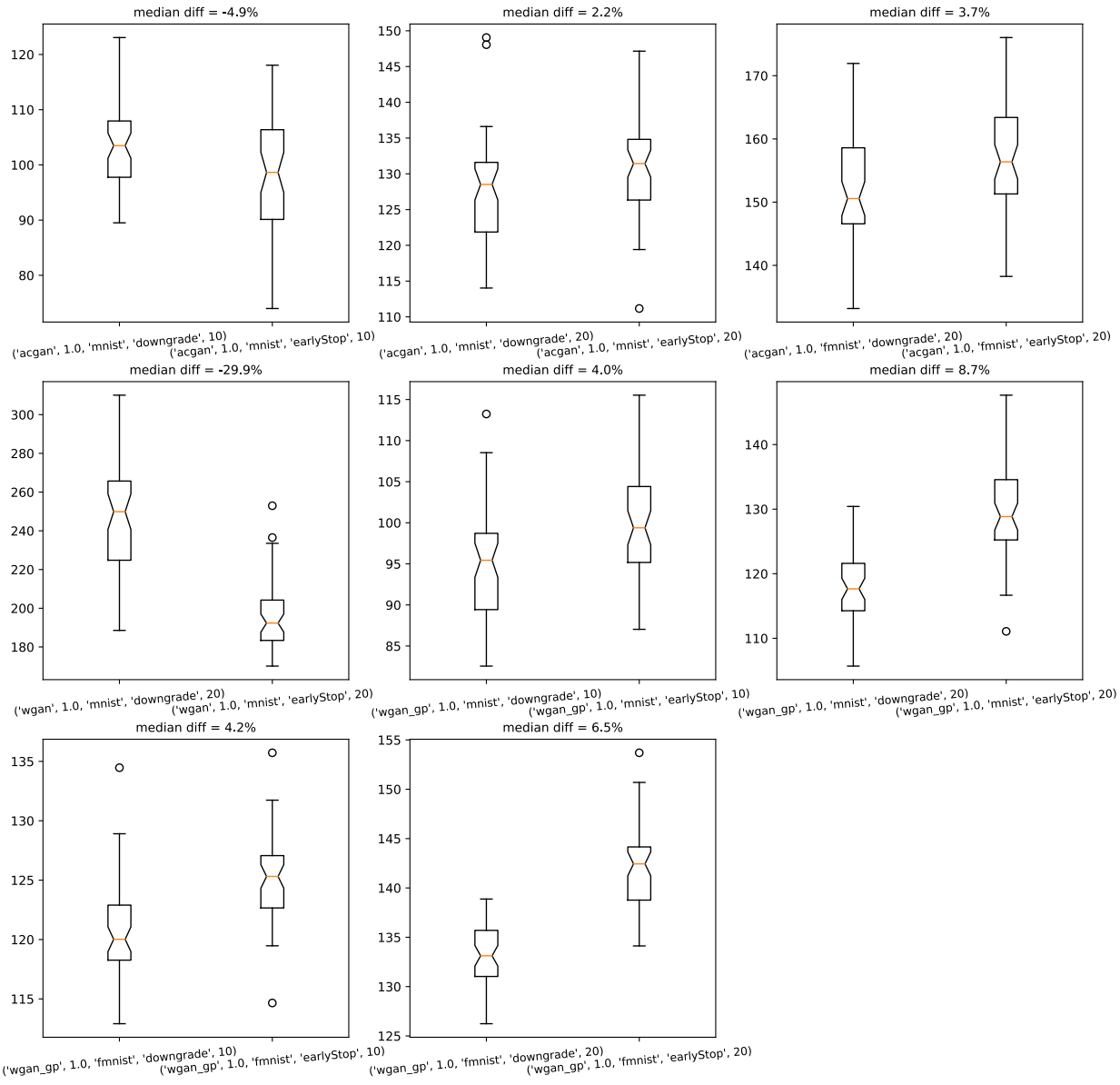


Figure 4.5: Decoy Comparison. Labels on the x-axis refer to: GAN type, PGD strength, dataset, decoy type, percentage of decoys.

Now, let us consider decoy performance after applying the countermeasure. In this case, the attack is successful only for WGAN-GP trained on F-MNIST with a 10% and 20% injection. In the 20% case, the bogus GAN used as a basis for **H.2** had a median FID of 119. EEDs and DGDs had FIDs of 130 and 124, respectively. The first one was 9% higher and the second one was 4% higher than the **H.2** median. In the 10% case, only EEDs succeeded and outperformed **H.2** with 3.5%.

This tells us that the early epoch approach outperforms the downgrades when the attack rejects the null hypotheses and evades detection. Moreover, this happens with standard statistical significance. Noting that the downgrade approach is an existing scientific proposal, the result forms another argument towards finding **F.2** that an improvement was achieved. However, in both cases, improvement is true only for complex loss functions and datasets.

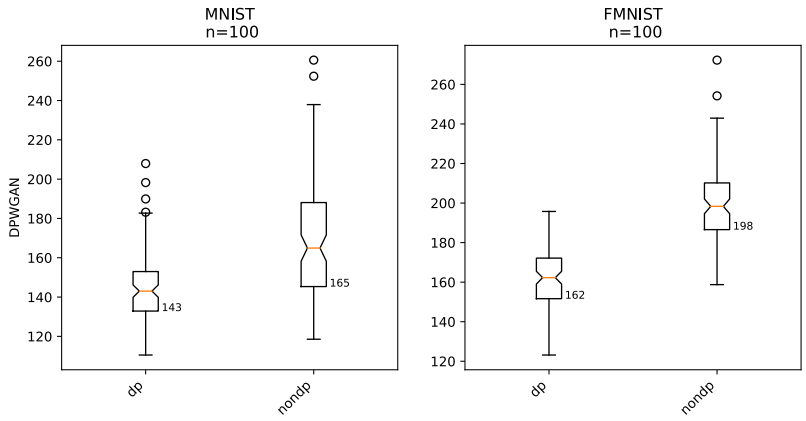
4.6 Private GAN

As mentioned earlier, a simplified approach was taken in this part. The synthesis quality of samples generated by a poisoned DP-WGAN was not measured. Since this was tested extensively for the non-private counterpart, it is reasonable to assume that it will fail with the private one. However, it is unclear if the countermeasure will detect malicious samples at a similar rate.

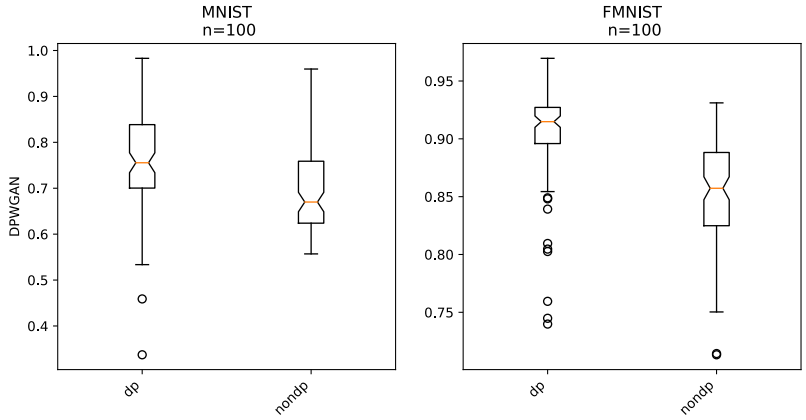
DP-WGAN adds specially crafted noise to the gradients to protect the privacy of the training set. As mentioned in Section 2.1, adding noise to gradients may act as a form of regularization or stimulate posterior sampling. The noise may improve the generalization of the discriminator. Moreover, unlike PGD, with Def-GAN we are not probing the discriminator directly. Thus, there is a good chance that improved generalization may produce a different detection outcome versus the non-private case.

First, the clean synthetic data generated by a private and a non-private GAN was compared. When anonymization noise is added, the GAN performs better. This can be seen in Figure 4.6a. For a visual reference please see Figure D.3. In Figure 4.6a FID is lower with statistical significance for the DP version. It is probable that, as hypothesized above, the anonymization noise improved GAN's performance.

Second, MWA on the private GAN was tested with 10% of EEDs for a response to the countermeasure. The same pattern holds for detection as for generation. As we can see from Figure 4.6b, the private version outperforms in terms of detection with statistical



(a) FID Scores of clean private and non-private GANs



(b) AUC Scores of private and non-private attack with 10% EEDs

Figure 4.6: DP-WGAN Results.

significance. Moreover, both the first and the second experiments demonstrate that the DP version performs better on F-MNIST, which is a more sophisticated dataset. The implication is that finding **F.5** is correct and private GANs are likely more resilient to an MWA-type attack.

Good response of private GANs to the countermeasure is important. Unlike countermeasures covered in Section 2.7, the one used in this research does not require modifying the algorithms. The earlier approach usually entails additional computation of original data. For differentially private algorithms, any additional computation on data consumes the privacy budget.

Since GANs are complex algorithms, the privacy budget is already thin [55]. For example,

a differentially private WGAN proposed by Xie *et al.* [55] requires a double-digit budget to achieve acceptable results. This leaves little room for additional computation without increasing the privacy risks beyond the threshold of utility.

For the above reason, it is safe to assume that a computationally-heavy defence strategy will likely render a private GAN non-functional. Thus, the observed resilience gives hope that existing peculiarities of private GANs can be leveraged to defend them. Thus, private GANs may be possible to defend using current techniques, which supports finding **F.5**.

Chapter 5

Conclusions and Future Work

This thesis aimed to explore the vulnerability of GANs to adversarial input attacks. Such attacks work by injecting malicious samples into the training data to hinder a GAN's convergence. Previous works looked at robustifying individual components of GANs or defending against very specific threats. However, a comprehensive study of attack impact on synthetic data was lacking. Specifically, it was unclear what conditions are necessary for such an attack to succeed and whether it is possible to defend against.

An attack was devised based on a known approach to answer these questions. As discussed in Section 2.7, a common way to attack a GAN is to apply a first-order attack for neural networks on a subset of data and aimed at the discriminator component of a GAN. This would make the discriminator mislabel this subset as fake and the minimax game would be derailed because of wrong signals arriving to the generator. In this work, such subset is referred to as DGDs.

The above approach was improved by using different source data for crafting adversarial samples. Instead of directly using a subset of training data, samples generated by a GAN that trained for only an epoch were used. The hope was that by nudging the discriminator with PGD to rank these samples highly on the realness scale, the generator would treat aborted convergence as a goal. Empirical results showed that this is indeed a superior approach.

Adversarial samples described in this paragraph are referred to in this thesis as EEDs.

However, the attack proposed in this thesis outperformed only in specific conditions. The gains were made on the more sophisticated dataset or GANs with more advanced loss functions. A similar pattern held when trying to break through a countermeasure.

In general, the attacks overcame the defence only on the above dataset and GAN architecture. However, my version succeeded for injections of 10% and 20%, while the baseline succeeded only with 20%. In both cases, my version achieved a higher FID score, which indicates better performance.

Above injection thresholds led me to conclude that 10-20% is the lower bound of injection size. For EEDs, 10% was the lower bound. DGDs have a lower bound somewhere in the 10-20% range. Thus, EEDs require a smaller injection. However, when a countermeasure was not applied, both attacks were successful more often, with EEDs performing better. Finally, the adversarial samples proposed in this thesis are more difficult to identify visually.

This thesis concluded with the implications of the results for private versions of GANs. The one tested here proved to be more resilient regarding the countermeasure than its non-private counterpart. Such countermeasure is likely the only viable option because it does not require additional computation on original data. On the other hand, other strategies almost always entail additional computation. The problem is that computation on the original data depletes the privacy budget of private GANs. Given that the budgets are already thin for private GANs, any additional computation would likely make them impossible to converge. From this perspective, the results in this thesis imply that in the case of private GANs, it may be easier to defend against MWA and similar types of attacks.

5.1 Future Work

The proposed here approach to crafting adversarial samples performed better than the ones proposed in the literature. In addition to the quantitative performance, these samples demon-

strated a qualitative improvement. Specifically, they are harder to detect for a human operator. However, there is much to explore in terms of amplifying this advantage. In addition, a formal study focused on concealment will help quantify this qualitative difference. A sketch of a possible approach was given in Section 4.5.

Another open question is the countermeasure. The one used in this thesis is based on filtering out the samples. However, other approaches were proposed that may perform better. Usually, the latter entails additional costs. Given that some GANs are defenceless using the current countermeasure, it is worth investigating the cost-benefit equilibrium of other approaches.

Finally, the results here show that it is possible that private versions of GANs can be defended because they are likely more responsive to a countermeasure and more resilient to an attack. However, this result needs further validation on more sophisticated GAN architectures and a wider range of privacy budgets. Unfortunately, only a few versions of GANs were made private [20]. Moreover, these versions are the less advanced architectures. However, it still remains interesting to see whether they can be defended against MWA. This warrants a thorough investigation of the security of private GANs.

Appendix A

Algorithms

Algorithm 1 Generative Adversarial Network. Source: Goodfellow *et al.* [24]

for number of training iterations **do**
  for number of discriminator iterations **do**
    Sample minibatch of noise samples $z^{(1)}, \ldots, z^{(m)}$.
    Sample minibatch of data samples $x^{(1)}, \ldots, x^{(m)}$.
    Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(x^{(i)}) + \log \left(1 - D(G(z^{(i)}))\right) \right]$$

  end for
  Sample minibatch of noise samples $z^{(1)}, \ldots, z^{(m)}$.
  Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D(G(z^{(i)}))\right)$$

end for

Algorithm 2 Wasserstein Generative Adversarial Network. Source: Arjovsky *et al.* [4]

Require: α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, number of critic iterations per generator iteration.

Require: w_0, initial critic parameters. θ_0, initial generator parameters.

while θ has not converged **do**
  for n_{critic} iterations **do**
    Sample minibatch of noise samples $z^{(1)}, \ldots, z^{(m)}$.
    Sample minibatch of data samples $x^{(1)}, \ldots, x^{(m)}$.
    $g_w \leftarrow \nabla_w \frac{1}{m} \sum_{i=1}^{m} \left[f_w(x^{(i)}) - f_w(g_\theta(z^{(i)})) \right]$
    $w \leftarrow w + \alpha \cdot RMSProp(w, g_w)$
    $w \leftarrow clip(w, -c, c)$
  end for
  Sample minibatch of noise samples $z^{(1)}, \ldots, z^{(m)}$.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot RMSProp(\theta, g_\theta)$
end while

Algorithm 3 Differentially Private Wasserstein Generative Adversarial Network. Source: Xie *et al.* [55]

Require: α_c, the learning rate of the critic. α_g, the learning rate of the generator. c_p, the clipping parameter. m, the batch size. n_{critic}, number of critic iterations. $n_{generator}$, number of generator iterations. σ_n, noise scale. c_g, bound on the gradient of the Wasserstein distance with respect to weights.

Require: w_0, initial critic parameters. θ_0, initial generator parameters.

for $n_{generator}$ iterations **do**
  for n_{critic} iterations **do**
    Sample minibatch of noise samples $z^{(1)}, \ldots, z^{(m)}$.
    Sample minibatch of data samples $x^{(1)}, \ldots, x^{(m)}$.
    $g_w \leftarrow \frac{1}{m} \left[\sum_{i=1}^{m} \nabla_w \left[f_w(x^{(i)}) - f_w(g_\theta(z^{(i)})) \right] + N(0, \sigma_n^2 c_g^2 I) \right]$
    $w \leftarrow w + \alpha_c \cdot RMSProp(w, g_w)$
    $w \leftarrow clip(w, -c_p, c_p)$
  end for
  Sample minibatch of noise samples $z^{(1)}, \ldots, z^{(m)}$.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha_g \cdot RMSProp(\theta, g_\theta)$
end for

Algorithm 4 Wasserstein Generative Adversarial Network with Gradient Penalty. Source: Gulrajani *et al.* [26]

Require: Gradient penalty coefficient λ, number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2

Require: w_0, initial critic parameters. θ_0, initial generator parameters.

while θ has not converged **do**
  for n_{critic} iterations **do**
    for $i = 1, \ldots, m$ **do**
      Sample real data point $\boldsymbol{x}$, latent variable $\boldsymbol{z}$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\nabla_{\hat{\boldsymbol{x}}}||D_w(\hat{\boldsymbol{x}})||_2 - 1)^2$
    end for
    $w \leftarrow Adam\left(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2\right)$
  end for
  Sample minibatch of latent variables $\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}$.
  $\theta \leftarrow Adam\left(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z}^{(i)})), \theta, \alpha, \beta_1, \beta_2\right)$
end while

Appendix B

Neural Netowork Architectures

Architecture representation follows modelling conventions and nomenclature developed and used by PyTorch[1].

B.1 WGAN

```
Discriminator(
  (model): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2, inplace=True)
    (4): Linear(in_features=256, out_features=1, bias=True)
  )
)
Generator(
  (model): Sequential(
    (0): Linear(in_features=100, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=256, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Linear(in_features=512, out_features=1024, bias=True)
    (9): BatchNorm1d(1024, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Linear(in_features=1024, out_features=784, bias=True)
    (12): Tanh()
  )
)
```

[1] https://pytorch.org/

B.2 WGAN-GP

```
Discriminator(
  (model): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2, inplace=True)
    (4): Linear(in_features=256, out_features=1, bias=True)
  )
)
Generator(
  (model): Sequential(
    (0): Linear(in_features=100, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=256, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Linear(in_features=512, out_features=1024, bias=True)
    (9): BatchNorm1d(1024, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Linear(in_features=1024, out_features=784, bias=True)
    (12): Tanh()
  )
)
```

B.3 CGAN

```
Discriminator(
  (label_embedding): Embedding(10, 10)
  (model): Sequential(
    (0): Linear(in_features=794, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): Dropout(p=0.4, inplace=False)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=512, out_features=512, bias=True)
    (6): Dropout(p=0.4, inplace=False)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Linear(in_features=512, out_features=1, bias=True)
  )
)
Generator(
  (label_emb): Embedding(10, 10)
  (model): Sequential(
    (0): Linear(in_features=110, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=256, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Linear(in_features=512, out_features=1024, bias=True)
    (9): BatchNorm1d(1024, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Linear(in_features=1024, out_features=784, bias=True)
    (12): Tanh()
  )
)
```

B.4 AC-GAN

```
Discriminator(
  (conv_blocks): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Dropout2d(p=0.25, inplace=False)
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Dropout2d(p=0.25, inplace=False)
    (6): BatchNorm2d(32, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): Dropout2d(p=0.25, inplace=False)
    (10): BatchNorm2d(64, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (11): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (12): LeakyReLU(negative_slope=0.2, inplace=True)
    (13): Dropout2d(p=0.25, inplace=False)
    (14): BatchNorm2d(128, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
  )
  (adv_layer): Sequential(
    (0): Linear(in_features=512, out_features=1, bias=True)
    (1): Sigmoid()
  )
  (aux_layer): Sequential(
    (0): Linear(in_features=512, out_features=10, bias=True)
    (1): Softmax(dim=1)
  )
)
Generator(
  (label_emb): Embedding(10, 100)
  (l1): Sequential(
    (0): Linear(in_features=100, out_features=8192, bias=True)
  )
  (conv_blocks): Sequential(
    (0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (1): Upsample(scale_factor=2.0, mode=nearest)
    (2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): BatchNorm2d(128, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Upsample(scale_factor=2.0, mode=nearest)
    (6): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): BatchNorm2d(64, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.2, inplace=True)
    (9): Conv2d(64, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): Tanh()
  )
)
```

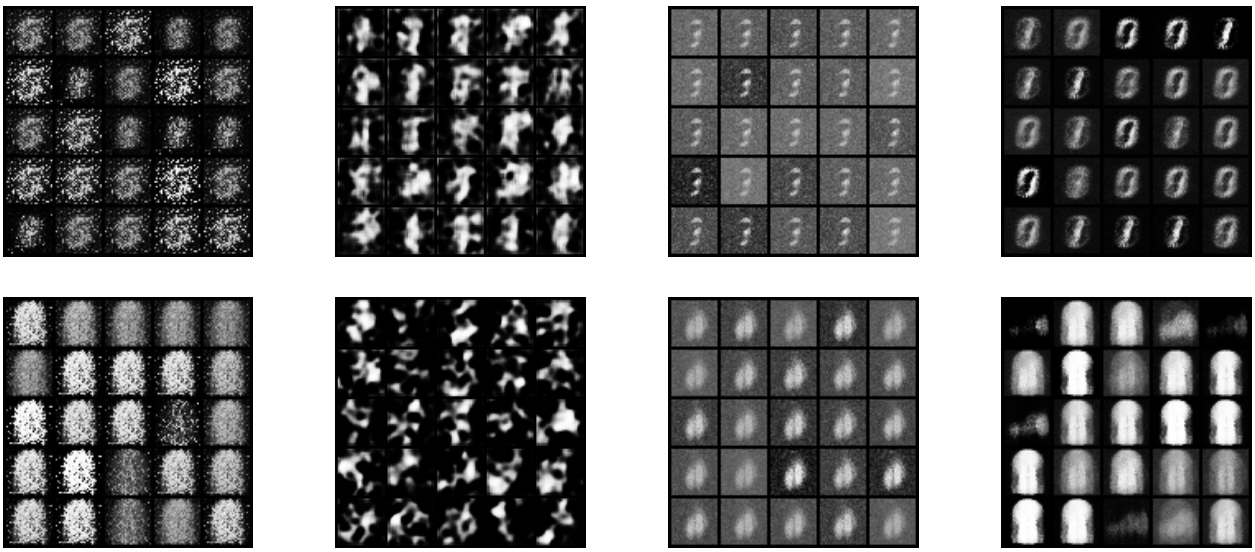
B.5 DP-WGAN

```
Discriminator(
  (model): Sequential(
    (0): Linear(in_features=784, out_features=64, bias=False)
    (1): Sigmoid()
    (2): Linear(in_features=64, out_features=1, bias=True)
  )
)
Generator(
  (model): Sequential(
    (0): Linear(in_features=100, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
```

```
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Linear(in_features=256, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Linear(in_features=512, out_features=1024, bias=True)
    (9): BatchNorm1d(1024, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)
    (11): Linear(in_features=1024, out_features=784, bias=True)
    (12): Tanh()
  )
)
```

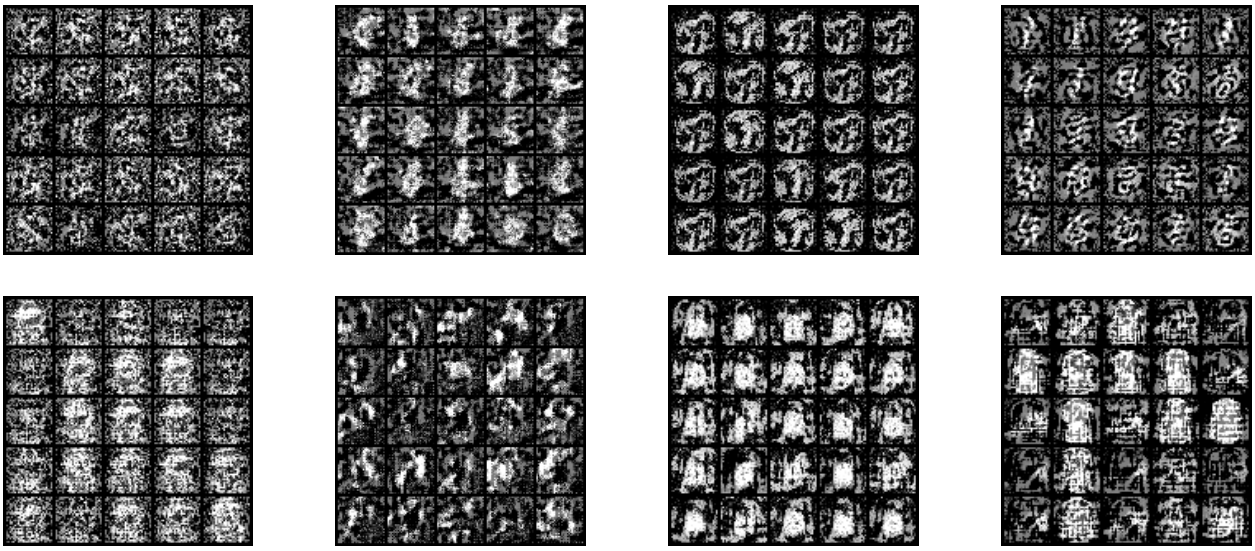
Appendix C

Malicious Samples



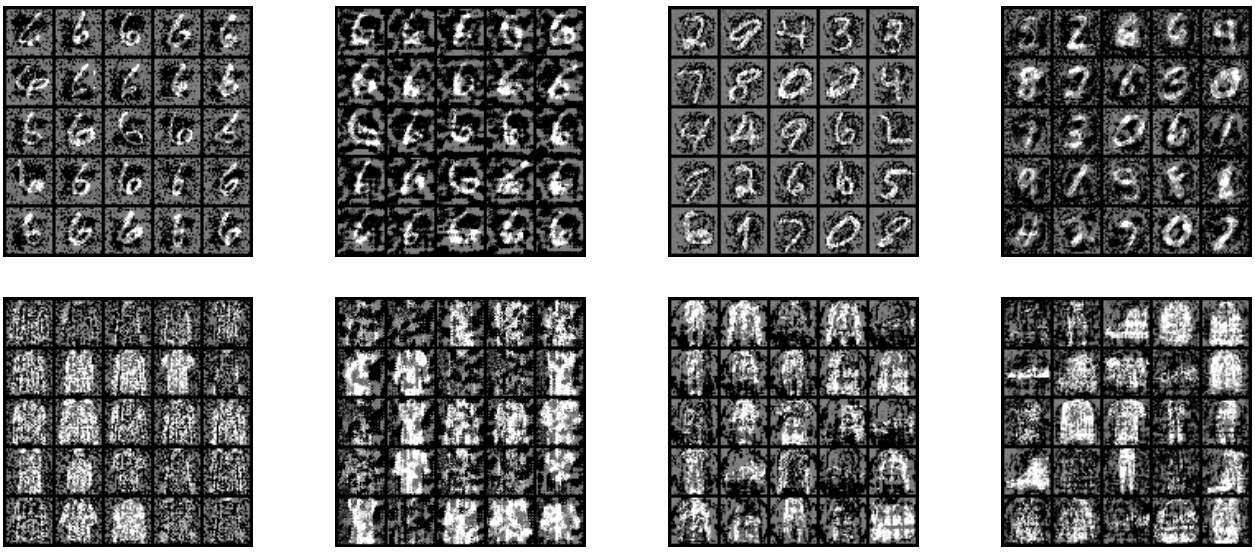
(a) CGAN (b) AC-GAN (c) WGAN (d) WGAN-GP

Figure C.1: Decoys Without PGD. MNIST(top) and F-MNIST(bottom).



(a) CGAN (b) AC-GAN (c) WGAN (d) WGAN-GP

Figure C.2: EarlyStop Decoys With PGD. MNIST(top) and F-MNIST(bottom). Iteration 42.

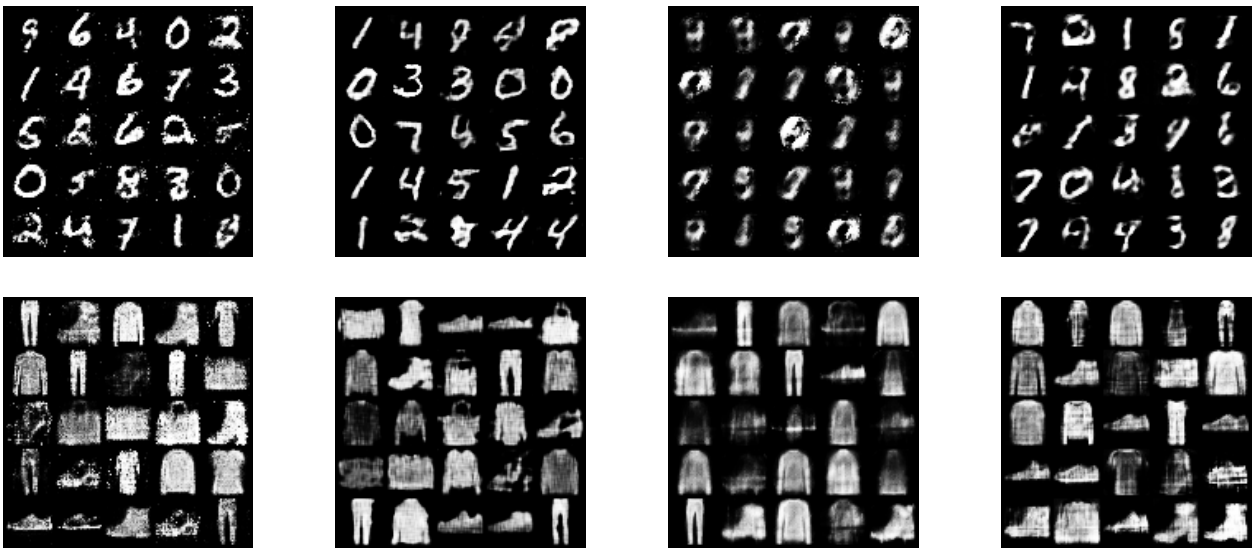


(a) CGAN (b) AC-GAN (c) WGAN (d) WGAN-GP

Figure C.3: Downgrade Decoys With PGD. MNIST(top) and F-MNIST(bottom). Iteration 42.

Appendix D

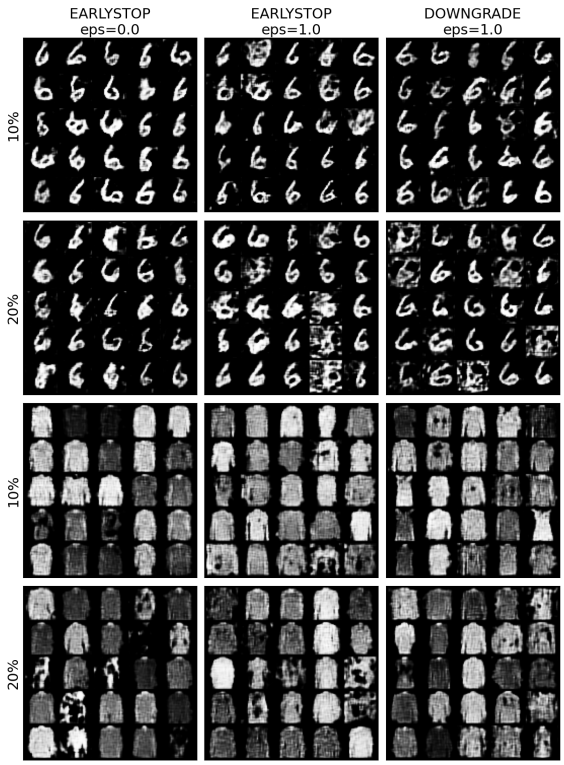
Synthetic Data and Discriminators



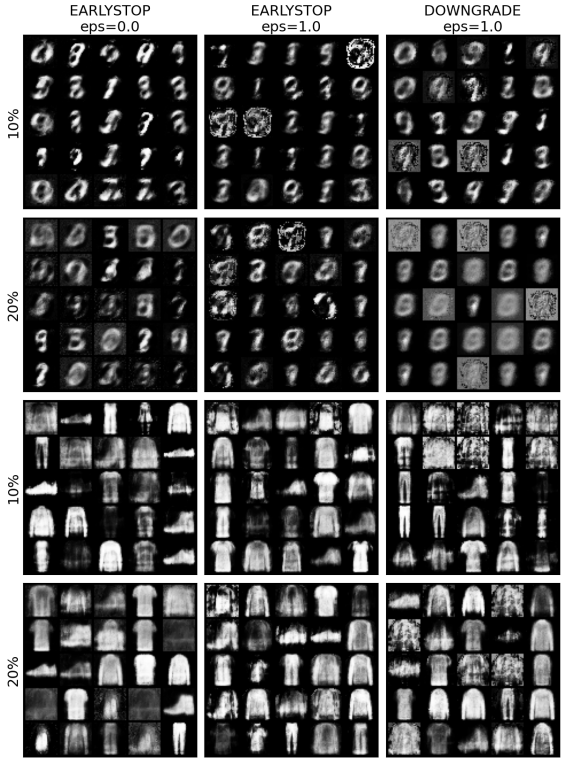
(a) CGAN (b) AC-GAN (c) WGAN (d) WGAN-GP

Figure D.1: Clean Synthetic Data. MNIST(top) and F-MNIST(bottom). Iteration 42.

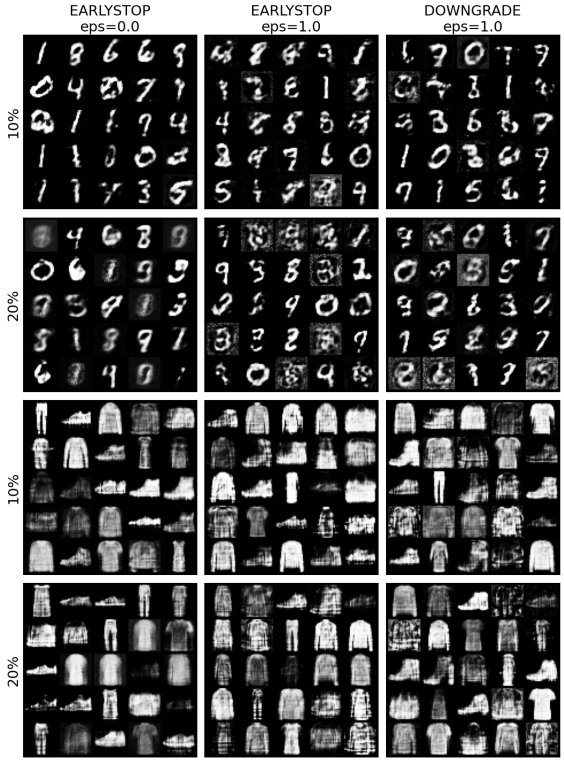
(a) CGAN



(b) AC-GAN



(c) WGAN



(d) WGAN-GP

Figure D.2: Poisoned Synthetic Data. Iteration 42.

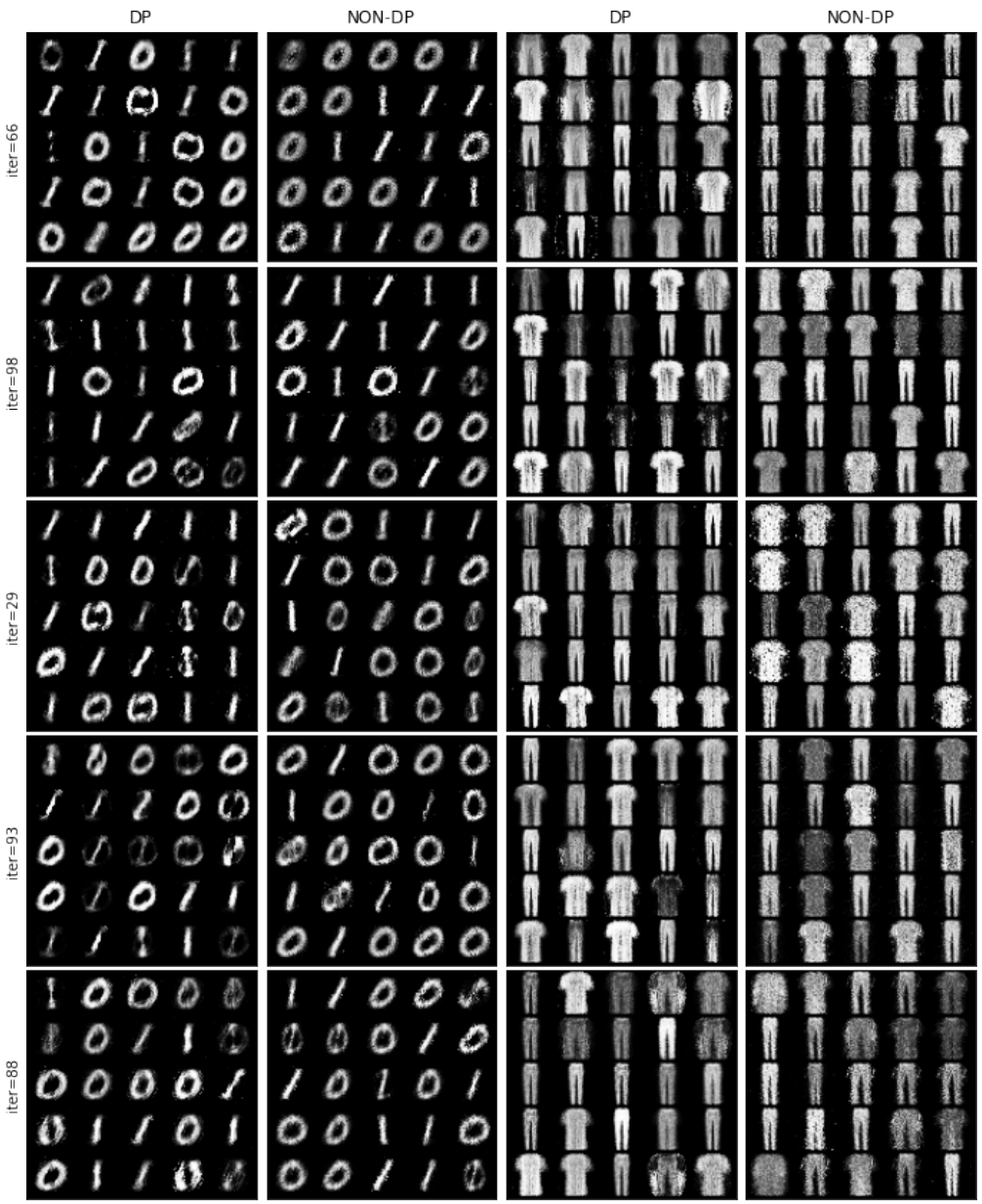
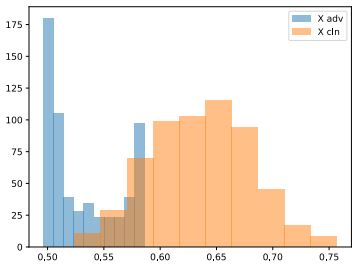
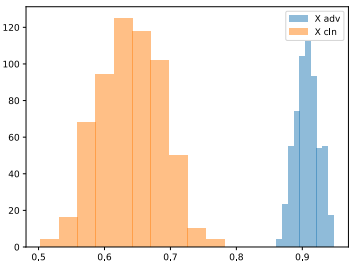


Figure D.3: Clean Synthetic Data Generated by DPWGAN. MNIST (left) and F-MNIST (right).

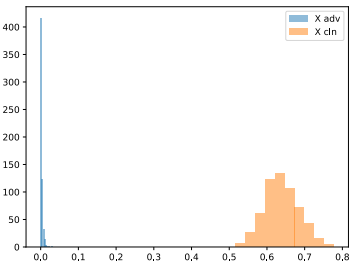
0) MNIST ES eps=0.0 10% FID=86(high)



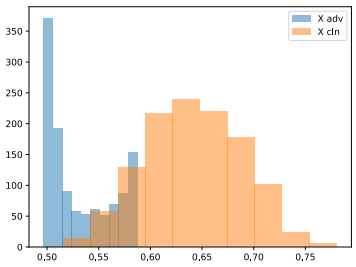
1) MNIST ES eps=1.0 10% FID=70(high)



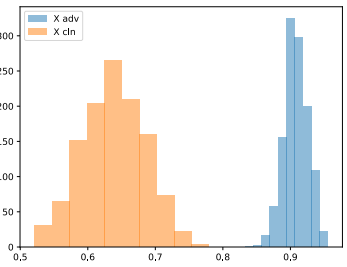
2) MNIST DG eps=1.0 10% FID=67(low)



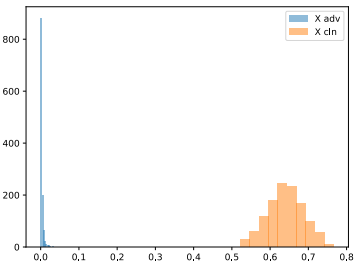
3) MNIST ES eps=0.0 20% FID=51(low)



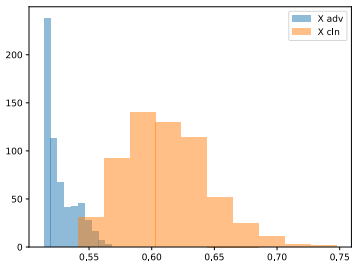
4) MNIST ES eps=1.0 20% FID=69(high)



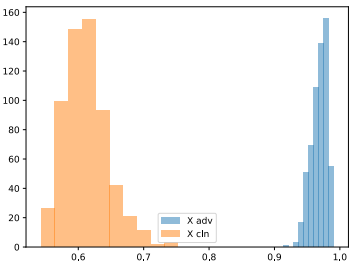
5) MNIST DG eps=1.0 20% FID=83(high)



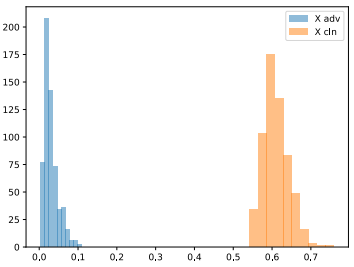
6) FMNIST ES eps=0.0 10% FID=152(high)



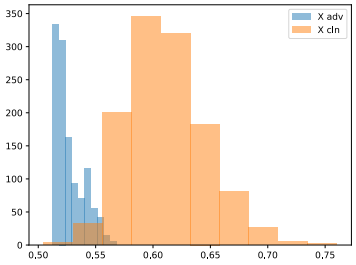
7) FMNIST ES eps=1.0 10% FID=139(low)



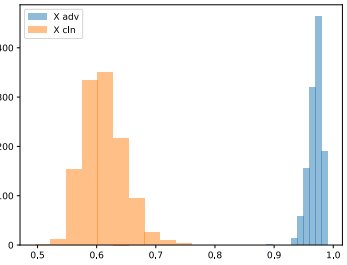
8) FMNIST DG eps=1.0 10% FID=156(high)



9) FMNIST ES eps=0.0 20% FID=143(low)



10) FMNIST ES eps=1.0 20% FID=131(low)



11) FMNIST DG eps=1.0 20% FID=120(low)

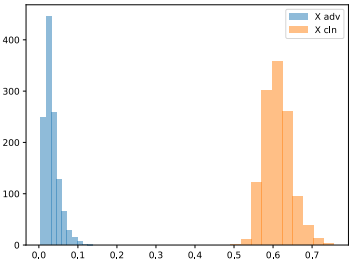


Figure D.4: Poisoned Discriminator CGAN. ES - Early Stop, DG - Downgrade, eps - PGD strength. Percentage refers to the number of decoys. Iteration 42.

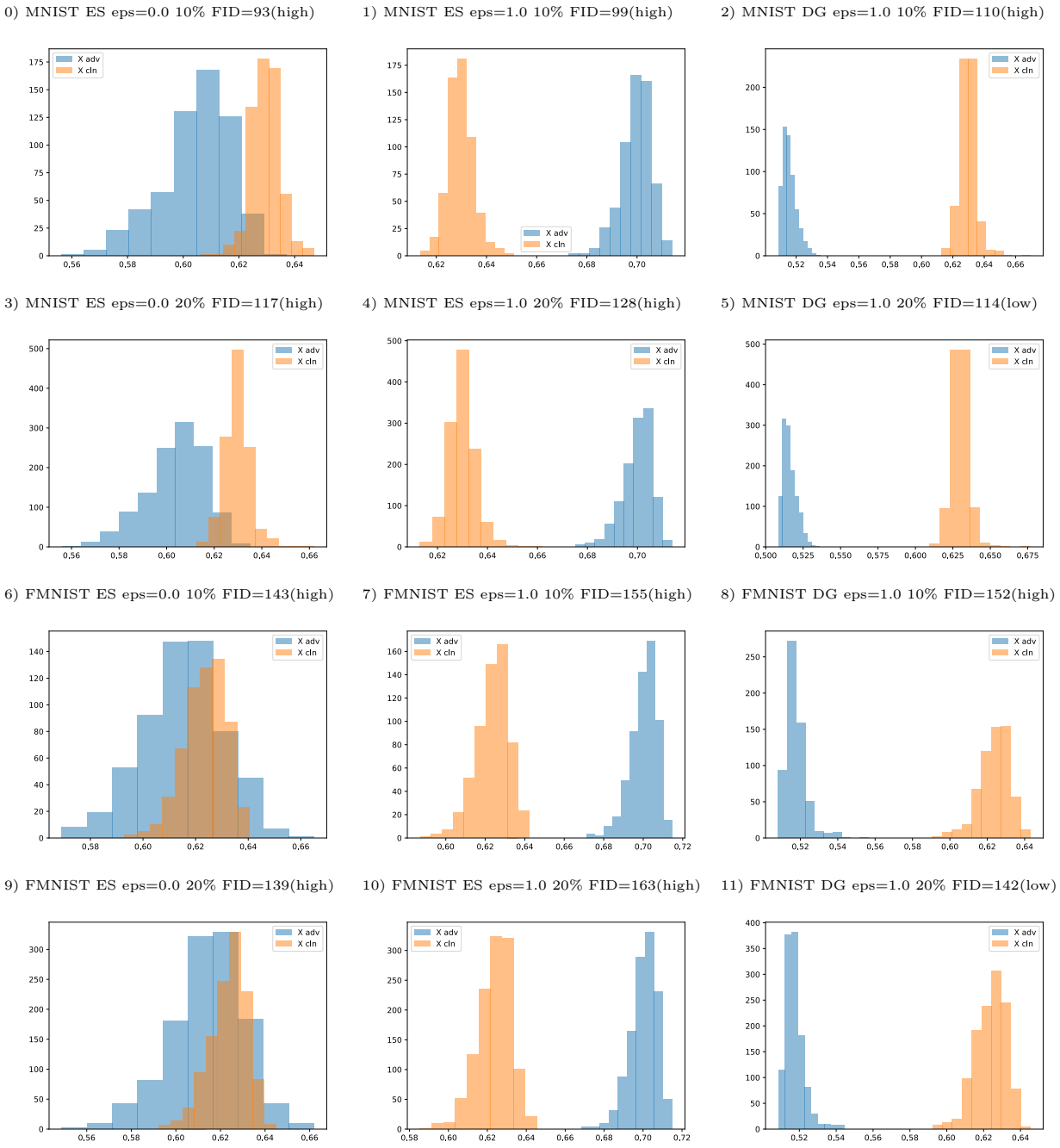
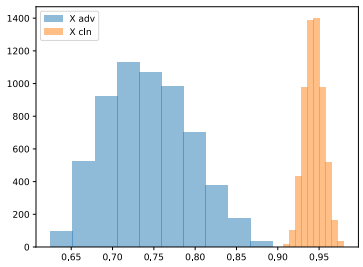
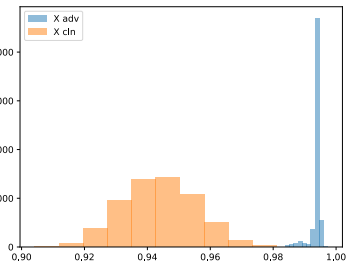


Figure D.5: Poisoned Discriminator AC-GAN. ES - Early Stop, DG - Downgrade, eps - PGD strength. Percentage refers to the number of decoys. Iteration 42.

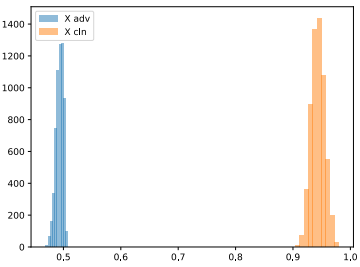
0) MNIST ES eps=0.0 10% FID=168(low)



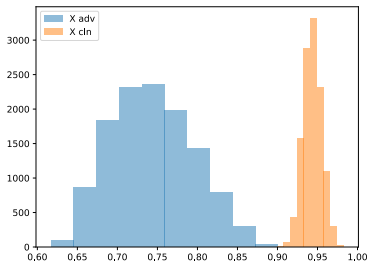
1) MNIST ES eps=1.0 10% FID=176(high)



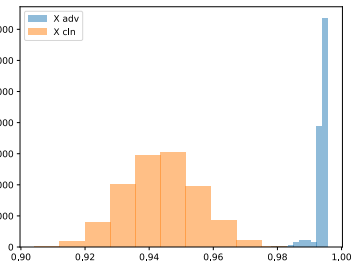
2) MNIST DG eps=1.0 10% FID=176(high)



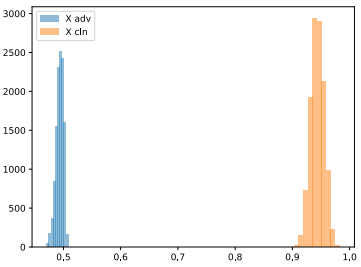
3) MNIST ES eps=0.0 20% FID=231(high)



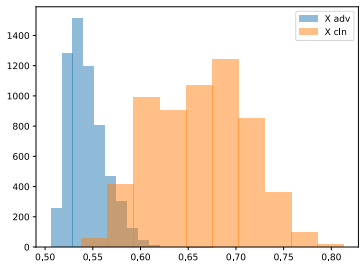
4) MNIST ES eps=1.0 20% FID=183(low)



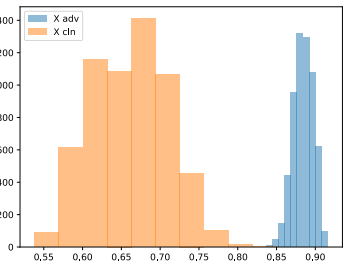
5) MNIST DG eps=1.0 20% FID=273(high)



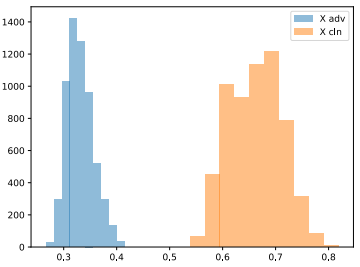
6) FMNIST ES eps=0.0 10% FID=143(low)



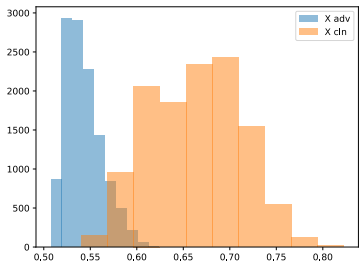
7) FMNIST ES eps=1.0 10% FID=152(high)



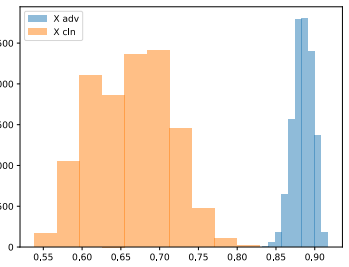
8) FMNIST DG eps=1.0 10% FID=143(high)



9) FMNIST ES eps=0.0 20% FID=165(low)



10) FMNIST ES eps=1.0 20% FID=166(high)



11) FMNIST DG eps=1.0 20% FID=166(high)

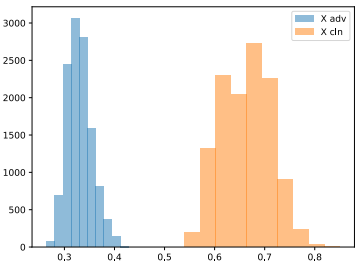


Figure D.6: Poisoned Discriminator WGAN. ES - Early Stop, DG - Downgrade, eps - PGD strength. Percentage refers to the number of decoys. Iteration 42.

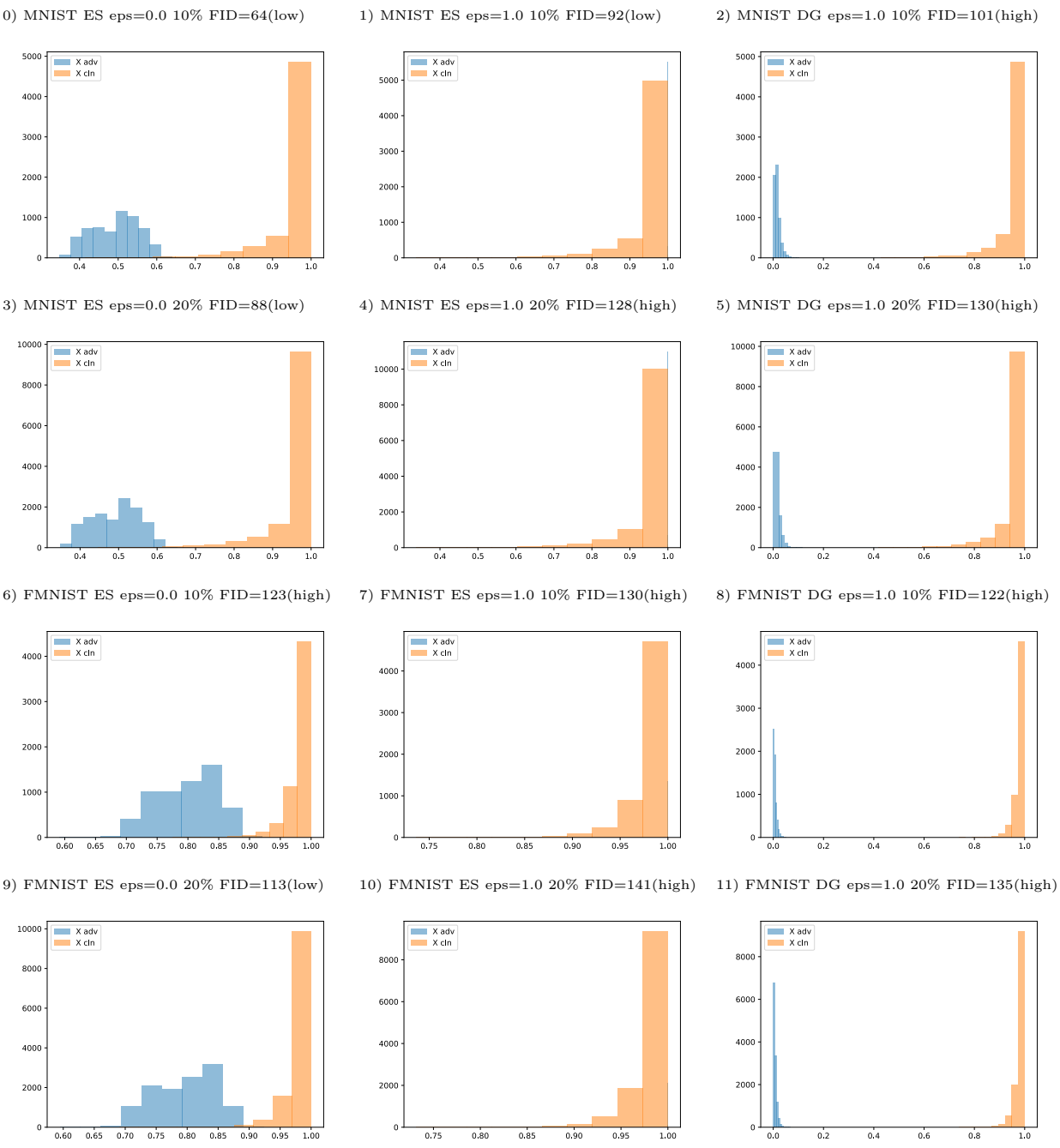


Figure D.7: Poisoned Discriminator WGAN-GP. Some adversarial samples are barely visible because they all achieved 100%. ES - Early Stop, DG - Downgrade, eps - PGD strength. Percentage refers to the number of decoys. Iteration 42.

Bibliography

[1] John M Abowd. *Research Data Centers, Reproducible Science, and Confidentiality Protection: The Role of the 21st Century Statistical Agency*. Demography Seminar. Center for Demography and Ecology, University of Wisconsin–Madison, June 5, 2017. URL: https://cde.wisc.edu/events/demsem/.

[2] Hamed Alqahtani, Manolya Kavakli-Thorne, and Gulshan Kumar. "Applications of Generative Adversarial Networks (GANs): An Updated Review". *Archives of Computational Methods in Engineering* vol. 28, no. 2 (Mar. 1, 2021), pp. 525–552. ISSN: 1886-1784. DOI: 10.1007/s11831-019-09388-y.

[3] Martín Arjovsky and Léon Bottou. "Towards Principled Methods for Training Generative Adversarial Networks". In: *Proceedings of the 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=Hk4_qw5xe.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein Generative Adversarial Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, July 17, 2017, pp. 214–223. URL: https://proceedings.mlr.press/v70/arjovsky17a.html (visited on 11/18/2021).

[5] Aayush Arora and Shantanu. "A Review on Application of GANs in Cybersecurity Domain". *IETE Technical Review* vol. 0, no. 0 (Dec. 6, 2020), pp. 1–9. ISSN: 0256-4602. DOI: 10.1080/02564602.2020.1854058.

[6] Dina Bashkirova, Ben Usman, and Kate Saenko. "Adversarial Self-Defense for Cycle-Consistent GANs". In: *Proceedings of the 32nd Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach et al. 2019, pp. 635–645.

[7] Yaniv Benny et al. "Evaluation Metrics for Conditional Image Generation". *International Journal of Computer Vision* vol. 129, no. 5 (May 1, 2021), pp. 1712–1731. ISSN: 1573-1405. DOI: 10.1007/s11263-020-01424-w.

[8] Charles Blundell et al. "Weight Uncertainty in Neural Network". In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Ed. by Francis R. Bach and David M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 1613–1622. URL: http://proceedings.mlr.press/v37/blundell15.html.

[9] Ashish Bora, Eric Price, and Alexandros G. Dimakis. "AmbientGAN: Generative Models from Lossy Measurements". In: *Proceedings of the 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=Hy7fDog0b.

[10] Edward Choi et al. "Generating Multi-label Discrete Patient Records Using Generative Adversarial Networks". In: *Proceedings of the 2nd Machine Learning for Healthcare Conference*. Machine Learning for Healthcare Conference. PMLR, Nov. 6, 2017, pp. 286–305. URL: https://proceedings.mlr.press/v68/choi17a.html (visited on 11/10/2021).

[11] Grigorios G. Chrysos, Jean Kossaifi, and Stefanos Zafeiriou. "RoCGAN: Robust Conditional GAN". *International Journal of Computer Vision* vol. 128, no. 10 (Nov. 1, 2020), pp. 2665–2683. ISSN: 1573-1405. DOI: 10.1007/s11263-020-01348-5.

[12] Francesco Croce and Matthias Hein. "Reliable Evaluation of Adversarial Robustness with an Ensemble of Diverse Parameter-Free Attacks". In: *Proceedings of the 37th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, Nov. 21, 2020, pp. 2206–2216. URL: https://proceedings.mlr.press/v119/croce20b.html (visited on 11/08/2021).

[13] Gavin Weiguang Ding, Luyu Wang, and Xiaomeng Jin. *Advertorch v0.1: An Adversarial Robustness Toolbox Based on PyTorch*. Feb. 20, 2019. arXiv: 1902.07623 [cs, stat]. URL: www.arxiv.org/abs/1902.07623 (visited on 11/24/2021).

[14] Irit Dinur and Kobbi Nissim. "Revealing Information While Preserving Privacy". In: *Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS '03. New York, NY, USA: Association for Computing Machinery, June 9, 2003, pp. 202–210. ISBN: 978-1-58113-670-8. DOI: 10.1145/773153.773173.

[15] Cynthia Dwork. "A Firm Foundation for Private Data Analysis". *Communications of the ACM* vol. 54, no. 1 (2011), pp. 86–95. DOI: 10.1145/1866739.1866758.

[16] Cynthia Dwork and Aaron Roth. "The Algorithmic Foundations of Differential Privacy". *Foundations and Trends in Theoretical Computer Science* vol. 9, no. 3-4 (2014), pp. 211–407. DOI: 10.1561/0400000042.

[17] Cynthia Dwork et al. "Calibrating Noise to Sensitivity in Private Data Analysis". In: *Proceedings of the 3rd Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*. Ed. by Shai Halevi and Tal Rabin. Vol. 3876. Lecture Notes in Computer Science. Springer, 2006, pp. 265–284. DOI: 10.1007/11681878_14.

[18] Justin Engelmann and Stefan Lessmann. "Conditional Wasserstein GAN-based Oversampling of Tabular Data for Imbalanced Learning". *Expert Systems with Applications* vol. 174 (July 15, 2021), p. 114582. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2021.114582.

[19] Joshua Falk and Giovanni Meneses. *DPWGAN*. GitHub: Civis Analytics, Nov. 12, 2021. URL: https://github.com/civisanalytics/dpwgan (visited on 11/25/2021).

[20] Liyue Fan. "A Survey of Differentially Private Generative Adversarial Networks". In: *Proceedings of the 34th Conference on Artificial Intelligence (AAAI-21) Workshop on Privacy-Preserving Artificial Intelligence*. AAAI Conference on Artificial Intelligence (AAAI-21). Vol. 34. New York, USA: AAAI Press, 2020. ISBN: 978-1-57735-835-0. URL: https://www2.isye.gatech.edu/~fferdinando3/cfp/PPAI20/.

[21] Nariman Farsad. *Deep Learning*. CP8318 Machine Learning Lectures. Ryerson University, Aut. 2020.

[22] Mingming Gong et al. "Twin Auxiliary Classifiers GAN". *Advances in neural information processing systems* vol. 32 (Dec. 2019), pp. 1328–1337. ISSN: 1049-5258. pmid: 32103879. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7042662/ (visited on 11/19/2021).

[23] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6572.

[24] Ian J. Goodfellow et al. "Generative Adversarial Nets". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. International Conference on Neural Information Processing Systems. NIPS'14. Cambridge, MA, USA: MIT Press, Dec. 8, 2014, pp. 2672–2680.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cumberland, UNITED STATES: MIT Press, 2016. ISBN: 978-0-262-33737-3. URL: http://ebookcentral.proquest.com/lib/ryerson/detail.action?docID=6287197 (visited on 11/12/2021).

[26] Ishaan Gulrajani et al. "Improved Training of Wasserstein GANs". In: *Proceedings of the 30th Annual Conference on Neural Information Processing Systems*. Annual Conference on Neural Information Processing Systems. Ed. by I. Guyon et al. Curran Associates, Inc., 2017.

[27] Martin Heusel et al. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium". In: *Proceedings of the 30th Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon et al. 2017, pp. 6626–6637. URL: https://proceedings.neurips.cc/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html.

[28] Andrew Ilyas et al. "Adversarial Examples Are Not Bugs, They Are Features". In: *Proceedings of the 32nd Annual Conference on Neural Information Processing System, Vancouver, BC, Canada*. Vancouver, BC, Canada: Curran Associates, Inc., 2019.

[29] Sara Kaviani and Insoo Sohn. "Defense against Neural Trojan Attacks: A Survey". *Neurocomputing* vol. 423 (Jan. 29, 2021), pp. 651–667. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2020.07.133.

[30] Martin Krzywinski and Naomi Altman. "Visualizing Samples with Box Plots". *Nature Methods* vol. 11, no. 2 (2 Feb. 1, 2014), pp. 119–120. ISSN: 1548-7105. DOI: 10.1038/nmeth.2813.

[31] Y. Lecun et al. "Gradient-Based Learning Applied to Document Recognition". *Proceedings of the IEEE* vol. 86, no. 11 (Nov./1998), pp. 2278–2324. ISSN: 00189219. DOI: 10.1109/5.726791.

[32] Jaewoo Lee and Chris Clifton. "How Much Is Enough? Choosing Epsilon for Differential Privacy". In: *Proceedings of the International Conference on Information Secu-*

rity. Ed. by Xuejia Lai, Jianying Zhou, and Hui Li. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 325–340. ISBN: 978-3-642-24861-0. DOI: 10.1007/978-3-642-24861-0_22.

[33] Erik Linder-Norén. *Eriklindernoren/PyTorch-GAN*. GitHub, Nov. 23, 2021. URL: https://github.com/eriklindernoren/PyTorch-GAN (visited on 11/23/2021).

[34] Xuanqing Liu and Cho-Jui Hsieh. "Rob-GAN: Generator, Discriminator, and Adversarial Attacker". In: *Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). June 2019, pp. 11226–11235. DOI: 10.1109/CVPR.2019.01149.

[35] Pei-Hsuan Lu, Pang-Chieh Wang, and Chia-Mu Yu. "Empirical Evaluation on Synthetic Data Generation with Generative Adversarial Network". In: *Proceedings of the 9th International Conference on Web Intelligence, Mining and Semantics*. WIMS2019. Seoul, Republic of Korea: Association for Computing Machinery, June 26, 2019, pp. 1–6. ISBN: 978-1-4503-6190-3. DOI: 10.1145/3326467.3326474.

[36] Mario Lucic et al. "Are GANs Created Equal? A Large-Scale Study". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. International Conference on Neural Information Processing Systems. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., Dec. 3, 2018, pp. 698–707.

[37] Tengyu Ma et al. *Deep Learning*. CS229 Machine Learning Lectures. Stanford University, 2021. URL: http://cs229.stanford.edu/syllabus.html (visited on 11/12/2021).

[38] Aleksander Madry et al. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *Proceedings of the 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference*

Track Proceedings. OpenReview.net, 2018. URL: https://openreview.net/forum?id=rJzIBfZAb.

[39] Xudong Mao et al. "On the Effectiveness of Least Squares Generative Adversarial Networks". *IEEE Transactions on Pattern Analysis and Machine Intelligence* vol. 41, no. 12 (Dec. 2019), pp. 2947–2960. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2018.2872043.

[40] D. J. Miller, Z. Xiang, and G. Kesidis. "Adversarial Learning Targeting Deep Neural Network Classification: A Comprehensive Review of Defenses Against Attacks". *Proceedings of the IEEE* vol. 108, no. 3 (Mar. 2020), pp. 402–433. ISSN: 1558-2256. DOI: 10.1109/JPROC.2020.2970615.

[41] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. Nov. 6, 2014. arXiv: 1411.1784 [cs, stat]. URL: http://arxiv.org/abs/1411.1784 (visited on 11/18/2021).

[42] Augustus Odena, Christopher Olah, and Jonathon Shlens. "Conditional Image Synthesis with Auxiliary Classifier GANs". In: *Proceedings of the International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, July 17, 2017, pp. 2642–2651. URL: http://proceedings.mlr.press/v70/odena17a.html (visited on 05/06/2021).

[43] Hanwool Park. *Defense-GAN_Pytorch*. GitHub, June 16, 2021. URL: https://github.com/sky4689524/DefenseGAN-Pytorch (visited on 11/25/2021).

[44] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Proceedings of the 32nd Annual Conference on Neural Information Processing Systems*. Annual Conference on Neural Information Processing Systems, NeurIPS. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[45] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. "Defense-Gan: Protecting Classifiers against Adversarial Attacks Using Generative Models". In: *Proceedings of the 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=BkJ3ibb0-.

[46] Pascal Schöttle et al. "Detecting Adversarial Examples - a Lesson from Multimedia Security". In: *Proceedings of the 26th European Signal Processing Conference (EUSIPCO)*. European Signal Processing Conference (EUSIPCO). Sept. 2018, pp. 947–951. DOI: 10.23919/EUSIPCO.2018.8553164.

[47] Shiliang Sun et al. "A Survey of Optimization Methods From a Machine Learning Perspective". *IEEE Transactions on Cybernetics* vol. 50, no. 8 (Aug. 2020), pp. 3668–3681. ISSN: 2168-2275. DOI: 10.1109/TCYB.2019.2950779.

[48] Christian Szegedy et al. "Going Deeper with Convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[49] Kiran Koshy Thekumparampil et al. "Robustness of Conditional GANs to Noisy Labels". In: *Proceedings of the 31st Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio et al. 2018, pp. 10292–10303. URL: https://proceedings.neurips.cc/paper/2018/hash/565e8a413d0562de9ee4378402d2b481-Abstract.html.

[50] Rafael Valle. *Hands-On Generative Adversarial Networks with Keras: Your Guide to Implementing Next-Generation Generative Adversarial Networks*. Birmingham: Packt Publishing, Limited, 2019. ISBN: 9781789538205;1789538203;

[51] Zhenchen Wang, Puja Myles, and Allan Tucker. "Generating and Evaluating Synthetic UK Primary Care Data: Preserving Data Utility & Patient Privacy". In: *Proceedings*

of the 2019 IEEE 32nd International Symposium on Computer-Based Medical Systems (CBMS). IEEE International Symposium on Computer-Based Medical Systems (CBMS). Cordoba, Spain: IEEE, June 2019, pp. 126–131. ISBN: 978-1-72812-286-1. DOI: 10.1109/CBMS.2019.00036.

[52] Max Welling and Yee Teh. *Bayesian Learning via Stochastic Gradient Langevin Dynamics*. Test Of Time Award Presentation. International Conference on Machine Learning, 2021. URL: https://icml.cc/virtual/2021/test-of-time/11808#details (visited on 11/13/2021).

[53] Max Welling and Yee Whye Teh. "Bayesian Learning via Stochastic Gradient Langevin Dynamics". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. International Conference on International Conference on Machine Learning. ICML'11. Madison, WI, USA: Omnipress, June 28, 2011, pp. 681–688. ISBN: 978-1-4503-0619-5.

[54] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Sept. 15, 2017. arXiv: 1708.07747 [cs, stat]. URL: http://arxiv.org/abs/1708.07747 (visited on 11/23/2021).

[55] Liyang Xie et al. "Differentially Private Generative Adversarial Network". *CoRR* (2018). arXiv: 1802.06739. URL: http://arxiv.org/abs/1802.06739.

[56] Zhi Xu, Chengtao Li, and Stefanie Jegelka. "Robust GANs against Dishonest Adversaries". In: *Proceedings of the International Conference on Machine Learning Workshop on Security and Privacy of ML*. International Conference on Machine Learning. Oct. 9, 2019. arXiv: 1802.09700. URL: http://arxiv.org/abs/1802.09700 (visited on 03/18/2021).

[57] Digvijay Yadav and Sakina Salmani. "Deepfake: A Survey on Facial Forgery Technique Using Generative Adversarial Network". In: *Proceedings of the 2019 International Conference on Intelligent Computing and Control Systems (ICCS)*. International Confer-

ence on Intelligent Computing and Control Systems (ICCS). May 2019, pp. 852–857. DOI: 10.1109/ICCS45141.2019.9065881.

[58] Jingwen Zhao, Yunfang Chen, and Wei Zhang. "Differential Privacy Preservation in Deep Learning: Challenges, Opportunities and Solutions". *IEEE Access* vol. 7 (2019), pp. 48901–48911.

[59] Tianhang Zheng, Changyou Chen, and Kui Ren. "Distributionally Adversarial Attack". *Proceedings of the AAAI Conference on Artificial Intelligence* vol. 33, no. 01 (01 July 17, 2019), pp. 2253–2260. ISSN: 2374-3468. DOI: 10.1609/aaai.v33i01.33012253.

[60] Brady Zhou and Philipp Krähenbühl. "Don't Let Your Discriminator Be Fooled". In: *Proceedings of the 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HJE6X305Fm.

Acronyms

AC-GAN Auxiliary Conditional GAN. 9, 12, 15, 23, 38, 43

AUC Area Under the Curve. 39

BCE Binary Cross Entropy. 32, 33

CE Cross Entropy. 33

CGAN Conditional GAN. 12, 15, 38, 43

Def-GAN Defense GAN. 22, 34, 39, 56

DGD Downgrade Decoy. 28, 43, 44, 51, 55, 59, 60

DP Differential Privacy. 2, 18, 19

DP-WGAN Differentially Private Wasserstein GAN. 20, 25, 36, 38, 56

EED Early Epoch Decoy. 28, 41, 43–45, 47, 50, 51, 53–57, 60

EM Earth Mover Distance. 15

F-MNIST Fashion-MNIST. 29

FID Fréchet Inception Distance. 20, 25, 39

FPR False Positive Rate. 5, 50

GAN Generative Adversarial Network. 1, 4, 12

JSD Jensen–Shannon Divergence. 13–15

MNIST Modified National Institute of Standards and Technology Database. 29

MSE Mean Squared Error. 35

MWA Monkey-Wrench Attack. vi, 5, 27, 28, 35, 38

PGD Projected Gradient Descent. 21, 23, 28

ROC Receiver Operating Characteristic. 39, 48

SGD Stochastic Gradient Descent. 9, 11

SGLD Stochastic Gradient Langevin Dynamics. 9, 11

TPR True Positive Rate. 5, 50

WGAN Wasserstein GAN. 12, 15, 16, 19, 38, 44

WGAN-GP Wasserstein GAN with Gradient Penalty. 9, 12, 24, 38, 44