國立臺灣大學電機資訊學院電信工程學研究所

碩士論文

Graduate Institute of Communication Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

差集生成網路–新穎資料生成

Difference-Seeking Generative Adversarial Network–Unseen Data Generation

宋易霖

Yi-Lin Sung

指導教授：貝蘇章博士

Advisor: Soo-Chang Pei, Ph.D.

中華民國 108 年 5 月

May, 2019

doi:10.6342/NTU201901502

國立臺灣大學（碩）博士學位論文
口試委員會審定書

差集生成對抗網路—新穎資料生成

Difference-Seeking Generative Adversarial Network—Unseen Data Generation

本論文係 宋易霖 君（r06942076）在國立臺灣大學電信工程學研究所完成之碩（博）士學位論文，於民國 108 年 5 月 21 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

貝蘇章 （簽名）

（指導教授）

丁建均 [illegible]

[illegible]

[illegible]

所　　長 [illegible] （簽名）

doi:10.6342/NTU201901502

誌謝

我很感謝在碩士這兩年時光遇到的每個人，讓我發現即使論文投稿一直失敗，這段旅程還是很充實的。

第一個要感謝的是貝蘇章教授，謝謝你當初願意收一個化工系的學生進入實驗室，老師耐心的指導使我成長，勤奮不懈的身影是最好的身教。畢業後我也會抱持著同樣的精神，努力成為一個在科技產業上有貢獻的人，當然第一步還是希望能投稿成功...祝老師退而不休快樂!

我也要謝謝李宏毅教授，修了老師的兩門課讓我進入深度學習領域的大門。後來擔任助教的時候很佩服老師對知識的堅持以及對實驗敏銳的觀察。未來繼續在這條路上希望還能和老師有所交流。

這本論文的完成很大的部份必須歸功於謝松憲學長，除此之外，碩二每週和學長討論絕對是讓我進步最大的原因。學長讓我了解數學基礎對研究的重要性，也因此我修了幾門回想起來很痛苦，但收穫良多的數學課。學長對於研究的想法和知識的細節也讓我非常欽佩。我們一定要把這個研究成果投稿到 AI 的頂會上！

當然也要感謝實驗室的同學以及學弟妹。感謝你們罩我 DSP，不然後果不堪設想。也謝謝你們包容我這個難熟的人，直到快畢業了才感覺跟大家變熟，真的有點可惜。未來不知道會不會常見面，但如果需要我而我有能力的話，我一定大力相挺。

最後要老套的感謝一下我的家人以及女友，謝謝你們一直相信一個明明不太強的我。我會努力的朝自己的目標邁進，回饋社會也回饋你們。

…寫於 2019 年 7 月 15 日 …

doi:10.6342/NTU201901502

doi:10.6342/NTU201901502

Acknowledgements

I'm glad to thank everyone I met during these two years.

doi:10.6342/NTU201901502

doi:10.6342/NTU201901502

摘要

新穎資料泛指那些不落在訓練資料的分佈中的資料，而他們在某些應用是很重要的，如半監督學習、增強網路的穩定性和異常偵測等。新穎資料通常難以取得，但是如果能夠有演算法能夠產生這些資料並在訓練時使用，那麼將可以大幅增強模型。因此如何產生這些資料是一個常見的研究議題。不同應用所需要的新穎資料往往不太相同，目前針對各種應用也有不同的方法。在這篇論文中，我們提出一個演算法-差集生成對抗網路，能夠產生各種新穎資料。我們發現新穎資料所在的分佈常常是兩個已知分佈的差集，而這兩個已知分佈的資料是比較容易蒐集到的，甚至都可以從訓練資料變化而來。我們將差集對抗網路應用在半監督學習、加強深度網路的穩定性以及異常偵測，實驗結果證明我們的方法是有效的。除此之外，我們也提供理論的證明保證演算法的收斂性。

關鍵字： 差集學習、生成對抗網路、半監督式學習、強健的深度網路、異常偵測

doi:10.6342/NTU201901502

doi:10.6342/NTU201901502

Abstract

Unseen data, which are not samples from the distribution of training data and are difficult to collect, have exhibited the importance in many applications (*e.g.,* novelty detection, semi-supervised learning, adversarial training and so on.). In this paper, we introduce a general framework, called **D**ifference-**S**eeking **G**enerative **A**dversarial **N**etwork (DSGAN), to create various kinds of unseen data. The novelty is to consider the probability density of unseen data distribution to be the difference between those of two distributions $p_{\bar{d}}$ and p_d, whose samples are relatively easy to collect. DSGAN can learn the target distribution p_t (or the unseen data distribution) via only the samples from the two distributions p_d and $p_{\bar{d}}$. Under our scenario, p_d is the distribution of seen data and $p_{\bar{d}}$ can be obtained from p_d via simple operations, implying that we only need the samples of p_d during training. Three key applications, semi-supervised learning, increasing the robustness of neural network and novelty detection, are taken as case studies to illustrate that DSGAN enables to produce various unseen data. We also provide theoretical analyses about the convergence of DSGAN.

Keywords: Difference-Seeking, Generative Adversarial Network, Semi-Supervised Learning, Robustness of Neural Network, Novelty Detection

doi:10.6342/NTU201901502

doi:10.6342/NTU201901502

Contents

doi:10.6342/NTU201901502

doi:10.6342/NTU201901502

List of Figures

doi:10.6342/NTU201901502

doi:10.6342/NTU201901502

List of Tables

doi:10.6342/NTU201901502

doi:10.6342/NTU201901502

Chapter 1

Introduction

Unseen data are not samples from the distribution of training data and are difficult to collect. It has been demonstrated that the unseen samples can be applied to several applications. [5] proposed how to create complement data and theoretically showed that complement data, considered as unseen data, can improve the semi-supervised learning. In novelty detection, [11] proposed a method to generate unseen data and used them to train a anomaly detector. Another related issue is adversarial training [12], where classifiers are trained to resist against adversarial examples, which are unseen during the training phase. However, the aforementioned methods only focus on producing specific kind of unseen data instead of enabling to generate general types of unseen data.

In this paper, we propose a general framework, called DSGAN, to generate a variety of unseen data. DSGAN is one of the generative approaches. In tradition, generative approaches, which are usually conducted in an unsupervised learning manner, are developed for learning data distribution from its samples and thereafter produce novel and high-dimensional samples, such as synthesized image, from learned distributions [13]. The state-of-the-art approach is so-called Generative Adversarial Networks (GAN) [14]. GAN produces sharp images based on a game-theoretic framework, but can be tricky and unstable to train due to multiple interacting losses. Specifically, GAN consists of two functions: generator and discriminator. Both functions are represented as parameterized neural networks. The discriminator network is trained to classify whether or not inputs belong to the real data set or fake data set created by the generator. The generator learns to

doi:10.6342/NTU201901502

map a sample from a latent space to some distribution to increase the classification errors of the discriminator.

Nevertheless, if we aim to learn a generator to create unseen data, traditional GAN requires preparing plenty of training samples of unseen classes for training, leading to the contradiction with the definition of unseen data. This fact motivates us to present DSGAN, which can generate unseen data by taking seen data as training samples (see Fig. 1.1, which illustrates the difference between GAN and DSGAN). The key idea is to consider the distribution of unseen data as the difference between two distributions, in which both are relatively easy to obtain. For example, out-of-distribution examples in the MNIST dataset, from another point of view, are found to belong to the difference between the set of examples in MNIST and the universal set. It should be noted that the target distribution is equal to the training data distribution in traditional GAN; however, these two distributions, target distribution and training data distribution, are considered different in DSGAN.

In this paper, we make the following contributions:

(1) We propose DSGAN to generate any unseen data only if the density of target (unseen data) distribution is the difference between those of any two distributions, $p_{\bar{d}}$ and p_d. By contrast, traditional GAN fails to learn the difference between two distributions.

(2) We show that DSGAN possesses the flexibility to learn different target (unseen data) distributions in three key applications, semi-supervised learning, increasing the robustness of neural network and novelty detection. Specifically, for novelty detection, DSGAN can produce boundary points around seen data because this kind of unseen data is easily misclassified. DSGAN also generates boundary samples to increase the robustness of neural network, but the distance measured in ℓ_{inf} norm. For semi-supervised learning, unseen data are the linear combination of any labeled data and unlabeled data, excluding labeled and unlabeled data themselves[1].

(3) Our theoretical analysis shows that, with enough capacity of the generator and the

[1]The linear combination of any labeled data and unlabeled data probably belongs to the set of seen data (labeled data and unlabeled data), which contradicts with the definition of unseen data. Thus, the samples generated by DSGAN should not include seen data themselves.

doi:10.6342/NTU201901502

discriminator, the generator can learn the target distribution p_t, whose support set is the difference of support sets between $p_{\bar{d}}$ and p_d, under mild conditions.

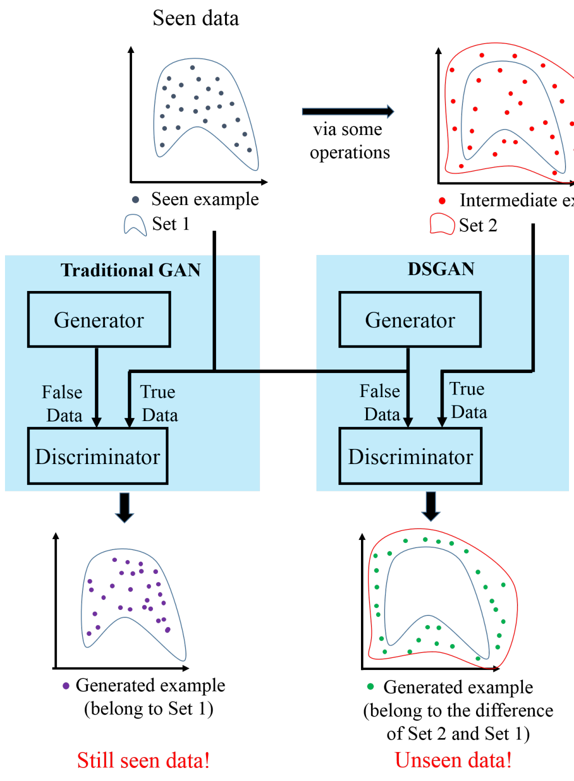


Figure 1.1: Illustration of the differences between traditional GAN and DSGAN.

doi:10.6342/NTU201901502

doi:10.6342/NTU201901502

Chapter 2

Backgrounds

2.1 Deep Generative Model

A deep generative model is to learn the underlying data distribution $P_{\mathcal{X}}$ from limited training data $\mathcal{X}$ via neural network. The learned data distribution can be apply to several scenarios, e.g. classification problem, representation learning, compressed sensing, etc. One can view that to learn a generative model is similar to teach machine understanding the world. Recently, there are some well-known algorithms for deep generative models, including Variational Autoencoders (VAE) ([7]), Generative Adversarial Networks (GAN) ([14]), autoregressive models ([15]), and normalizing flow models ([16]). In this thesis, I focus on studying GANs.

2.2 Generative Adversarial Network

GAN is one of the popular framework in generation in recent years and it is successfully apply to various applications ([17], [18], [19]). GAN sets up a min-max game between the generator and discriminator. Generator tries to learn the data distribution to fool the discriminator while the discriminator learns to distinguish the input coming from the data distribution or the generator.

To learn the generator, one has to define a prior distribution p_z of the input variable z, and p_z is $\mathcal{U}(0, 1)$ or $N(0, 1)$ in most of the time. Generator is a differentiable mapping

doi:10.6342/NTU201901502

function G with output space $G(z;\theta_g)$. The goal of the generator is to find a optimal θ_g to let the distribution of $G(z;\theta_g)$ (that is, p_g) equal to the data distribution $p_{\mathcal{X}}$. The discriminator is define as $D(x;\theta_d)$ that outputs scalar value. $D(x)$ can be viewed as the probability of that x is from $p_{\mathcal{X}}$. In other words, the discriminator is a binary classifier to distinguish the input is from $p_{\mathcal{X}}$ or p_g. The workflow of GAN is demonstrate in Fig. 2.1.

One train D to maximize the probability that assign 1 to the x and 0 to $G(z)$ (or x_g). In the mean while, G is being trained to maximize $D(x_g)$. This procedure can be treated as a min-max game between G and D. More specifically, the objective function of GAN is

$$F(G,D) = \mathbb{E}_{x\sim p_d}\left[\log\left(D(x)\right)\right] + \mathbb{E}_{z\sim p_z}\left[\log\left(1 - D(G(z))\right)\right] \tag{2.1}$$

And one can optimize 2.1 by updating G and D, that is

$$\min_G \max_D F(G,D).$$

Under mild assumptions, optimizing 2.1 equals to minimize the Jensen-Shannon divergence between $p_{\mathcal{X}}$ and p_g. The global optimum is reached only if $p_g = p_{\mathcal{X}}$. The detailed proof is in [14].

In 2.1, the generator is going to be minimized until the discriminator is at optimal. However, this training procedure is inefficient. Therefore, [14] claims that the discriminator only need constant multiple times (*e.g.* 5) updating steps per generator updating step. [20] points out that use higher learning rate for discriminator also achieve similar result. With this technique, we can alternatively train G and D each for one iteration, and it can shorten the training time.

2.3 Wasserstein GAN

It is demonstrated that there are some problems in training GAN: unstable training process, gradient vanishing problem, mode collapse in generator, etc. Lots of works are trying to address those issues. [21] proposed the convolutional architectures for both generator and

doi:10.6342/NTU201901502

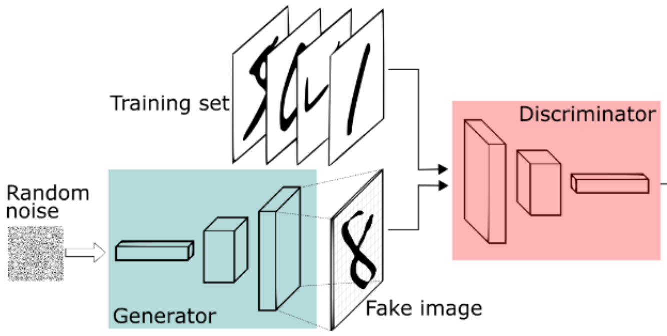


Figure 2.1: The workflow of GAN. (Source: *https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394*)

discriminator to stabilize training. [22] define an alternative loss function and the proposed GAN can not only generate high quality images but also have more stable training process. [23] mathematically analyze the training dynamics of training GAN. Moreover, they state that the unstable issues of GAN comes from the objective function. Based on [24], [25] proposed Wasserstein GAN with a modified objective function,

$$W(G, D) = \mathbb{E}_{x \sim p_d}[D(x)] - \mathbb{E}_{z \sim p_z}[D(G(z))] \tag{2.2}$$

Same as what we do in training GAN, we update G and D to minimize and maximize 2.2 respectively, that is,

$$\min_{G} \max_{\|D\|_L \leq 1} W(G, D)$$

where $\|D\|_L \leq 1$ denotes that D meets the 1-Lipschitz continuity.

Training WGAN can be viewed as minimizing the Wasserstein distance (Earth-Mover distance) between $p_{\mathcal{X}}$ and p_g. WGAN is exhibited to be more stable and it can cover more modes of the training data. The reason why Wasserstein distance works better than Jensen-Shannon divergence in training GAN refer to [24] and [25].

In order to let WGAN success, one has to constrain the Lipschitz continuity of the discriminator. In [25], the author use the weight clipping to enforce the condition. However, the method may lead some undesired behavior. [26] instead constrain the gradient norm of the discriminator's output with respect to its input, and the proposed GAN called WGAN-GP (GP is the abbreviation of gradient penalty).

doi:10.6342/NTU201901502

2.4 Semi-Supervised Learning with GANs

Please refer to Sec. 5.1.

2.5 Robust Issue of Neural Networks

Please refer to Sec. 5.2.

2.6 Novelty Detection by Reconstruction Method

Please refer to Sec. 5.3.

2.7 Related Works

We introduce related works about generating unseen data.

[11] proposed a method to generate samples of unseen classes in the unsupervised manner via an adversarial learning strategy. But, it requires solving an optimization problem for each sample, which undoubtedly lead to high computation cost. On the contrary, DSGAN has the capability to create infinite diverse unseen samples. [27] presented a new GAN architecture that can learn both distributions of unseen data from part of seen data and unlabeled data. But, unlabeled data must be a mixture of seen and unseen samples; DSGAN does not require any unseen data instead. [5] aims to generate complementary samples (or out-of-distribution samples) but assumes that in-distribution can be estimated by a pretrained model such as PixelCNN++, which might be difficult and expensive to train. [28] uses a simple classifier to replace the role of PixelCNN++ in [5] such that the training is much easier and more suitable. Nevertheless, their method only focuses on generates unseen data surrounding the low-density area of seen data, but DSGAN is more flexible to generate different kinds of unseen data (*e.g.,* the linear combination of seen data described in Sec.6.1). In addition, their method needs the label information of data while ours is fully unsupervised.

doi:10.6342/NTU201901502

Related works about semi-supervised learning and enhancing robustness of neural network are presented in Sec. 5

doi:10.6342/NTU201901502

doi:10.6342/NTU201901502

Chapter 3

Proposed Method-DSGAN

3.1 Formulation

We denote the generator distribution as p_g and training data distribution as p_d, both in a N-dimensional space. Let $p_{\bar{d}}$ be the distribution decided by user. For example, $p_{\bar{d}}$ can be the convolution of p_d and normal distribution. Let p_t be the target distribution which the user is interested in, and it can be expressed as

$$(1-\alpha)p_t(x) + \alpha p_d(x) = p_{\bar{d}}(x), \tag{3.1}$$

where $\alpha \in [0,\ 1]$. Our method, DSGAN, aims to learn p_g such that $p_g = p_t$. Note that if the support set of p_d belongs to that of $p_{\bar{d}}$, then there exists at least an α such that the equality in (3.1) holds. However, even though the equality does not hold, intuitively, DSGAN tries to learn p_g such that $p_g(x) \sim \dfrac{p_{\bar{d}}(x) - \alpha p_d(x)}{1-\alpha}$ with the constraint $p_g(x) \geq 0$. In other words, the generator is going to output samples located in high-density areas of $p_{\bar{d}} - \alpha p_d$. Furthermore, we show that DSGAN can learn p_g, whose support set is the difference between those of $p_{\bar{d}}$ and p_d in Proposition 2.

At first, we formulate the generator and discriminator in GANs. The inputs z of the generator are drawn from $p_z\left(z\right)$ in an M-dimensional space. The generator function $G(z;\theta_g) : \mathbb{R}^M \to \mathbb{R}^N$ represents a mapping to data space, where G is a differentiable function with parameters θ_g. The discriminator is defined as $D\left(x;\theta_d\right) : \mathbb{R}^N \to [0,\ 1]$ that

doi:10.6342/NTU201901502

outputs a single scalar. $D(x)$ can be considered as the probability that x belongs to a class of real data.

Similar to traditional GAN, we train D to distinguish the real data from the fake data sampled from G. Meanwhile, G is trained to produce realistic data as possible to mislead D. But, in DSGAN, the definitions of "real data" and "fake data" are different from those in traditional GAN. The samples from $p_{\bar{d}}$ are considered as real but those from the mixture distribution between p_d and p_g are considered as fake. The objective function is defined as follows:

$$\begin{aligned} V(G,D) := \mathbb{E}_{x\sim p_{\bar{d}}(x)}\left[\log D(x)\right] + (1-\alpha)\mathbb{E}_{z\sim p_z(z)}\left[\log\left(1-D\left(G\left(z\right)\right)\right)\right] + \\ \alpha\mathbb{E}_{x\sim p_d(x)}\left[\log\left(1-D(x)\right)\right]. \end{aligned} \tag{3.2}$$

We optimize (3.2) through a min-max game between G and D, that is,

$$\min_G \max_D V(G,D).$$

During the training procedure, an iterative approach like traditional GAN is to alternate between k steps of training D and one step of training G. In practice, minibatch stochastic gradient descent via back propagation is used to update θ_d and θ_g. In other words, for each of p_g, p_d and $p_{\bar{d}}$, m sample are required for computing gradients, where m is the number of samples in a minibatch. The training procedure is illustrated in Algorithm 1. DSGAN suffers from the same drawbacks with traditional GAN (*e.g.*, mode collapse, overfitting, and strong discriminator) such that the generator gradient vanishes. There are literature [4, 24, 29] focusing on dealing with the above problems, and such ideas can be readily combined into DSGAN.

[3] and [30] proposed the similar objective function like (3.2). Their goal is to learn the conditional distribution of training data. Nevertheless, we aim to learn the target distribution p_t in Eq. (3.1), not the training data distribution.

doi:10.6342/NTU201901502

Algorithm 1 The training procedure of DSGAN using minibatch stochastic gradient descent. k is the number of steps applied to discriminator. α is the ratio between p_g and p_d in the mixture distribution. We used $k = 1$ and $\alpha = 0.8$ in experiments.

```
01. for number of training iterations do
02.     for $k$ steps do
03.         Sample minibatch of $m$ noise samples $z^{(1)}, ..., z^{(m)}$ from $p_g(z)$.
04.         Sample minibatch of $m$ samples $x_d^{(1)}, ..., x_d^{(m)}$ from $p_d(x)$.
05.         Sample minibatch of $m$ samples $x_{\bar{d}}^{(1)}, ..., x_{\bar{d}}^{(m)}$ from $p_{\bar{d}}(x)$.
06.         Update the discriminator by ascending its stochastic gradient:
            $$\nabla_{\theta_d} \left[ \frac{1}{m} \sum_{i=1}^{m} \log D\left(x_d^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) + \log\left(1 - D\left(x_{\bar{d}}^{(i)}\right)\right) \right]$$
07.     end for
08.     Sample minibatch of $m$ noise samples $z^{(1)}, ..., z^{(m)}$ from $p_g(z)$.
09.     Update the generator by descending its stochastic gradient:
        $$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \left[\log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$$
10. end for
```

3.2 Case Study on Synthetic Data and MNIST

3.2.1 Case Study on Various Unseen Data Generation

To get more intuitive understanding about DSGAN, we conduct several case studies on 2D synthetic datasets and MNIST. $\alpha = 0.8$ in Eq. (3.1) is used.

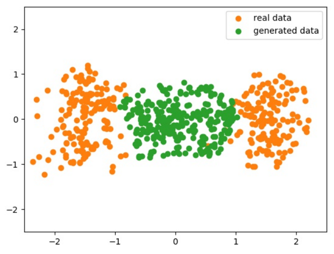


Figure 3.1: Complement points (in Green) between 2 circles (in Orange).

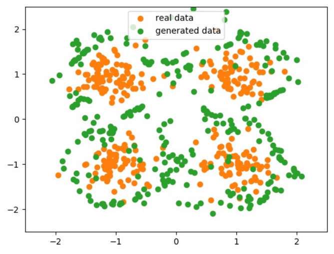


Figure 3.2: Boundary points (in Green) among 4 circles (in Orange).

Complement samples generation Fig. 3.1 illustrates that DSGAN is able to generate complement samples between 2 circles. Given the density function of the 2 circles as

doi:10.6342/NTU201901502

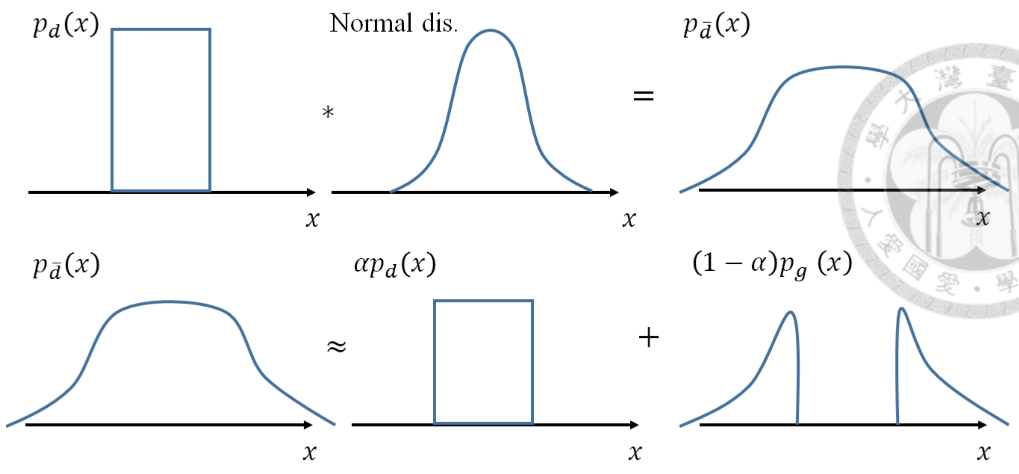


Figure 3.3: The illustration about generating unseen data in boundary around training data. First, the convolution of p_d and normal distribution makes the density on boundary be no longer zero. Second, we seek p_g such that Eq. (3.1) holds, where the support set of p_g is approximated by the difference of those between $p_{\bar{d}}$ and of p_d.

p_d, we assign samples drawn from $p_{\bar{d}}$ as the linear combinations of 2 circles. Then, by applying DSGAN, we achieve our goal to generate complement samples. In fact, this kind of unseen data is used in semi-supervised learning.

Boundary samples generation Fig. 3.2 illustrates that DSGAN generates boundary points among 4 circles. This kind of unseen data is used in novelty detection. In this case, we assign p_d and $p_{\bar{d}}$ as "the density function of 4 circles" and "the convolution of p_d and the normal distribution," respectively. The intuition of our idea is also illustrated by an 1D example in Fig. 3.3.

Difference-set generation We also validate DSGAN on high dimensional dataset such as MNIST. In this example, we define p_d to be the distribution of digit "1" and $p_{\bar{d}}$ to be the distribution containing both digits "1" and "7". Since the density $p_d(x)$ is high when x is digit "1," the generator is prone to output digit "7" with high probability. The illustration of difference-set generation is demonstrated in Fig. 3.4 and 3.5.

From the above results, we can observe two properties of generator distribution p_g: i) the higher density of $p_d(x)$, the lower density of $p_g(x)$; ii) p_g prefers to output samples from high-density areas of $p_{\bar{d}}(x) - \alpha p_d(x)$.

doi:10.6342/NTU201901502

In the next section, we will show that the objective function is equivalent to minimizing the Jensen-Shannon divergence between the mixture distribution (p_d and p_g) and $p_{\bar{d}}$ if G and D are given enough capacity.

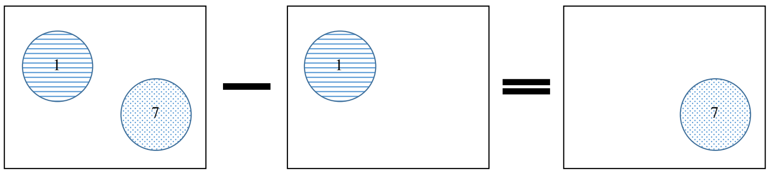


Figure 3.4: Illustration of difference-set seeking in MNIST.

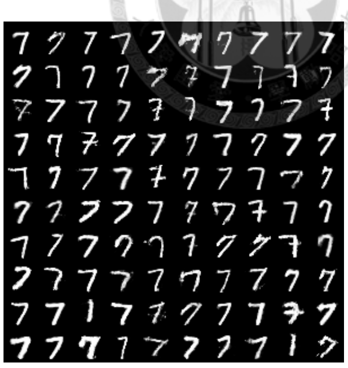


Figure 3.5: DSGAN learns the difference between two sets.

3.3 Discussions about the objective function of DSGAN

There are two main issues in 3.1. The first is that how the α influence the learned distribution p_g. From the objective function, one can imagine that the larger α will reduce the overlap between p_d and p_g. However, will the p_g be really far away from p_d if α is close to 1? Depending on the design of $p_{\bar{d}}$, the answer can be yes or no. Second, in some cases, it is possible that p_d is not fully contained in $p_{\bar{d}}$. In other words, $\dfrac{p_{\bar{d}}(x) - \alpha p_d(x)}{1-\alpha}$ can be negative for some x when α is large enough. In this section, we are going to demonstrate that the negative part will not influence the learning of generator.

We will discuss about the issues using Fig. 3.6.

The influence of α In Fig. 3.6, the overlapped area between $p_{\bar{d}}$ and p_d is 0.5 unit (let the area of p_d is 1 unit), so $\alpha = 0.5$ is the smallest choice to let p_g is disjoint to p_d. As the $\alpha = 0.8$, the generated points locate at the place which has a gap to yellow points. In theory, the result of $\alpha = 0.8$ should be the same as that of $\alpha = 0.5$, since the discriminator should give the whole area which is outside p_d but inside $p_{\bar{d}}$ same score. However, due to the continuity of the discriminator (continuity is the key point to make GAN work, such as

WGAN), the score of the area just beside the p_d is lower than where is far from it, when one has larger α. Because of this reason, the p_g tend to keep away from p_d. At α equals to 0.95, we can find that p_g still locate inside $p_{\bar{d}}$. In $\frac{p_{\bar{d}}(x) - \alpha p_d(x)}{1-\alpha}$, there must be some points making the expression are positive (when $p_{\bar{d}} \neq p_d$), due to $\int_x p(x)dx = 1$. Therefore, p_g will generate such points. In this example, one can observe that p_g is bounded by $p_{\bar{d}}$. As the support of $p_{\bar{d}}$ is much larger than which of p_d, excessive α will let p_g stay away from p_d. On the other side, one can use smaller support of $p_{\bar{d}}$ to make p_g close to p_d.

Negative density One can figure out that p_d is not fully contained in $p_{\bar{d}}$ in this example, while the rectangle which is bounded by $x <= 0$, $x >= -1$, $y >= -0.8$ and $y <= 0.8$, are in p_d but not in $p_{\bar{d}}$. However, in the case which $\alpha = 0.5$, we notice that the generator still generates perfect difference set even though there are some deficient places with negative density. This can be explained through the intrinsic property of the discriminator. By 3.2, the discriminator's output of the points with negative density (x') tend to be 0. Observing the first term in 3.2, since x' is not in $p_{\bar{d}}$, then no gradients will lead $D(x')$ to arise to 1. $D(x') = 0$ meets our goal that p_g don't overlap with p_d. Therefore, although there exists the negative density, the objective will not be effected.

3.4 Tricks for Stable Training

We provide a trick to stabilize the training procedure by reformulating the objective function. Specifically, $V(G, D)$ in (3.2) is reformulated as:

$$\begin{aligned} V(G,D) &= \int_x p_{\bar{d}}(x) \log(D(x)) + ((1-\alpha)p_g(x) + \alpha p_d(x)) \log(1 - D(x))\, dx \\ &= \mathbb{E}_{x\sim p_{\bar{d}}(x)} [\log D(x)] + \mathbb{E}_{x\sim(1-\alpha)p_g(x)+\alpha\sim p_d(x)} [\log(1 - D(x))] . \end{aligned} \tag{3.3}$$

Instead of sampling a mini-batch of m samples from p_z and p_d in Algorithm 1, $(1-\alpha)m$ and αm samples from both distributions are required, respectively. The computation cost in training can be reduced due to fewer samples. Furthermore, although (3.3) is equivalent to (3.2) in theory, we find that the training using (3.3) achieves better performance

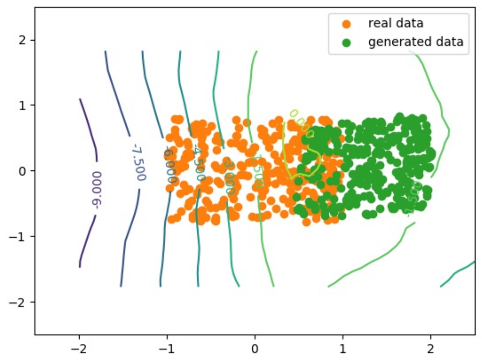
doi:10.6342/NTU201901502

than using (3.2) via empirical validation in Table 6.1. We conjecture that the equivalence between (3.3) and (3.2) is based on the linearity of expectation, but mini-batch stochastic gradient descent in practical training may lead to the different outcomes.

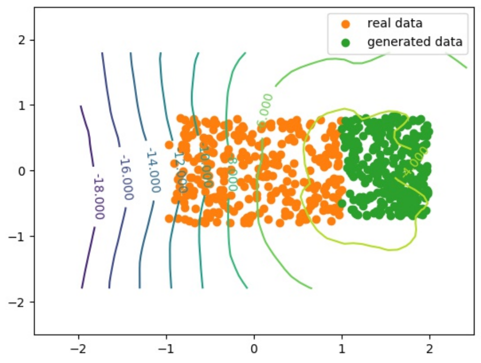
3.5 Appendix: More Results for Case Study

Additional results for boundary samples generation and difference set generation are presented in Fig. 3.7 and Fig. 3.8, respectively.

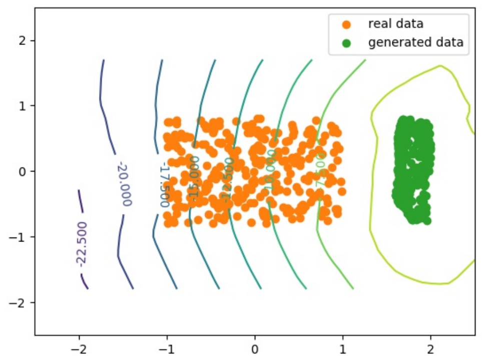
doi:10.6342/NTU201901502



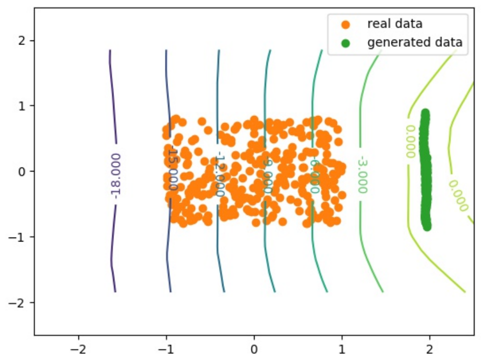
(a) $\alpha = 0.30$



(b) $\alpha = 0.50$



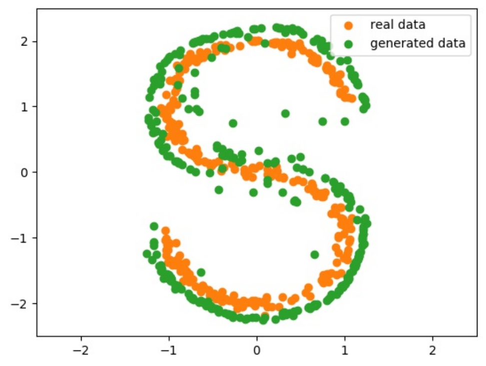
(c) $\alpha = 0.80$



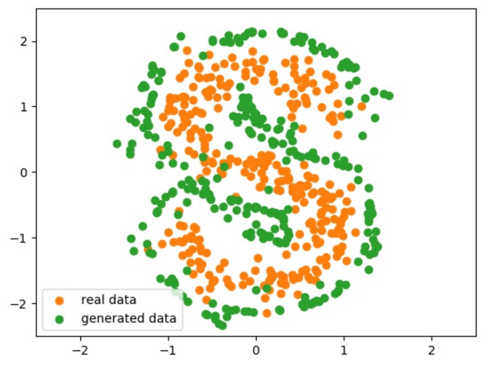
(d) $\alpha = 0.95$

Figure 3.6: Demonstrate the influence of α on the synthetic dataset. In this example, p_d is the orange rectangle (bounded by $x <= 1$, $x >= -1$, $y >= -0.8$ and $y <= 0.8$), and $p_{\bar{d}}$ is the rectangle which is shifted p_d right by 1 unit (not appear in the figures). We can observe that p_g is farther away from p_d (green points) when α increases. When α is 0.5, p_g learns perfect difference between $p_{\bar{d}}$ and p_d. When α is 0.95, p_g generates the rightmost points of $p_{\bar{d}}$. The contour is the output of the discriminator, the place with higher score the generator going. Note that the outputs of the discriminator are not restricted in $[0, 1]$, because we use WGAN's structure in this experiment.

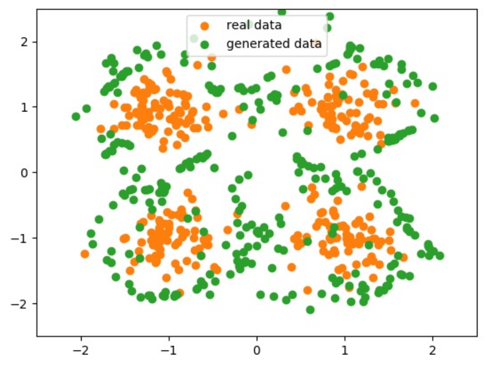
doi:10.6342/NTU201901502



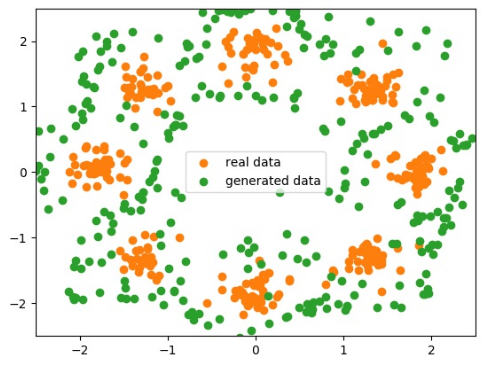
(a) S shape with compact data points. $\alpha = 0.9$.



(b) S shape with scattering data points. $\alpha = 0.9$.



(c) 4 Gaussians. $\alpha = 0.9$.



(d) 8 Gaussians. $\alpha = 0.8$.

Figure 3.7: Extra 2D results for boundary sample generation. The orange points are data points, and the green points are generated points.

doi:10.6342/NTU201901502

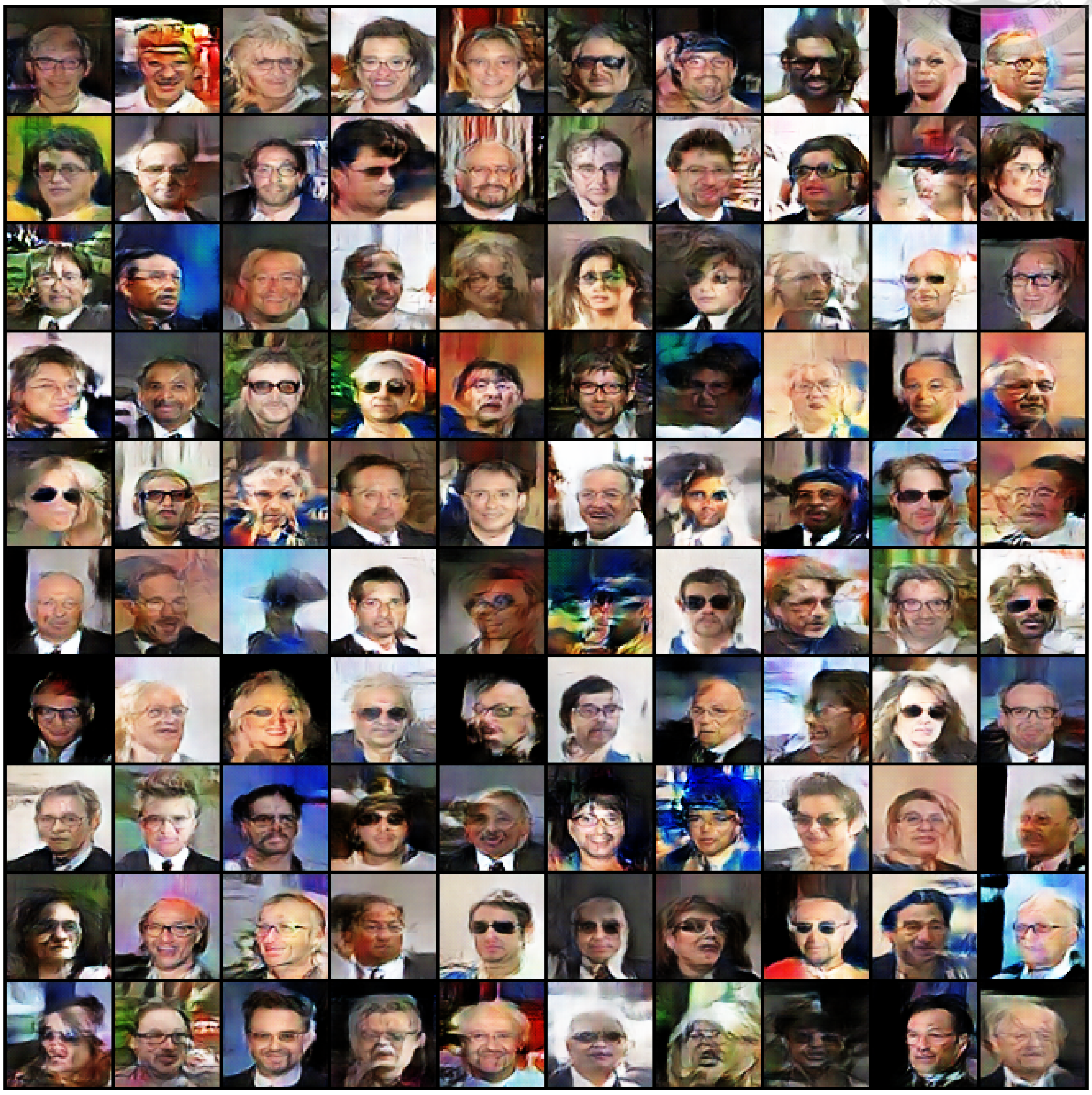


Figure 3.8: Difference set generation for CelebA dataset ([1]). $p_{\bar{d}}$ is 20000 images from CelebA dataset. In $p_{\bar{d}}$, 1000 images are humans with glasses while others are ones without glasses. p_d all contains human wearing glasses, and its size is 19000. In this case, our generator successfully learned to produce images which are human with glasses. Note that $\alpha = 0.95$.

doi:10.6342/NTU201901502

Chapter 4

Theoretical Results

There are two assumptions for subsequent proofs. First, in a nonparametric setting, we assume both generator and discriminator have infinite capacity. Second, p_g is defined as the distribution of the samples drawn from $G(z)$ under $z \sim p_z$. We will first show the optimal discriminator given G and then show that minimizing $V(G, D)$ via G given the optimal discriminator is equivalent to minimizing the Jensen-Shannon divergence between $(1-\alpha)p_g + \alpha p_d$ and $p_{\bar{d}}$.

Proposition 1. *For G being fixed, the optimal discriminator D is*

$$D_G^*(x) = \frac{p_d(x)}{p_d(x) + (1-\alpha)p_g(x) + \alpha p_d(x)}.$$

Proof. Given any generator G, the training criterion for the discriminator D is to maximize the quantity $V(G, D)$:

$$\begin{aligned} V(G,D) &= \int_x p_{\bar{d}}(x)\log\left(D\left(x\right)\right)dx + (1-\alpha)\int_z p_z(z)\log\left(1 - D\left(G\left(z\right)\right)\right)dz \\ &\quad + \alpha\int_x p_d(x)\log\left(1 - D\left(x\right)\right)dx \\ &= \int_x p_{\bar{d}}(x)\log\left(D\left(x\right)\right)dx + (1-\alpha)\int_x p_g(x)\log\left(1 - D\left(x\right)\right)dz \\ &\quad + \alpha\int_x p_d(x)\log\left(1 - D\left(x\right)\right)dx \\ &= \int_x p_{\bar{d}}(x)\log\left(D\left(x\right)\right) + \left((1-\alpha)p_g(x) + \alpha p_d(x)\right)\log\left(1 - D\left(x\right)\right)dx. \end{aligned}$$

doi:10.6342/NTU201901502

For any $(a, b) \in \mathbb{R}^2 \backslash \{0,\ 0\}$, the function $a \log{(y)} + b \log{(1-y)}$ achieves its maximum in $[0,\ 1]$ at $y = \frac{a}{a+b}$. The discriminator only needs to be defined within $\text{Supp}(p_{\bar{d}}) \bigcup \text{Supp}(p_d) \bigcup \text{Supp}(p_g)$. We complete this proof. □

Moreover, D can be considered to discriminate between samples from $p_{\bar{d}}$ and those from $((1-\alpha)p_g(x) + \alpha p_d(x))$. By replacing the optimal discriminator into $V(G, D)$, we obtain

$$\begin{aligned}
C(G) &= \max_D V(G, D) \\
&= \mathbb{E}_{x \sim p_{\bar{d}}(x)} \left[\log D^*_G(x)\right] + (1-\alpha)\mathbb{E}_{z \sim p_z(z)} \left[\log{(1 - D^*_G\left(G\left(z\right)\right))}\right] \\
&\quad + \alpha \mathbb{E}_{x \sim p_d(x)} \left[\log{(1 - D^*_G(x))}\right] \\
&= \mathbb{E}_{x \sim p_{\bar{d}}(x)} \left[\log D^*_G(x)\right] + \mathbb{E}_{x \sim p^*(x)} \left[\log{(1 - D^*_G\left(x\right))}\right] \\
&= \mathbb{E}_{x \sim p_{\bar{d}}(x)} \left[\log \frac{p_{\bar{d}}(x)}{p_{\bar{d}}(x) + (1-\alpha)p_g(x) + \alpha p_d(x)}\right] \\
&\quad + \mathbb{E}_{x \sim p^*(x)} \left[\log \frac{(1-\alpha)p_g(x) + \alpha p_d(x)}{p_{\bar{d}}(x) + (1-\alpha)p_g(x) + \alpha p_d(x)}\right],
\end{aligned} \tag{4.1}$$

where $p^*(x) = (1-\alpha)p_g(x) + \alpha p_d(x)$ and the third equality holds because of the linearity of expectation.

Actually, the results so far show the optimal solution of D given G being fixed in (4.1). Now, the next step is to find the optimal G with D^*_G being fixed.

Theorem 1. *The global minimum of $C(G)$ is achieved if and only if $(1-\alpha)p_g(x) + \alpha p_d(x) = p_{\bar{d}(x)}$ for all x's. At that point, $C(G)$ achieves the value $-\log 4$.*

doi:10.6342/NTU201901502

Proof. We start from

$$\begin{aligned}(4.1) = &-\log(4)\\ &+\mathbb{E}_{x\sim p_{\bar{d}}(x)}\left[\log\frac{2p_{\bar{d}}(x)}{p_{\bar{d}}(x)+(1-\alpha)p_g(x)+\alpha p_d(x)}\right]\\ &+\mathbb{E}_{x\sim p^*(x)}\left[\log\frac{2\left((1-\alpha)p_g(x)+\alpha p_d(x)\right)}{p_{\bar{d}}(x)+(1-\alpha)p_g(x)+\alpha p_d(x)}\right]\\ =&-\log(4)+\mathrm{KL}\left(p_{\bar{d}}\,\middle\|\,\frac{p_{\bar{d}}+(1-\alpha)p_g+\alpha p_d}{2}\right)\\ &+\mathrm{KL}\left((1-\alpha)p_g(x)+\alpha p_d\,\middle\|\,\frac{p_{\bar{d}}+(1-\alpha)p_g+\alpha p_d}{2}\right)\\ =&-\log(4)+2\,\mathrm{JSD}\left(p_{\bar{d}}\,\|\,(1-\alpha)p_g+\alpha p_d\right),\end{aligned}$$

where $p^*(x) = (1-\alpha)p_g(x)+\alpha p_d(x)$, KL is the Kullback-Leibler divergence and JSD is the Jensen-Shannon divergence. The JSD returns the minimal value, which is 0, iff both distributions are the same, namely $p_{\bar{d}} = (1-\alpha)p_g+\alpha p_d$. Because $p_g(x)$'s are always non-negative, it should be noted both distributions are the same only if $\alpha p_d(x) \le p_{\bar{d}}(x)$ for all x's. We complete this proof. □

Note that $(1-\alpha)p_g(x)+\alpha p_d(x) = p_{\bar{d}(x)}$ may not hold if $\alpha p_d(x) > p_{\bar{d}(x)}$. But, DSGAN still works based on two facts: i) given D, $V(G,D)$ is a convex function in p_g and ii) due to $\int_x p_g(x)dx = 1$, the set collecting all feasible solutions of p_g is convex. In other words, there always exists a global minimum of $V(G,D)$ given D, but it may not be $-\log(4)$. In the following, we show that the support set of p_g is contained within the difference of support sets between $p_{\bar{d}}$ and p_d while achieving the global minimum such that we can generate the desired p_g by designing appropriate $p_{\bar{d}}$.

Proposition 2. *Suppose $\alpha p_d(x) \ge p_{\bar{d}}(x)$ for all $x \in \mathrm{Supp}(p_d)$ and all density functions $p_d(x)$, $p_{\bar{d}}(x)$ and $p_g(x)$ are continuous. If the global minimum of $C(G)$ is achieved, then*

$$\mathrm{Supp}\left(p_g\right) \subseteq \mathrm{Supp}\left(p_{\bar{d}}\right) - \mathrm{Supp}(p_d).$$

doi:10.6342/NTU201901502

Proof. Recall

$$\begin{aligned} C(G) &= \int_x p_{\bar{d}}(x) \log\left(\frac{p_{\bar{d}}(x)}{p_{\bar{d}}(x) + (1-\alpha)p_g(x) + \alpha p_d(x)}\right) \\ &\quad + p^*(x) \log\left(\frac{(1-\alpha)p_g(x) + \alpha p_d(x)}{p_{\bar{d}}(x) + (1-\alpha)p_g(x) + \alpha p_d(x)}\right) dx \\ &= \int_x S(p_g; x) dx \\ &= \int_{x \in \text{Supp}(p_{\bar{d}}) - \text{Supp}(p_d)} S(p_g; x) dx + \int_{x \in \text{Supp}(p_d)} S(p_g; x) dx \end{aligned}$$

$S(p_g; x)$ is to simplify the notations inside the integral. For any x, $S(p_g; x)$ in $p_g(x)$ is non-increasing and $S(p_g; x) \le 0$ always holds. Specifically, $S(p_g; x)$ is decreasing along the increase of $p_g(x)$ if $p_{\bar{d}}(x) > 0$; $S(p_g; x)$ attains the maximum value, zero, for any $p_g(x)$ if $p_{\bar{d}}(x) = 0$. Since DSGAN aims to minimize $C(G)$ with the constraint $\int_x p_g(d) dx = 1$, the solution attaining the global minima must satisfy $p_g(x) = 0$ if $p_{\bar{d}}(x) = 0$; otherwise, there exists another solution with smaller value of $C(G)$. Thus, Supp $(p_g) \subseteq$ Supp $(p_{\bar{d}})$.

Furthermore, $T(p_g; x) = \frac{\partial S(p_g; x)}{\partial p_g(x)} = \log\left(\frac{(1-\alpha)p_g(x) + \alpha p_d(x)}{p_{\bar{d}}(x) + (1-\alpha)p_g(x) + \alpha p_d(x)}\right)$, which is expected to be as small as possible to minimize $C(G)$, is increasing on $p_g(x)$ and converges to 0. Then, we show that $T(p_g; x)$ for $x \in$ Supp$(p_{\bar{d}}) \bigcap$ Supp(p_d) is always larger than that for $x \in$ Supp$(p_{\bar{d}}) -$ Supp(p_d) for all p_g. Specifically,

1. When $x \in$ Supp$(p_{\bar{d}}) \bigcap$ Supp(p_d), $T(p_g; x) \ge \log \frac{1}{2}$ always holds due to the assumption $\alpha p_d(x) \ge p_{\bar{d}}(x)$.

2. When $x \in$ Supp$(p_{\bar{d}}) -$ Supp(p_d), $T(p_g; x) < \log \frac{1}{2}$ for all $p_g(x)$'s satisfying $(1 - \alpha)p_g(x) \le p_{\bar{d}(x)}$.

Thus, the minimizer prefers $p_g(x) > 0$ for $x \in$ Supp$(p_{\bar{d}}) -$ Supp(p_d) and $(1-\alpha)p_g(x) \le p_{\bar{d}(x)}$. We check whether there exists a solution p_g such that $(1-\alpha)p_g(x) \le p_{\bar{d}(x)}$ and $\int_{x \in \text{Supp}(p_{\bar{d}}) - \text{Supp}(p_d)} p_g(d) dx = 1$, implying $p_g(x) = 0$ for $x \in$ Supp$(p_{\bar{d}}) \bigcap$ Supp(p_d).

doi:10.6342/NTU201901502

Based on the following expression,

$$\begin{aligned}
&\int_{x\in \text{Supp}(p_{\bar{d}})-\text{Supp}(p_d)} p_{\bar{d}}(x)dx \;+\; \int_{x\in \text{Supp}(p_d)} p_{\bar{d}}(x)dx = 1\\
\Rightarrow &\int_{x\in \text{Supp}(p_{\bar{d}})-\text{Supp}(p_d)} p_{\bar{d}}(x)dx \;\geq\; 1-\int_{x\in \text{Supp}(p_d)} \alpha p_d(x)dx\\
\Rightarrow &\int_{x\in \text{Supp}(p_{\bar{d}})-\text{Supp}(p_d)} p_{\bar{d}}(x)dx \;\geq\; 1-\alpha\\
\Rightarrow &\int_{x\in \text{Supp}(p_{\bar{d}})-\text{Supp}(p_d)} p_{\bar{d}}(x)dx \;\geq\; \int_{x\in \text{Supp}(p_{\bar{d}})-\text{Supp}(p_d)} (1-\alpha)p_g(x)dx,
\end{aligned}$$

the last inequality implies that there must exist a feasible solution. We complete this proof.

□

In sum, the generator is prone to output samples located in high-density areas of $p_{\bar{d}} - \alpha p_d$.

Another concern is the convergence of Algorithm 1.

Proposition 3. *The discriminator reaches its optimal value given G in Algorithm 1, and p_g is updated by minimizing*

$$\mathbb{E}_{x\sim p_{\bar{d}}(x)}\left[\log D_G^*(x)\right] + \mathbb{E}_{x\sim p^*(x)}\left[\log\left(1-D_G^*(x)\right)\right].$$

If G and D have enough capacity, then p_g converges to $\underset{p_g}{\operatorname{argmin}}\, \text{JSD}\left(p_{\bar{d}} \parallel (1-\alpha)p_g + \alpha p_d\right)$.

Proof. Consider $V(G,D) = U(p_g, D)$ as a function of p_g. By the proof idea of Proposition 2 in [14], if $f(x) = \sup_{\alpha\in\mathcal{A}} f_\alpha(x)$ and $f_\alpha(x)$ is convex in x for every α, then $\partial f_\beta(x) \in \partial f$ if $\beta = \text{argsup}_{\alpha\in\mathcal{A}} f_\alpha(x)$. In other words, if $\sup_D V(G,D)$ is convex in p_g, the subderivatives of $\sup_D V(G,D)$ includes the derivative of the function at the point, where the maximum is attained, implying the convergence with sufficiently small updates of p_g. We complete this proof. □

doi:10.6342/NTU201901502

doi:10.6342/NTU201901502

Chapter 5

Applications

DSGAN have been applied to three problems: semi-supervised learning, robustness enhancement of deep networks and novelty detection. As for semi-supervised learning, DSGAN acts as a "bad generator," which creates complement samples in the feature space of real data, while DSGAN generates adversarial examples located in the low-density areas of training data for robustness enhancement. For novelty detection, DSGAN generates samples (unseen data) as the boundary points around training data.

5.1 Semi-Supervised Learning

Semi-supervised learning (SSL) is a kind of learning model with the use of a small number of labeled data and a large amount of unlabeled data. The existing SSL methods based on generative model (*e.g.,* VAE [31] and GAN [4]) obtain good empirical results. [5] theoretically shows that good semi-supervised learning requires a bad GAN with the objective function:

$$\begin{aligned}\max_D \mathbb{E}_{x,y\sim\mathcal{L}} \log P_D\left(y \mid x, y \leq K\right) + \mathbb{E}_{x\sim p_d(x)} \log P_D\left(y \leq K \mid x\right) \\ + \mathbb{E}_{x\sim p_g(x)} \log P_g\left(K+1 \mid x\right),\end{aligned} \tag{5.1}$$

where (x, y) denotes a pair of data and its corresponding label, $\{1, 2, \ldots, K\}$ denotes the label space for classification, and $\mathcal{L} = \{(x, y)\}$ is the label dataset. Moreover, in the

doi:10.6342/NTU201901502

semi-supervised settings, p_d in (5.1) is the distribution of unlabeled data. Note that the discriminator D in GAN also plays the role of classifier. If the generator distribution exactly matches the real data distribution (*i.e.*, $p_g = p_d$), then the classifier trained by the objective function (5.1) with the unlabeled data cannot have better performance than that trained by supervised learning with the objective function:

$$\max_{D} \mathbb{E}_{x,y\sim\mathcal{L}} \log P_D\left(y \mid x, y \leq K\right). \tag{5.2}$$

On the contrary, the generator is preferred to generate complement samples, which lie on the low-density area of p_d. Under some mild assumptions, those complement samples help D to learn correct decision boundaries in the low-density area because the probabilities of true classes are forced to be low on out-of-distribution areas.

The complement samples in [5] are complicate to produce. We will demonstrate that DSGAN is easy to generate complement samples in Sec. 6.

5.2 Robustness Enhancement of Deep Networks

Deep neural networks have impacted on our daily life. Neural networks, however, are vulnerable to adversarial examples, as evidenced in recent studies [32, 33]. Thus, there has been significant interest in how to enhance the robustness of neural networks. Unfortunately, if the adversary has full access to the network, namely white-box attack, a complete defense strategy has not yet been found.

In the research papers, it's not hard to see a well trained deep neural model reaching more than 90% accuracy on a classification task. It seems that the machine beats human in recognition nowadays. Nonetheless, the machine vision is surprisingly fragile. For most of the inputs which are classified correctly, one can constructed an adversarial example by adding a specific noise to original input, to make the predicted results totally wrong. In most of the time, the original image and its adversarial example are undistinguished for human, such as Fig. 5.1.

One can create an adversarial example through 5.3, where p is 2 or inf, typically. Intu-

doi:10.6342/NTU201901502

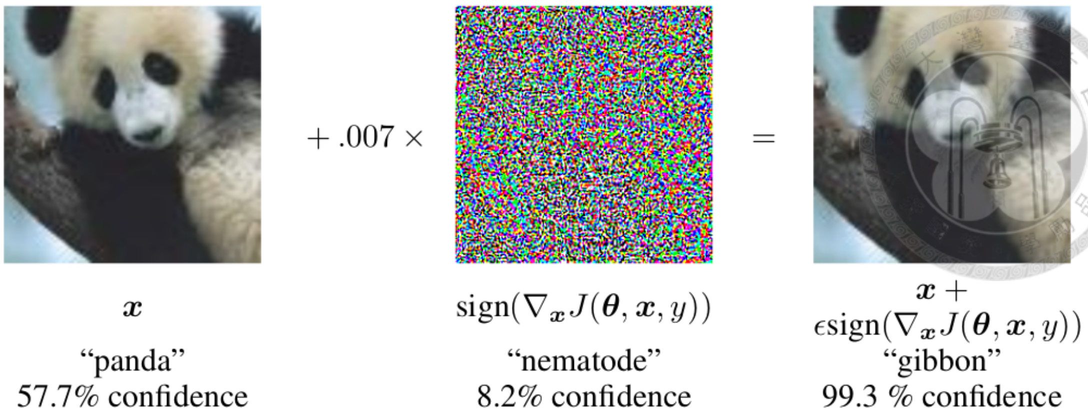

+ .007 ×
=
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 5.1: Demonstration for the adversarial example. Adding a special noise to the panda image can change the prediction of the model to "gibbon". Moreover, the noise on the adversarial example is unperceivable for human.

itively, optimizing the equation is to find a perturbation for the input to let the model less likely to predict the correct label.

$$\max_{\|\delta\|_p \leq \epsilon} \ell\left(x+\delta; y; C_\theta\right) \tag{5.3}$$

[34] surveyed the state-of-the-art defense strategies and showed that adversarial training [35] is more robust than other strategies. Given adversarial examples and a trained classifier C parameterized with θ and a loss function $\ell\left(x; y; C_\theta\right)$, adversarial training solves a min-max game, where the first step is to find adversarial examples within ϵ-ball for maximizing the loss, and the second step is to train the model for minimizing the loss. Specifically, the objective in [35] is

$$\operatorname*{argmin}_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{L}}\left[\max_{\delta\in[-\epsilon,\ \epsilon]^N} \ell\left(x+\delta; y; C_\theta\right)\right]. \tag{5.4}$$

The authors used projected gradient descent (PGD) to find adversarial examples by maximizing the inner optimization.

Adversarial training requires the pretrained classifiers to calculate PGD such that adversarial examples may be effective on specific classifiers. But, our DSGAN generates unseen data in the low-density area, which include adversarial examples, because [36] pointed out that adversarial examples frequently locate in the low-density area. In other

doi:10.6342/NTU201901502

words, the generated data in DSGAN are more universal but less effective for specific classifiers. In Sec. 6.2, we enhance the robustness of deep learning networks in a semi-supervised manner, which use generated samples of DSGAN to fine-tune C_θ. ϵ-ball in terms of ℓ_2 or ℓ_{inf} can be intuitively incorporated into the generation of adversarial examples.

5.3 Novelty Detection

Novelty detection determines if a query example is from one seen class. If the samples of one seen class are considered as positive data, then this difficulty is the absence of negative data in the training phase such that supervised learning cannot work. To overcome this problem, one of classical methods is the One-Class SVM (OCSVM) [37] that only requires positive data as training inputs. However, OCSVM often suffers from the curse of dimensionality due to bad computational scalability.

Recently, novelty detection has made a great progress with the advent of deep leaning. [38][39] focus on learning a representative latent space for the one seen class. When testing, the query image is projected onto the learned latent space. Then, the difference between the query image and its inverse image (reconstruction) is measured. In other words, all we need is to train an encoder for projection and a decoder for reconstruction. Under the circumstance, autoencoder (AE) usually is adopted to learn both encoder and decoder [38][10]. Let Enc$(\cdot)$ be the encoder and Dec$(\cdot)$ be the decoder, respectively. The loss function of AE is defined as:

$$\min_{\text{Enc,Dec}} \mathbb{E}_{x\sim p_{pos}(x)} \left[||x - \text{Dec}(\text{Enc}(x))||_2^2 \right], \tag{5.5}$$

where p_{pos} is the distribution of one seen class. After training, a query example x_{test} is classified as the seen class if

$$||x_{test} - \text{Dec}(\text{Enc}(x_{test}))||_2^2 \leq \tau, \tag{5.6}$$

doi:10.6342/NTU201901502

where $\tau \in \mathbb{R}^+$ plays the trade-off between true positive rate and false positive rate. However, (5.6) is based on two assumptions: (1) the positive samples from one seen class should have lower reconstruction error; (2) the AE (or latent space) cannot describe negative examples from unseen classes well, leading to a relatively higher reconstruction error. In general, the first assumption inherently holds when both testing and training data come from the same seen class. However, [38][10] observed that the assumption (2) does not hold at all times because the loss function in (5.5) does not include a loss term to enforce negative data to have high reconstruction error.

To make the assumption (2) hold, given positive data as training inputs, we propose using DSGAN to generate negative examples in the latent space in Sec. 6.3. Then, the loss function of AE is modified to enforce negative data to have high reconstruction error.

doi:10.6342/NTU201901502

doi:10.6342/NTU201901502

Chapter 6

Experiments

In this section, we demonstrate the empirical results about semi-supervised learning, robustness enhancement of deep networks and novelty detection in Sec. 6.1, Sec. 6.2 and Sec. 6.3, respectively.

Note that, the training procedure of DSGAN can be improved by other extensions of GANs such as WGAN [25], WGAN-GP [26], EBGAN [40], LSGAN [41] and etc. WGAN-GP was adopted in our method such that DSGAN is stable in training and suffers less mode collapse.

6.1 DSGAN in Semi-Supervised Learning

Following the previous works, we apply the proposed DSGAN in semi-supervised learning on three benchmark datasets, including MNIST [42], SVHN [43], and CIFAR-10 [44].

We first introduce how DSGAN generates complement samples in the feature space. Specifically, [5] proved that if complement samples generated by G can satisfy the following two assumptions in (6.1) and (6.2):

$$\forall x \sim p_g(x), 0 > \max_{1 \le i \le K} w_i^T f(x) \text{ and } \forall x \sim p_d(x), 0 < \max_{1 \le i \le K} w_i^T f(x), \tag{6.1}$$

doi:10.6342/NTU201901502

where f is the feature extractor and w_i is the linear classifier for the i^{th} class and

$$\begin{aligned} &\forall x_1 \sim \mathcal{L}, x_2 \sim p_d(x), \exists x_g \sim p_g(x) \text{ s.t.} \\ &f(x_g) = \beta f(x_1) + (1-\beta) f(x_2) \text{ with } \beta \in [0,\ 1], \end{aligned} \tag{6.2}$$

then all unlabeled data will be classified correctly via the objective function (5.1). Specifically, (6.1) ensures that classifiers are capable of discriminating generated data from unlabeled data, and (6.2) is to make the decision boundary locate in low-density areas of p_d.

The assumption in (6.2) implies the complement samples have to be in the space created by linear combination of labeled and unlabeled data. Besides, they cannot fall into the real data distribution p_d due to the assumption (6.1). In order to let DSGAN generate such samples, we let the samples of $p_{\bar{d}}$ be the linear combination of those from $\mathcal{L}$ and p_d. Since $p_g(x) \approx \dfrac{p_{\bar{d}}(x) - \alpha p_d(x)}{1-\alpha}$, p_g will tend to match $p_{\bar{d}}$ while the term $-\alpha p_d$ ensures that samples from p_g do not belong to p_d. Thus, p_g satisfies both assumptions in (6.1) and (6.2).

In practice, we parameterize f and all w_i's together as a neural network. The details of the experiments, including the network architectures, can be found in Appendix 6.1.3.

6.1.1 Datasets: MNIST, SVHN, and CIFAR-10

For evaluating the semi-supervised learning task, we used 60000/ 73257/ 50000 samples and 10000/ 26032/ 10000 samples from the MNIST/ SVHN/ CIFAR-10 datasets for training and testing, respectively. Due to the semi-supervised setting, we randomly chose 100/ 1000/ 4000 samples from the training samples as the MNIST/ SVHN/ CIFAR-10 labeled dataset, and the amount of labeled data for all classes are equal.

Our criterion to determine the hyperparameters is introduced in Appendix 6.1.3. We performed testing with 10/ 5/ 5 runs on MNIST/ SVHN/ CIFAR-10 based on the selected hyperparameters and randomly selected labeled dataset. Following [5], the results are recorded as the mean and standard deviation of the number of errors from each run.

doi:10.6342/NTU201901502

6.1.2 Main Results

First, the hyperparameters we chose are depicted in Table 6.3 in Appendix 6.1.3. Second, the results obtained from our DSGAN and the state-of-the-art methods on three benchmark datasets are depicted in Table 6.2. The effectiveness of applying the tricks in Sec. 3.4 is show in Table 6.1, and it validate the influence of the tricks.

Table 6.1: Semi-supervised learning results on MNIST whether to use the sampling tricks.

Methods	**MNIST (# errors)**
Our method w/o tricks	91.0 ± 7.0
Our method w/ tricks	82.7 ± 4.6

It can be observed that our results can compete with state-of-the-art methods on the three datasets. Note that the results of badGAN [5] were reproduced by the released codes of the authors.

In comparison with [5], our methods don't need to rely on an additional density estimation network PixelCNN++ [45]. Although PixelCNN++ is one of the best density estimation network, it cannot estimate the density in the feature space, which is dynamic during training. This drawback makes the models in [5] fail to fulfill the assumptions in their paper.

Moreover, it can also be observed in Table 6.2 that our results are comparable to the best record of badGAN [5] and are better than other approaches in MNIST and SVHN. In CIFAR-10, our method is only inferior to CT-GAN. It might not be a fair comparison since CT-GAN uses extra techniques, including temporal ensembling and data augmentation, but other methods do not.

doi:10.6342/NTU201901502

Table 6.2: Comparison of semi-supervised learning between our DSGAN and state-of-the-art methods: CatGAN [2], TripleGAN [3], FM [4], badGAN [5] and CT-GAN [6]. For a fair comparison, we only consider the GAN-based methods. $*$ indicates the use of the same architecture of classifier. † indicates a larger architecture of classifier. ‡ indicates the use of data augmentation. The results for MNIST are recorded in the number of errors while the others are in percentage of errors.

Methods	**MNIST**	**SVHN**	**CIFAR-10**
CatGAN	191 ± 10	-	19.58 ± 0.46
TripleGAN†	91 ± 58	5.77 ± 0.17	16.99 ± 0.36
FM*	93 ± 6.5	8.11 ± 1.3	18.63 ± 1.32
badGAN*	86.2 ± 13.2	4.48 ± 0.16	16.25 ± 0.33
CT-GAN‡	-	-	9.98 ± 0.21
Our method*	82.7 ± 4.6	4.88 ± 0.07	15.08 ± 0.24

6.1.3 Appendix: Experimental Details

Hyperparameters

The hyperparameters were chosen to make our generated samples consistent with the assumptions in (6.1) and (6.2). However, in practice, if we make all the samples produced by the generator following the assumption in (6.2), then the generated distribution is not close to the true distribution, even a large margin between them existing in most of the time, which is not what we desire. So, in our experiments, we make a concession that the percentage of generated samples, which accords with the assumption, is around 90%. To meet this objective, we tune the hyperparameters. Table 6.3 shows our setting of hyperparameters, where β is defined in (6.2).

Table 6.3: Hyperparameters in semi-supervised learning.

Hyperparameters	**MNIST**	**SVHN**	**CIFAR-10**
α	0.8	0.8	0.5
β	0.3	0.1	0.1

doi:10.6342/NTU201901502

Architecture

In order to fairly compare with other methods, our generators and classifiers for MNIST, SVHN, and CIFAR-10 are same as in [4] and [5]. However, different from previous works that have only a generator and a discriminator, we design an additional discriminator in the feature space, and its architecture is similar across all datasets with only the difference in the input dimensions. Following [5], we also define the feature space as the input space of the output layer of discriminators.

Compared to SVHN and CIFAR-10, MNIST is a simple dataset as it is only composed of fully connected layers. Batch normalization (BN) or weight normalization (WN) is used to every layer to stable training. Moreover, Gaussian noise is added before each layer in the classifier, as proposed in [46]. We find that the added Gaussian noise exhibits a positive effect for semi-supervised learning and keep to use it. The architecture is shown in Table 6.4.

Table 6.5 and Table 6.6 are models for SVHN and CIFAR-10, respectively, and these models are almost the same except for some implicit differences, *e.g.*, the number of convolutional filters and types of dropout. In these tables, given a dropping rate, "Dropout" is a normal dropout in that the elements of input tensor are randomly set to zero while Dropout2d is a dropout only applied on the channels to randomly zero all the elements.

Furthermore, the training procedure alternates between k steps of optimizing D and one step of optimizing G. We find that k in Algorithm 1 is a key role in the problem of mode collapse for different applications. For semi-supervised learning, we set $k = 1$ for all datasets.

6.2 DSGAN in Robustness Enhancement of Deep Networks

Our proposed DSGAN is capable of improving the robustness of deep models (classifiers) against adversarial examples. In the experiments, we mainly validate DSGAN on CIFAR-10, including natural images, which are easier to attack, compared with the MNIST dataset.

doi:10.6342/NTU201901502

Table 6.4: Network architectures for semi-supervised learning on MNIST. (GN: Gaussian noise)

Generator G	Discriminator D	Classifier C
Input $\boldsymbol{z} \in \mathbb{R}^{100}$ from unif(0, 1)	Input 28×28 gray image	Input 28×28 gray image
100×500 FC layer with BN	250×400 FC layer	GN, std = 0.3
Softplus	ReLU	784×1000 FC layer with WN ,ReLU
500×500 FC layer with BN	400×200 FC layer	GN, std = 0.5
Softplus	ReLU	1000×500 FC layer with WN, ReLU
500×784 FC layer with WN	200×100 FC layer	GN, std = 0.5
Sigmoid	ReLU	500×250 FC layer with WN, ReLU
	100×1 FC layer	GN, std = 0.5
		250×250 FC layer with WN, ReLU
		GN, std = 0.5
		250×250 FC layer with WN, ReLU
		250×10 FC layer with WN

Table 6.5: The architectures of generator and discriminator for semi-supervised learning on SVHN and CIFAR-10. N was set to 128 and 192 for SVHN and CIFAR-10, respectively.

Generator G	Discriminator D
Input $\boldsymbol{z} \in \mathbb{R}^{100}$ from unif(0, 1)	Input 32×32 RGB image
100×8192 FC layer with BN, ReLU	$N \times 400$ FC layer, ReLU
Reshape to $4 \times 4 \times 512$	400×200 FC layer, ReLU
5×5 conv. transpose 256 stride = 2 with BN, ReLU	200×100 FC layer, ReLU
5×5 conv. transpose 128 stride = 2 with BN, ReLU	100×1 FC layer
5×5 conv. transpose 3 stride = 2 with WN, Tanh	

Recall that the objective function (5.4) requires finding adversarial examples to maximize the classification error $\ell(\cdot)$. Adversarial examples usually locate on the low-density area of p_d and are generated from labeled data via gradient descent. Instead of using gradient descent, we aim to generate adversarial examples via GAN. Apart from the feature space in semi-supervised learning, DSGAN directly generates samples in the image space. By assigning $p_{\bar{d}}$ as the convolution of p_d and uniform distribution, the samples from p_g will locate on the low-density area of p_d. Furthermore, the distortion ϵ is directly related to the range of uniform distribution. It, however, may be impractical for training the generator for each class. Thus, we propose a novel semi-supervised approach here.

doi:10.6342/NTU201901502

Table 6.6: The architecture of classifiers for semi-supervised learning on SVHN and CIFAR-10. (GN: Gaussian noise, lReLU(leak rate): LeakyReLU(leak rate))

Classifier C for SVHN	Classifier C for CIFAR-10
Input 32×32 RGB image	Input 32×32 RGB image
GN, std = 0.05	GN, std = 0.05
Dropout2d, dropping rate = 0.15	Dropout2d, dropping rate = 0.2
3×3 conv. 64 stride = 1 with WN, lReLU(0.2)	3×3 conv. 96 stride = 1 with WN, lReLU(0.2)
3×3 conv. 64 stride = 1 with WN, lReLU(0.2)	3×3 conv. 96 stride = 1 with WN, lReLU(0.2)
3×3 conv. 64 stride = 2 with WN, lReLU(0.2)	3×3 conv. 96 stride = 2 with WN, lReLU(0.2)
Dropout2d, dropping rate = 0.5	Dropout, dropping rate = 0.5
3×3 conv. 128 stride = 1 with WN, lReLU(0.2)	3×3 conv. 192 stride = 1 with WN, lReLU(0.2)
3×3 conv. 128 stride = 1 with WN, lReLU(0.2)	3×3 conv. 192 stride = 1 with WN, lReLU(0.2)
3×3 conv. 128 stride = 2 with WN, lReLU(0.2)	3×3 conv. 192 stride = 2 with WN, lReLU(0.2)
Dropout2d, dropping rate = 0.5	Dropout, dropping rate = 0.5
3×3 conv. 128 stride = 1 with WN, lReLU(0.2)	3×3 conv. 192 stride = 1 with WN, lReLU(0.2)
1×1 conv. 128 stride = 1 with WN, lReLU(0.2)	1×1 conv. 192 stride = 1 with WN, lReLU(0.2)
1×1 conv. 128 stride = 1 with WN, lReLU(0.2)	1×1 conv. 192 stride = 1 with WN, lReLU(0.2)
Global average Pooling	Global average Pooling
128×10 FC layer with WN	192×10 FC layer with WN

Three stages are required to train our model: First, we train a baseline classifier on all the training data. All the training data are labeled, which represent samples from $\mathcal{L}$ in (6.3). Second, we train a generator to generate unseen data, including adversarial examples, without depending on classifiers and treat these unseen data as additional unlabeled training data ($x \sim p_g$ in (6.3)). Note that the generated data in this stage can be applied to all classifiers. Third, we fine-tune the classifier C_θ with all training data and the generated data produced by the generator via minimizing the following objective:

$$\underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{(x,y)\sim\mathcal{L}}\left[\ell\left(x; y; C_\theta\right)\right] + w \cdot \mathbb{E}_{x_g \sim p_g(x)}\left[\mathcal{H}\left(C_\theta(x_g)\right)\right], \tag{6.3}$$

where the first term is a typical supervised loss such as cross-entropy loss and the second term is the entropy loss $\mathcal{H}$ of generated unlabeled samples corresponding to the classifier, which means that we would like the classifier to confidently classify the generated samples. In other words, if an adversarial example x_g is the closest to one of labeled data x, it should be classified into the class of x. Thus, the additional entropy loss will prevent our model from the attack by adversarial examples.

Furthermore, in (6.3), one can view that w is the trade-off between the importance of labeled data in the high-density area and that of unlabeled data in the low-density area. If w is 0, the model might be prone to classify correctly only on the labeled data. When increasing w, the model will place more emphasis on unlabeled data. Since the unlabeled data act as adversarial examples, the classifier is more robustness.

6.2.1 Experiments Settings

We evaluated the trained models against a range of adversaries, where the distortion is evaluated in terms of ℓ_2-norm or ℓ_{inf}-norm. The adversaries include:

- White-box attacks with Fast Gradient Sign Method (FGSM) [12] using ℓ_{inf}-norm.
- White-box attacks with PGD [47] using ℓ_{inf}-norm.
- White-box attacks with Deepfool [48] using ℓ_2-norm (denotd as DeepFoolL2Attack) and ℓ_{inf}-norm (denoted as DeepFoolLinfinityAttack).

According to different adversaries, we generated 10000 adversarial examples from testing data and calculated the accuracy of the model after attacking. The accuracy is recorded as the probability that adversarial examples fail to attack when the distortion created by attacking algorithm cannot exceed a maximum value. We also train our models with different ranges of uniform distribution. The experimental detail can be found in Appendix 6.2.3.

To validate our method, we proposed two kinds of baseline networks. One is a baseline classifier we train in the first stage, which is a typical classifier trained by all data. The other one is the model with noisy inputs. Adding noise to the input is a prevalent strategy to train a classifier and it is also able to protect the neighborhood of the training data. For fair comparison with our method, uniform noise was used in the second baseline model.

6.2.2 Main Results

Fig. 6.1, 6.2 and 6.3 demonstrates that our models exhibit stronger robustness among all the adversaries.

doi:10.6342/NTU201901502

We claim that our method can outperform other baselines in a wide range of values of w. We find that the model benefits from controlling the weight w. When we increase the w from 1 to 3, and then from 3 to 10, the robustness keeps becoming stronger.

Our second baseline models have the similar intuition with our method as they propagate the label information to the neighborhood of each data point by introducing the noise to inputs. This strategy can improve the accuracy and robustness. Nevertheless, the training data distribution after applying noise can be viewed as a smoother version of the original distribution. Most samples still locate in the high-density area of the original distribution. In view of this, the second baseline models cannot emphasize low-density samples via w as our proposed model does, leading to inferior robustness.

Our method relies on a generator to produce low-density data. The generated samples help our model to put decision boundary outside low-density area. Thus, the model can theoretically resist adversarial attacks with larger distortions. It is worth mentioning that our method is able to combine with the idea of second baseline to the supervised term in (6.3) and the performance might be improved.

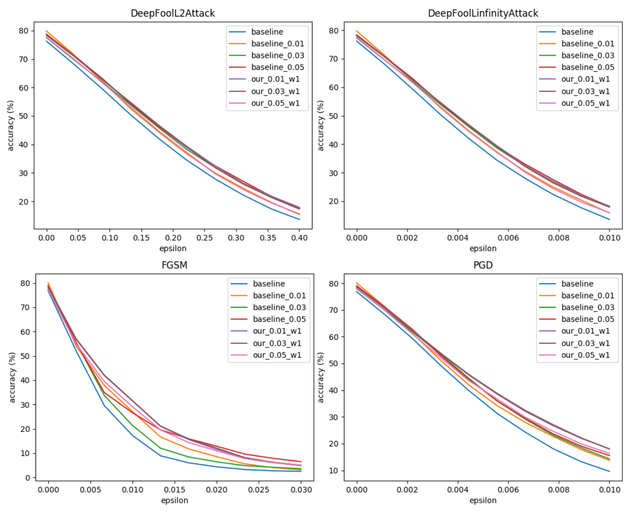


Figure 6.1: Accuracy of baseline and our models after attacks. Blue line indicates the first baseline model. Orange, green and red lines denote the second baseline models with different ranges of uniform noise. Purple, brown and pink lines indicate our methods. In the legend, the float number (0.01, 0.03 and 0.05) also indicates the variance of noises, and "w1" means that w in (6.3) is set to 1. "epsilon" means the ℓ_2 (or ℓ_{inf}) norm between the original image (pixel values are normalized to a range of $[-0.5, 0.5]$) and corresponding adversarial example.

doi:10.6342/NTU201901502

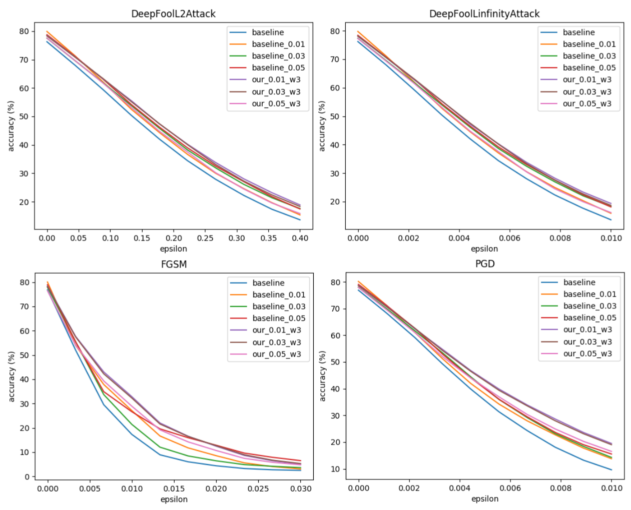


Figure 6.2: The setting is the same with Fig. 6.3 unless $w = 3$.

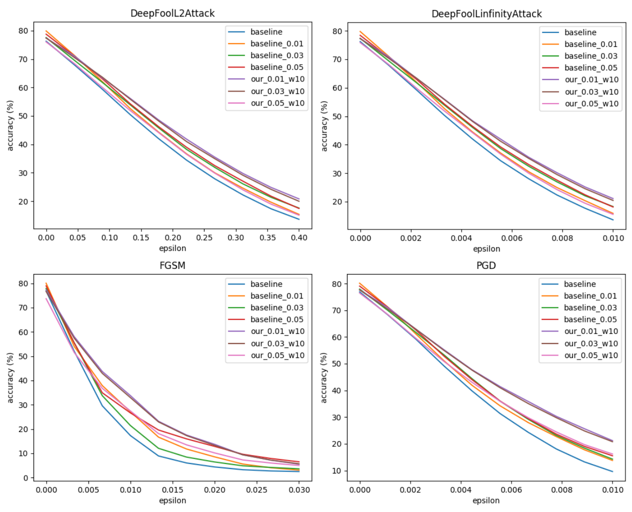


Figure 6.3: The setting is the same with Fig. 6.3 unless $w = 10$.

6.2.3 Appendix: Experimental Details

The size of labeled data for CIFAR-10 is 50000 (45000 for training and 5000 for validation) and we balance the number of data for each class.

In our experiments, as for the second stage, we train DSGAN for 200 epochs in Algorithm 1 to generate our adversarial examples. In the third stage, we finetune the baseline classifier for 50 epochs.

In the experiments on CIFAR-10, the generator and discriminator are the same as those in semi-supervised learning. The architecture is described in Table 6.5 and the classifier is modified from the one shown in Table 6.6. First, we get rid of all the dropouts and Gaussian noise so that we can compare among different models with less randomness.

doi:10.6342/NTU201901502

The architecture is described in Table 6.7. Furthermore, k is assigned to 5 in all experiments.

Table 6.7: The architecture of classifier for robustness enhancement of deep networks on CIFAR-10. (lReLU(leak rate): LeakyReLU(leak rate))

Classifier C for CIFAR-10
Input 32×32 RGB image
3×3 conv. 96 stride = 1 with WN, lReLU(0.2)
3×3 conv. 96 stride = 2 with WN, lReLU(0.2)
3×3 conv. 192 stride = 1 with WN, lReLU(0.2)
3×3 conv. 192 stride = 2 with WN, lReLU(0.2)
3×3 conv. 192 stride = 1 with WN, lReLU(0.2)
1×1 conv. 192 stride = 1 with WN, lReLU(0.2)
Global average Pooling
192×192 FC layer with WN, lReLU(0.2)
192×192 FC layer with WN, lReLU(0.2)
192×192 FC layer with WN, lReLU(0.2)
192×10 FC layer with WN

6.3 DSGAN in Novelty Detection

In this section, we study how to use DSGAN for helping novelty detection. As mentioned in Sec. 5.3, we need to train AE such that (i) positive samples from one seen class have lower reconstruction error; (ii) negative samples from unseen classes incur relatively higher reconstruction error.

The nuclear idea is to use DSGAN to generate negative samples, which originally do not exist under the scenario of novelty detection. Then, we add a new loss term to penalize low reconstruction errors of negative samples (see the third stage below). Three stages are required to train our model (AE):

1. Train the encoder $\text{Enc}(\cdot)$ and decoder $\text{Dec}(\cdot)$ using the loss function (5.5).

2. Given $x \sim p_{pos}$, collect $\text{Enc}(x)$ as samples drawn from p_d. $p_{\bar{d}}$ is the convolution of p_d and a normal distribution with zero mean and variance σ. Then, we train DSGAN to generate negative samples, which are drawn from $p_{\bar{d}}(x) - p_d(x)$ and

doi:10.6342/NTU201901502

are the boundary points around positive samples in the latent space. Note that there are some variations in DSGAN: the input of the generator G is $\text{Enc}(x)$ instead of a random vector z in the latent space; we also add $\| \text{Enc}(x) - G(\text{Enc}(x))\|_2^2$ to train generator so that the output is close to the input.

3. Fixing the encoder, we retrain the decoder by the modified loss function:

$$\min_{\text{Dec}} \mathbb{E}_{x\sim p_{pos}(x)} \left[\|x - \text{Dec}(\text{Enc}(x))\|_2^2 + w \cdot \max\left(0, m - \|x - \text{Dec}(G(\text{Enc}(x)))\|_2^2\right]\right),$$

where w is the trade-off of reconstruction errors between positive samples $\text{Enc}(x)$ and negative samples $G(\text{Enc}(x))$. The second term charges the negative sample with high error bounded by m, which is suggested in [40].

The above algorithm, called VAE+DSGAN, can be used to strengthen the existing AE-based methods by using them in the first stage. In the simulation, we used variational autoencoder (VAE) [7], since it performs better than AE in novelty detection.

6.3.1 Main Results

In this section, following [10], the performance was evaluated using Area Under the Curve (AUC) of Receiver Operating Characteristics (ROC) curve. Given a dataset, one of classes was chosen as the seen class for training and all classes were used for testing. There exist several testing benchmarks for novelty detection, such as MNIST, COIL100 [49] and CIFAR-10. The state-of-the-art method [10] achieves high performance in AUC on MNIST and COIL100 (AUC is larger than 0.97). But, for CIFAR-10, [10] only achieves 0.656. Thus, we chose the challenge dataset CIFAR-10 as the benchmark to evaluate our method. The detailed network architecture can be found in Appendix 6.3.2.

Since VAE+DSGAN can be considered as finetuning VAE [7], we first illustrate the key difference between VAE and VAE+DSGAN in Fig. 6.4. Seen class, which is at the bottom of the images, is car. Other rows are images from unseen classes. One can see that the reconstructed images are fairly good even for the unseen class in VAE. By contrast, our method enforces the reconstructed images of unseen classes to be blurred, while it

doi:10.6342/NTU201901502

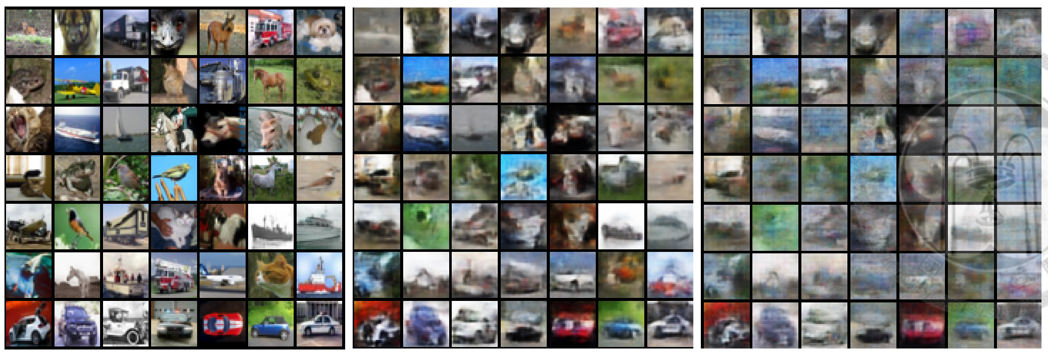


Figure 6.4: Comparison of the reconstructed results of VAE and our method. Seen class, which is at the bottom of the images, is car. Other rows are images from unseen classes. Our method exhibits a relatively larger gap, in terms of reconstruction error between seen data and unseen data, than VAE.

Table 6.8: Comparison between our method (VAE+DSGAN) and state-of-the-art methods: VAE [7], AND [8], DSVDD [9], and OCGAN [10]. The results for Cifar-10 were recorded in terms of AUC value. The number in the top row indicates the seen class, where 0: Plain, 1: Car, 2: Bird, 3: Cat, 4: Deer, 5: Dog, 6: Frog,7: Horse, 8: Ship, 9: Truck.

	0	1	2	3	4	5	6	7	8	9	MEAN
VAE	.700	.386	.679	.535	.748	.523	.687	.493	.696	.386	.583
AND	.735	.580	**.690**	.542	**.761**	.546	.751	.535	.717	.548	.641
DSVDD	.617	**.659**	.508	.591	.609	**.657**	.677	**.673**	.759	**.731**	.648
OCGAN	**..757**	.531	.640	.620	.723	.620	**.723**	.575	**.820**	.554	.657
Our method	.737	.614	.676	**.644**	.759	.562	.660	.646	.769	.633	**.670**

still preserves the reconstruction quality of the seen class. Thus, our method achieves a relatively larger gap, in terms of reconstruction error between the seen data and unseen data, than VAE.

In Table 6.8, we compared the proposed method with several methods, including VAE [7], AND [8], DSVDD [9], and OCGAN [10] in terms of AUC value. One can see that our method almost outperforms VAE in most of cases. Furthermore, the mean of AUC values of our method also is higher than those of state-of-the-art methods. It is worth mentioning that, in addition to VAE, DSGAN has potential of combining with other AE-based methods.

doi:10.6342/NTU201901502

6.3.2 Experimental Details

The architecture of GAN and VAE are depicted in Table 6.9 and 6.10, respectively.

In the experiment, we first trained the VAE for 500 epochs and then we trained DSGAN for 500 epochs with $m = 1.5$ and $w = 0.5$. Third, we fixed the encoder and tuned the decoder with both positive and negative samples (generated by DSGAN) for 600 epochs.

Table 6.9: The architectures of generator and discriminator in DSGAN for novelty detection.

Generator *G*	Discriminator *D*
Input: 128 dimension feature	Input: 128 dimension feature
128×1024 FC layer with BN, ReLU	128×400 FC layer, ReLU
1024×512 FC layer with BN, ReLU	400×200 FC layer, ReLU
512×256 FC layer with BN, ReLU	200×100 FC layer, ReLU
256×128 FC layer	100×1 FC layer

Table 6.10: The architectures of VAE for novelty detection.

Encoder	Decoder
5×5 conv. 32 stride = 2, with BN, lReLU(0.2)	5×5 conv. transpose 128 stride = 2 with BN, lReLU(0.2)
5×5 conv. 64 stride = 2, with BN, lReLU(0.2)	5×5 conv. transpose 64 stride = 2 with BN, lReLU(0.2)
5×5 conv. 128 stride = 2, with BN, lReLU(0.2)	5×5 conv. transpose 32 stride = 2 with BN, lReLU(0.2)
(For mean)	5×5 conv. transpose 3 stride = 2, Tanh
4×4 conv. 128 stride = 1	
(For std)	
4×4 conv. 128 stride = 1	

doi:10.6342/NTU201901502

Chapter 7

Conclusions

We propose DSGAN that can produce any unseen data based on the assumption that the density of unseen data distribution can be the difference between the densities of any two distributions. DSGAN is useful in the environment when samples from unseen data distribution are more difficult to collect than those from the two known distributions. Empirical and theoretical results are provided to validate the effectiveness of DSGAN. Finally, because DSGAN is developed based on GAN, it is easy to extend any improved versions of GAN to DSGAN.

There are three common operations, union, intersection and complement, in set theory. Union and complement can be tackled with traditional GAN and DSGAN, respectively. In the future work, we would like to extend the concepts of DSGAN to cope with the intersection. Once the basic operations are completed, we anticipate that GAN is capable of learning a variety of distributions.

doi:10.6342/NTU201901502

doi:10.6342/NTU201901502

Bibliography

[1] Shuo Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. From facial parts responses to face detection: A deep learning approach. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 3676–3684, 2015.

[2] Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. In *ICLR*, 2016.

[3] Chongxuan Li, Kun Xu, Jun Zhu, and Bo Zhang. Triple generative adversarial nets. In *NIPS*, 2017.

[4] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In *NIPS*, pages 2234–2242. 2016.

[5] Zihang Dai, Zhilin Yang, Fan Yang, William W Cohen, and Ruslan R Salakhutdinov. Good semi-supervised learning that requires a bad gan. In *NIPS*, pages 6510–6520. 2017.

[6] Xiang Wei, Boqing Gong, Zixia Liu, Wei Lu, and Liqiang Wang. Improving the improved training of wasserstein gans: A consistency term and its dual effect. In *ICLR*, 2018.

[7] D. P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*. 2014.

[8] D. Abati, A. Porrello, S. Calderara, and R. Cucchiara. And: Autoregressive novelty detectors. In *IEEE CVPR*, 2019.

doi:10.6342/NTU201901502

[9] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In *ICML*, pages 4393–4402, 2018.

[10] Pramuditha Perera, Ramesh Nallapati, and Bing Xiang. OCGAN: one-class novelty detection using gans with constrained latent representations. In *IEEE CVPR*, 2019.

[11] Y. Yu, W.-Y. Qu, N. Li, and Z. Guo. Open-category classification by adversarial sample generation. In *IJCAI*, pages 3357–3363, 2017.

[12] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[13] Y. Saito, S. Takamichi, and H. Saruwatari. Statistical parametric speech synthesis incorporating generative adversarial networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(1):84–96, 2018.

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680. 2014.

[15] A.V. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, pages 1747–1756. 2016.

[16] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *NeurIPS*, 2018.

[17] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2242–2251, 2017.

[18] Bo Dai, Dahua Lin, Raquel Urtasun, and Sanja Fidler. Towards diverse and natural image descriptions via a conditional gan. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2989–2998, 2017.

doi:10.6342/NTU201901502

[19] Behrooz Mahasseni, Michael Lam, and Sinisa Todorovic. Unsupervised video summarization with adversarial lstm networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2982–2991, 2017.

[20] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NIPS*, 2017.

[21] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2016.

[22] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2813–2821, 2017.

[23] Martín Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *CoRR*, abs/1701.04862, 2017.

[24] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. In *ICLR*. 2017.

[25] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *ICML*, volume 70, pages 214–223, 2017.

[26] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. In *NIPS*, 2017.

[27] M. Hou, B. Chaib-draa, C. Li, and Q. Zhao. Generative adversarial positive-unlabelled learning. In *IJCAI*, pages 2255–2261, 2018.

[28] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training confidence-calibrated classifiers for detecting out-of-distribution samples. In *ICLR*, 2018.

[29] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*. 2018.

doi:10.6342/NTU201901502

[30] Scott E. Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *ICML*, 2016.

[31] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *NIPS*, pages 3581–3589. 2014.

[32] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In *IEEE European Symposium on Security and Privacy (EuroS P)*, pages 372–387, 2016.

[33] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2017.

[34] A. Athalye, N. Carlini, and D. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML*. 2018.

[35] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*. 2018.

[36] Yingzhen Li. Are generative classifiers more robust to adversarial attacks?, 2018.

[37] Bernhard Schölkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Comput.*, 13(7):1443–1471, 2001.

[38] Stanislav Pidhorskyi, Ranya Almohsen, Donald A. Adjeroh, and Gianfranco Doretto. Generative probabilistic novelty detection with adversarial autoencoders. In *NIPS*, pages 6823–6834, 2018.

[39] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *MLSDA*, pages 4–11, 2014.

[40] J. J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. In *ICLR*, 2017.

doi:10.6342/NTU201901502

[41] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley. Least squares generative adversarial networks. In *IEEE ICCV*, pages 2813–2821, 2017.

[42] Y. LeCun, C. Cortes, and C. J. C. Burges. The mnist database of handwritten digits. 1998.

[43] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop*, 2011.

[44] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[45] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. In *ICLR*, 2017.

[46] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *NIPS*, 2015.

[47] A. Kurakin, I. J. Goodfellow, and S. Bengio. Adversarial machine learning at scale. In *ICLR*, 2017.

[48] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *IEEE CVPR*, 2016.

[49] Sameer A. Nene, Shree K. Nayar, and Hiroshi Murase. Columbia object image library (coil-20). 1996.

doi:10.6342/NTU201901502