TRAINING BIDIRECTIONAL GENERATIVE ADVERSARIAL NETWORKS WITH HINTS

by

Uras Mutlu

B.S., Computer Engineering, Istanbul Technical University, 2016

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University
2019

TRAINING BIDIRECTIONAL GENERATIVE ADVERSARIAL NETWORKS WITH HINTS

APPROVED BY:

Prof. Ethem Alpaydın (Thesis Supervisor)	....
Prof. Ali Taylan Cemgil	....
Prof. Olcay Taner Yıldız	....

DATE OF APPROVAL: 02.01.2019

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my thesis advisor Prof. Ethem Alpaydın for his support, guidance and patience through my masters education. I am very grateful that he believed in me and accepted me as his student when I had accomplished nothing. It has been a pleasure for me to work with him and learn from him. I would also like to thank Prof. Ali Taylan Cemgil and Prof. Olcay Taner Yıldız for participating in my thesis jury and their helpful comments and suggestions on my thesis.

I am the most thankful to have a remarkably supportive family and close friends. I could not have started my masters education and begun pursuing an academic career without their emotional support all along the way. I thank my dearest friends Ege Bilgin, Elif Güngör, Eylül Eracar, Kaan Sayım and Sercan Sağman, for being there for me whenever I needed to run away from the overwhelming hours of studying, working or trying to meet deadlines. I thank Pelin Yurdadön wholeheartedly for sharing my burden and offering help wherever she is and whenever she can.

I would like to show my gratitude to the colleagues and friends in Boğaziçi University and Perceptual Intelligence Laboratory for their support and friendship. I thank Alper Ahmetoğlu for having briefly adversarial but invaluably generative discussions and his very helpful insights. I also thank Burak Dündar and Derya Soydaner for their companionship. I am much obliged to Ahmet Alp Kındıroğlu, Alper Kamil Bozkurt, Alptekin Orbay, Doğa Siyli, Gizem Esra Ünlü, Mehmet Burak Kurutmaz, Oğulcan Özdemir and Ufuk Can Biçici for their friendship and the productive discussions in the laboratory room that we share.

This thesis is partially supported by Boğaziçi University Research Funds with Grant Number 18A01P7. We also thank TETAM for the computing facilities provided.

ABSTRACT

TRAINING BIDIRECTIONAL GENERATIVE ADVERSARIAL NETWORKS WITH HINTS

The generative adversarial network (GAN) is a deep learning architecture that learns a generative model by training a later discriminator to best differentiate "fake" examples generated by the generator from the "true" examples sampled from the training set. The generator of GAN takes a low-dimensional latent space vector as input and learns to generate the corresponding input example. The aim of the generator is to generate examples that can not be separated from the true examples by the discriminator. The aim of the discriminator is to maximize the separability of the generated examples from the true examples. A recent extension is the bidirectional GAN (BiGAN) where an encoder is also trained in the inverse direction to generate the latent space vector for a given training example. Recently, Wasserstein GAN has been proposed for GAN and our first contribution is to adapt Wasserstein loss to BiGANs. The added encoder of the BiGAN also allows us to define auxiliary reconstruction losses as hints to learn a better generator, and this is our second contribution. Through experiments on five image data sets, namely, MNIST, UT-Zap50K, GTSRB, Cifar10, and CelebA, we show that Wasserstein BiGANs, augmented with hints, learn better generators in terms of image generation quality and diversity, as measured visually by analyzing the generated samples, and numerically by the 1-nearest-neighbor test.

ÖZET

ÇİFT YÖNLÜ ÇEKİŞMELİ ÜRETİCİ AĞLARIN İPUÇLARIYLA EĞİTİLMESİ

Çekişmeli üretici ağlar (ÇÜA), eğitim kümesindeki "gerçek" örnekler ile üretici ağ tarafından üretilen "sahte" örnekleri birbirinden ayırmak için eğitilen bir ayırıcı ağ yardımıyla üretken bir model öğrenen bir derin öğrenme mimarisidir. ÇÜA'nın üretici ağı düşük boyutlu bir saklı uzay vektörünü girdi olarak alıp bu vektöre karşılık gelen bir örnek üretir. Yakın zamanda öne sürülen çift yönlü ÇÜA'da (ÇYÇÜA) ise ek bir kodlayıcı ağ yardımıyla ters yöne gidilerek girdi olarak verilen bir örnekten saklı uzay vektörü elde edilir. Bu tezdeki ilk katkımız, yine yakın zamanda önerilen Wasserstein ÇÜA'da kullanılan Wasserstein yitiminin ÇYÇÜA'ya uyarlanmasıdır. ÇYÇÜA'ya eklenen kodlayıcı ağ aynı zamanda ipucu niteliğinde yardımcı geri çatma yitimleri tanımlanmasını ve böylece daha iyi eğitilmesini sağlayabilir. Bu tezdeki ikinci katkımız da bu yardımcı geri çatma yitimlerinin tanımlanması ve uygulanmasıdır. Resim içerikli beş farklı veri kümesinde deneyler yapılarak Wasserstein ÇYÇÜA'nın ipuçları eklenmiş halinin resim üretim kalitesi ve çeşitliliği açısından daha iyi üretici ağlar öğrendiği gösterilmiştir. Bu sonuçlara hem üretilen resimlerin görsel analizi, hem de üretilen resimlerle veri kümesinde bulunan gerçek resimler arasında yapılan en-yakın-bir-komşu sınaması sonucu elde edilen nicel verilerin analizi ile varılmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

C	Rectified Linear Function
$\mathbf{C}$	Covariance
D	Discriminator network
D_c	Activations of the c^{th} convolutional layer of the discriminator network
Dec	Decoder network
E	Encoder network
E_m	m^{th} layer of the encoder network
f	First activation function
g	Second activation function
G	Generator network
G_n	n^{th} layer of the generator network
h	Hidden unit
i	i^{th} index
I_c	Activations of the c^{th} layer of the Inception-v3 network
j	j^{th} index
k	k^{th} index
$\mathcal{L}$	Loss
M	Classification model
p	Probability distribution
S	Sample
v	Weight of the connection between hidden and output neurons
w	Weight of the connection between input and hidden neurons
x	Input
$\hat{x}$	Generated data
x^t	Training data
$\mathcal{X}$	Training data distribution
y	Output
z	Latent space

z^t	Sampled latent space for training
$\hat{z}$	Inferred latent space
$\triangle$	Distance Metric
β	Penalty coefficient
μ	Mean
∇	Gradient

LIST OF ACRONYMS/ABBREVIATIONS

△-GAN	Triangle Generative Adversarial Network
1-NN	1-Nearest Neighbor
2D	Two Dimensional
3D	Three Dimensional
AE	Autoencoder
AAE	Adversarial Autoencoder
ALI	Adversarially Learned Inference
ALICE	Adversarially Learned Inference with Conditional Entropy
ANN	Artificial Neural Network
AVB	Adversarial Variational Bayes
BEGAN	Boundary Equilibrium Generative Adversarial Network
BiGAN	Bidirectional Generative Adversarial Network
BiCoGAN	Bidirectional Conditional Generative Adversarial Network
CGAN	Conditional Generative Adversarial Network
CNN	Convolutional Neural Network
ConvD	Convolutional space of the discriminator network
ConvGE	Convolutional spaces of the generator and the encoder network
DCGAN	Deep Convolutional Generative Adversarial Network
DGP	Direct Gradient Penalty
DS	Data Space
EBGAN	Energy-Based Generative Adversarial Network
FBGAN	Featurized Bidirectional Generative Adversarial Network
FA	Factor Analysis
FID	Fréchet Inception Distance
GAN	Generative Adversarial Network
GP	Gradient Penalty
GPU	Graphics Processing Unit
HQBiGAN	High Quality Bidirectional Generative Adversarial Network

Incpt	Feature space of Inception-v3 network
Info-GAN	Information Maximizing Generative Adversarial Network
IS	Inception Score
JGP	Joint Gradient Penalty
KL	Kullback-Leibler
LOO	Leave-One-Out
LS-BiGAN	Loss-Sensitive Bidirectional Generative Adversarial Network
LS-GAN	Loss-Sensitive Generative Adversarial Network
LSGAN	Least Squares Generative Adversarial Network
MDGAN	Multi-Discriminator Generative Adversarial Network
ML	Machine Learning
PCA	Principle Component Analysis
SGD	Stochastic Gradient Descent
VAE	Variational Autoencoder
WGAN	Wasserstein Generative Adversarial Network
WBiGAN	Wasserstein Bidirectional Generative Adversarial Network

1. INTRODUCTION

1.1. Supervised Learning

Machine learning (ML) is a field of computer science studying algorithms that learn from data, hugely nourished from the fields of statistics, probability theory, and optimization. There are some problems for which we do not have an algorithm and ML can help us with those. For example, it is not feasible to explicitly write a function to detect a cat in an image, but suppose we have a number of images containing many objects in them, including cats. We can use these images and the information we know about these images (if they contain a cat or not) to come up with a statistical model. Assuming that we have a rich and diverse collection of images, then we can build an adequate model that estimates the probability of a cat given an image as input. That is, using a method called *supervised learning*, we can come up with an algorithm that *learns* a function from the images and whether they contain a cat or not. More formally within the context of the given example, images are the *data* and information about whether an image contains a cat or not are their corresponding *labels (ground truth).*

One of the reasons why machine learning is very popular these days is the amount of data available. Clearly, a research field interested in learning from data benefits greatly from the so-called explosion of available data. Also improvements in hardware technology and the reduction in costs of computation power make more algorithms feasible. In recent years, the machine learning approach that has been the most popular is the artifical neural network (ANN) model trained with a method called back-propagation [2].

1.2. Neural Networks and Deep Learning

An artifical neural network is inspired by the human nervous system. There are input neurons and output neurons connected to each other. These connections have weight values that determine how much the value of the output neuron is influenced by the values of the input neurons. Basically, we can think of this network as a function of the input values given the weights of the connections as the function parameters.

When we have more layers of neurons between the input and output neurons, with each layer having a connection to the previous layer, we have a multilayer network and when we have many layers, we have a deep learning model. These intermediate layers between the input and the output are called hidden layers and the neurons in a hidden layer are named hidden units. Let us consider an example with only one hidden layer. Formally, let x be the input and y be the output. We can calculate the values (activations) of the hidden units as:

$$h_i = f\left(\sum_j w_{ij} x_j\right) \tag{1.1}$$

where w_{ij} is the weight of the connection between input x_j to the hidden unit h_i and f is the activation function. After calculating the values of the hidden units, output of the network can be calculated as:

$$y_i = g\left(\sum_k v_{ik} h_k\right) \tag{1.2}$$

where v_{ik} is now the weight from hidden unit h_k to the output unit y_i and g is the activation function. Figure 1.1 shows an example neural network.

We need a measure of error to be able to train (improve) our model. The higher is the value of the error function, the more the weights of the network must be tweaked so that the model can give the desired output and has smaller error. Training a deep learning model means minimizing the error of the network with respect to the input data. This is achieved with an iterative optimization algorithm called the stochastic gradient descent (SGD). Using SGD to minimize the error of a neural network is called the back-propagation algorithm [2].

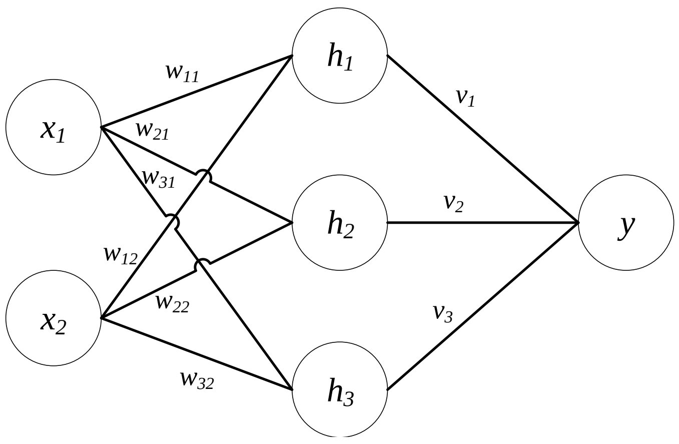


Figure 1.1. An example neural network with one hidden layer. Input layer contains two neurons, hidden layer contains three neurons and the output layer contains one neuron. w and v values are the weights of the connections that are learned from data.

Deep learning has become hugely popular after the AlexNet [1] architecture achieved great success in object recognition from images trained on a large scale image data set. AlexNet is a convolutional neural network (CNN), which is a special type of network mainly applied to the image domain. CNNs take advantage of the 2D structure of an image by scanning the local regions and checking for local features. Pixels in these local regions are multiplied with the weights of the convolutional layer which are meaningful features of the input image. These weights in local regions are called filters, also known as kernels. Each filter has a fixed size named the kernel size that defines the receptive field. The number of filters and the sizes of the receptive field are hyperparameters that determine the number of weights of a convolutional layer. Each filter produces a different set of 2D output neuron activations with their respective weights and when these output activations are stacked together, they form a 3D volume as the convolutional layer output. Figure 1.2 shows the AlexNet architecture with its convolutional layers represented as 3D volumes.

After the success of AlexNet, other significant deep learning architectures followed such as VGG-16 [3], Google Inception [4], and ResNet [5]. These models have generalized very well to the image domain and their pre-trained versions are often used in many computer vision studies.

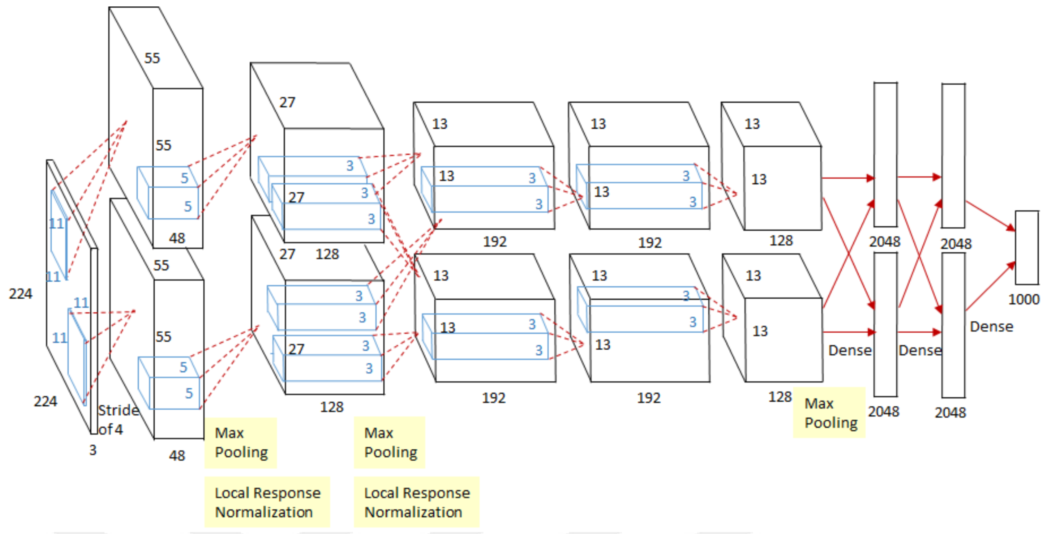


Figure 1.2. AlexNet architecture [1]. The input is a 224 × 224 × 3 image and the output are the class probabilities for classes in the ImageNet data set.

Deep learning models have also been applied to other areas such as natural language processing and speech processing. In the field of natural language processing, there are applications to machine translation [6, 7], building a language model [8], question-answering systems [9], and learning vector representations of words [10]. Speech processing systems are also greatly improved with deep learning models [11,12].

With the great success of deep learning in many problems, studies that experimentally show the weaknesses and security issues of deep learning models have also become popular. Studies on adversarial examples have shown that applying slight perturbations to the input images may cause misclassifications, sometimes with very high confidence [13, 14].

This thesis is one method for generative modeling, which is an application area of machine learning that aims to learn the distribution of the data and generate new examples from it. Learning the distribution of the data in an unsupervised manner means that the generative model learns the structure of the data. The learned model then can be used to obtain abstract representations as rich features that can be used in many applications. With improvements in generative modeling, it is possible to understand the properties and features of any data set without any supervision, which is crucial since there is a lot of data to extract meaning from in the real world.

1.3. Organization

This thesis is organized as follows: In Chapter 2, we discuss the generative adversarial network (GAN) model with the explanation and formalization of the algorithm, literature review, and evaluation methods. In Chapter 3, we explain the bidirectional GAN (BiGAN) with its formulation, adaptation to some other GAN optimization methods, and our proposed hints to improve their training. We give experimental results in Chapter 4 and finally in Chapter 5, we summarize the results and contributions of our work together with some possible future research directions.

2. GENERATIVE ADVERSARIAL NETWORKS

2.1. Algorithm

The Generative Adversarial Network (GAN), proposed in 2014 [15], has since become very popular in the field of generative modeling. GAN is composed of two networks, a generator G and a discriminator D (Figure 2.1). Both G and D are deep neural networks with convolutional and dense layers as appropriate. The generator takes z as input and generates x; z are low-dimensional and are sampled from an assumed probability distribution $p(z)$ (e.g., multivariate Gaussian with independent features). Once training is done, we can generate new x by sampling from $p(z)$ and using G.

The samples generated by G are called *fake*, they are the adverse examples to the *true* x^t that we have in our training set. The aim of the discriminator is to tell the true and fake samples apart as well as possible, and that is how it is trained; the aim of the generator on the other hand is to generate fake samples so well that the discriminator cannot tell them apart. The two networks G and D play an adversarial game and gradually improve their abilities. The following log-likelihood criterion is maximized by D and minimized by G:

$$\mathcal{L}_{GAN} = \sum_{x^t \in \mathcal{X}} \log D(x^t) + \sum_{z^t \sim p(z)} \log(1 - D(G(z^t))) \tag{2.1}$$

where x^t are the true samples drawn from the training set $\mathcal{X}$ and $G(z^t)$ are the fake samples with z^t sampled from $p(z)$. The algorithm of GAN is given in Algorithm 2.1

If we define p_g as the distribution of the samples obtained from the generator, and p_{data} as the distribution of the true samples, this adversarial game is equivalent to minimizing the KL-divergence between p_g and p_{data}.

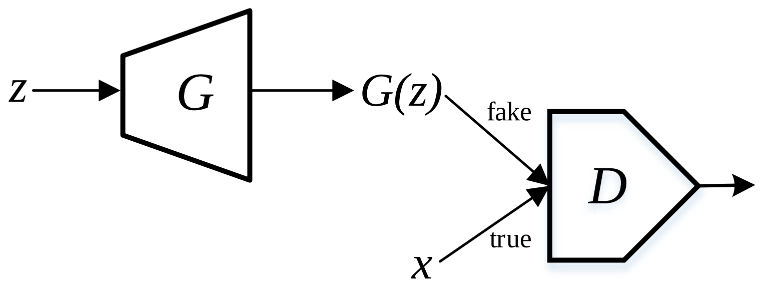
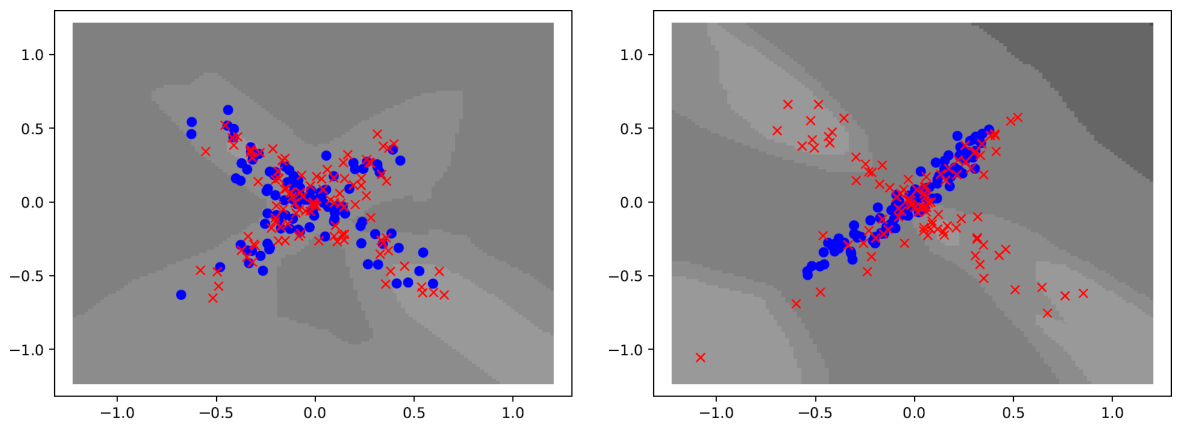


Figure 2.1. The generative adversarial network (GAN) is composed of a generator G and a discriminator D. G generates fake x from z and D learns to discriminate fake $G(z)$ from true x.

```
for number of training iterations do
  for $k$ steps do
    Sample $z^t \sim p(z)$
    Choose $x^t \in \mathcal{X}$
    Update $D(x)$ and $D(G(z))$ by maximizing Eq. 2.1, do not update $G$
  end for
  Sample $z^t \sim p(z)$
  Update $G(z)$ by minimizing Eq. 2.1, do not update $D$
end for
```

Figure 2.2. GAN Algorithm.



(a) Convergence (b) Mode collapse

Figure 2.3. Example data generated by a simple GAN. Cross shaped data points are the true data and the round shaped data points are the generated (fake) data.

The generator network is analogous to a counterfeiter who is trying to produce fakes without getting caught, and the discriminator network is analogous to the police, who is trying to detect if the instance is fake or legitimate [15].

2.2. Training GAN

There is a very rich literature on GANs, even though the architecture is relatively new. This literature review is implicitly split into two sections. We review various research directions and variants of GANs. We will then discuss some notable applications of GANs (chosen from a vast set).

Research has shown that training GANs is difficult [16]. Perhaps the most common problem in GAN training is known as *mode collapse*. This occurs when the generator network of the GAN learns to generate only one or more modes of the data, instead of all the modes. The discriminator is easily fooled since the mode that the generator learns to produce exists in the true data. In Figure 2.3, we show a simple experiment where the data is drawn from a mixture of two bivariate Gaussians. Figure 2.3a shows a good convergence where the generated data covers both components. On the other hand, in Figure 2.3b, the generated data covers only one component. There are a significant number of studies that aim to prevent or reduce the effect of mode collapse [17–20]. Another problem with GAN training is *vanishing gradients.* This occurs when the discriminator output is very close to zero. Both of these problems have been known since the original GAN article [15].

There have been many attempts to stabilize GAN training, both empirically and theoretically. Several empirical tips and tricks have been proposed, such as label smoothing, mini-batch discrimination, and feature matching [18]. Another empirical suggestion is to use denoising feature matching that stabilizes training by obtaining high-level representations from an auto-encoder and feeds these representations to the generator [19].

GANs are also used as class-conditional generators. Conditional GAN (CGAN) [21] provides the means to generate class-conditional samples by feeding the class label to both the generator and the discriminator. A more recent related work is Info-GAN [22], which is trained to code semantic information to the latent space by maximizing the mutual-information between a subset of the latent space and the fake (generated) data.

GAN has found a lot of applications in computer vision. Architectural variants of GAN are used for image-to-image translation [23, 24], enhancing the resolution of images (super-resolution) [25], image inpainting [26], high-resolution image blending [27], semantic image segmentation [28], and text-to-image synthesis [29, 30]. Some additional notable GAN variants include the attention-based GAN [31], and the larger-scale GAN models such as progressive GAN [32], large-scale GAN [33] and the very recent GAN with the style-based generator architecture [34].

There are GAN variants that use different loss functions to stabilize the training. Wasserstein GAN [17] and loss-sensitive GAN (LS-GAN) [20] are the ones that we use in this thesis and explained in Sections 2.3 and 2.4. Least squares GAN (LSGAN) [35] adopts the least squares loss function instead of the log-likelihood for the discriminator to stabilize the training and improve the image generation quality. Mode regularized

GAN [36] penalizes the missing modes during training to ensure that the GAN will not collapse into fewer modes than it should. Energy-based GAN (EBGAN) [37] proposes a set of energy functions and regularizers to stabilize GAN training. An interesting idea in this work is to use an autoencoder architecture for the discriminator and to utilize the reconstruction loss as the energy function. Unrolled GAN [38] uses unrolled optimization for the generator which is a computationally expensive yet successful method to stabilize the training of GANs.

2.3. Wasserstein GAN

From a theoretical perspective, altering the loss criterion of GAN to prevent mode collapse and stabilize training has been popular. A notable example is the Wasserstein GAN [17]. Wasserstein loss is nowadays used as the most popular alternative to the original GAN loss. Informally, Wasserstein distance (also referred to as the Earth-Mover distance) is the minimum cost of transporting the probability mass from one distribution to another; in our case, from that of the fake samples to that of the true samples. It has been shown that minimizing Wasserstein distance leads to more stable training [17] and better quality images when used in generating face images [32]. For the GAN, the Wasserstein loss is defined as:

$$\mathcal{L}_{WGAN} = \sum_{x^t, z^t \in p(z)} D(x^t) - D(G(z^t)) \tag{2.2}$$

for pairs of true and fake samples $(x^t, G(z^t))$. This loss is maximized by D and minimized by G; D tries to make this difference as large as possible and G tries to minimize it. D wants the output of $D(x^t)$ to be higher for true samples x^t than for generated fake samples $D(G(z^t))$ and G wants the opposite. Note that here D is not a 0/1 classifier (estimating the posterior probability that its input is a true sample) as in GAN proper but a regressor whose output a scalar score (of the "trueness" of its input)—D of Wasserstein GAN is called a *critic*. In terms of implementation, the (single) output of D in Wasserstein GAN is a linear unit whereas that of the vanilla GAN has a sigmoid.

An improved version of the Wasserstein GAN proposes an additional *gradient penalty* term to enforce a 1-Lipschitz constraint on the gradients of D [39]:

$$\mathcal{L}_{GP} = (\|\nabla_{G(z)} D(G(z))\|_2 - 1)^2 \tag{2.3}$$

and the augmented Wasserstein loss becomes:

$$\mathcal{L}_{WGAN\text{-}GP} = \mathcal{L}_{WGAN} + \beta \mathcal{L}_{GP} \tag{2.4}$$

where β is the hyper-parameter named the penalty coefficient that defines the trade-off between the two terms.

2.4. Loss-Sensitive GAN

A loss criterion for training GANs that is very similar to the Wasserstein loss is the *loss-sensitive GAN* (LS-GAN) proposed independently [20], which not only looks at the difference as in Wasserstein loss but also imposes a margin. LS-GAN also imposes a prior on the true data distribution by using a Lipschitz regularity condition. This means that in LS-GAN it is assumed that the density of the true data will not change sharply.

In LS-GAN, D is trained to minimize (and G is trained to maximize) the following (In LS-GAN, $D(x)$ learns the “fakeness” as opposed to Wasserstein GAN, where $D(x)$ learns the “trueness” of a sample):

$$\mathcal{L}_{LS\text{-}GAN} = \sum_{x^t, z^t \in p(z)} C(\triangle(x^t, G(z^t)) + D(x^t) - D(G(z^t))) \tag{2.5}$$

where $\triangle$ is a distance metric between x^t and $G(z^t)$ which defines the margin, and C is the rectified linear function to make sure that this equation satisfies $a_+ = max(a, 0)$. An important note here is that if the margin term is removed, LS-GAN becomes identical (except for sign) to minimizing the Wasserstein distance.

LS-GAN also employs a gradient penalty but it slightly differs from Wasserstein GAN. The *direct gradient penalty* is given as:

$$\mathcal{L}_{DGP} = \frac{1}{2}||\nabla_x D(x)||^2 \tag{2.6}$$

When training the discriminator to learn a loss function, the direct gradient penalty is added to the loss of D with the hyper-parameter β:

$$\mathcal{L}_{LS\text{-}GAN_{DGP}} = \mathcal{L}_{LS\text{-}GAN} + \beta\mathcal{L}_{DGP} \tag{2.7}$$

2.5. Evaluation

2.5.1. Inception Score (IS)

Originally, there were not any widely used evaluation criterion for GANs except for visual qualitative analysis. The search for a quantitative and robust evaluation metric for GANs have quickly become an area of interest. Inception Score (IS) is the first such criterion to have become widely adopted in the GAN literature [18]. Idea is to use the Google Inception network [4] pre-trained on the large-scale ImageNet data set [40] to compute the class label distribution of the fake examples. Formally, Inception Score is computed as:

$$\text{IS}(p_g) = e^{\mathbb{E}_{x \sim p_g}[KL(p_M(y|x)||p_M(y))]} \tag{2.8}$$

where M denotes the Google Inception network, x is a fake example, i.e., $x = G(z)$, $p_M(y|x)$ states the label distribution of x predicted by the classification model M, and $p_M(y)$ is the marginal of $p_M(y|x = G(z))$.

The motivation behind $p_M(y|x)$ is that if G is a good generator, the generated image will contain meaningful objects and it will be classified with one of the labels of the ImageNet data set with high probability. On the other hand, if the generated examples are ambiguous, we will not have one of the labels with high probability and $p_M(y|x)$ will be close to $p_M(y)$, leading to a low Inception Score.

There are two main problems here [41]. The first one is the bias introduced by the heavy reliance on the pre-trained network. In other words, the distributions on both sides of the KL divergence term in Eq. 2.8 are heavily dependent on the classification model M. The second problem is the missing distribution of the true samples p_{data} in the equation. One can of course calculate the Inception Score of the true samples and a comparison can be made.

2.5.2. Frechét Inception Distance (FID)

Frechét Inception Distance (FID) [42] is another widely used evaluation metric in the GAN literature. FID also uses the Google Inception network but in a different way. The intermediate representations of true and fake samples are obtained from the Google Inception network. The means and covariances of true and fake sample distributions are calculated under the multivariate Gaussian assumption. The Frechét distance is the distance between the two multivariate Gaussians calculated for the true ($data$) and fake (g) examples:

$$\mathrm{FID}(p_{data}, p_g) = ||\mu_{data} - \mu_g|| + \mathrm{Tr}(\mathbf{C}_{data} + \mathbf{C}_g - 2(\mathbf{C}_{data}\mathbf{C}_g)^{1/2}) \tag{2.9}$$

where μ and $\mathbf{C}$ denote the mean and covariance, respectively.

An FID score close to zero means that the two distributions are close to each other. Unlike the Inception Score, this metric includes the true data distribution in the calculation. The drawback is the bias introduced by the multivariate Gaussian assumption. This metric is also reliant on the pre-trained network.

2.5.3. 1-Nearest Neighbor Test (1-NN)

1-nearest neighbor (1-NN) test [41,43] is a two-sample classifier test to determine if two distributions are identical. With GAN, we have two distributions p_{data} and p_g and we want to assess how much they overlap.

The first step is to take equal number of samples S_{data} and S_g from both distributions and label them as positive and negative, respectively. Then a 1-NN classifier is trained with these samples and the leave-one-out (LOO) accuracy is computed for all of the samples. The idea is that if the two distributions actually overlap (desired case), then the LOO accuracy should be close to 50%, meaning that the true samples and fake samples are mixed well enough to have the accuracy of the 1-NN classifier close to random.

More interestingly, the true and fake accuracies can be analyzed separately, then the 1-NN test gives us more information. For example, when the accuracy for true samples (1-NN true accuracy) is low (close to 0%), it means that the generator is producing fake samples that mix with at least some modes of the true sample distribution. On the other hand, when the accuracy for fake samples (1-NN fake accuracy) is high, this shows that the neighbors of the fake samples are other fake samples. If we see both, this is an indication of mode collapse.

Although 1-NN test is not widely used in the GAN literature, we believe that this metric is both informative and unbiased since 1-NN is a non-parametric classifier that makes no assumptions about the data distributions nor needs any pre-trained classifier. In this thesis, the 1-NN test is our primary evaluation metric we use to compare different GAN models.

There are some other evaluation criteria proposed for GANs but we will not go into more detail in this thesis, but we can state that evaluating GAN performance quantitatively is still an open research problem.

3. BIDIRECTONAL GENERATIVE ADVERSARIAL NETWORKS AND VARIANTS

3.1. Background

In many real-world applications with images, speech, text, and so on, our observations x are high-dimensional; at the same time, we know that all these dimensions are not all necessary or independent. An important research area in machine learning is hence dimensionality reduction where we want to map x to a lower-dimensional z-space without loss of information, and many methods, e.g., principal components analysis (PCA), have been proposed to learn such a mapping. In a *generative model*, we posit that the dimensions of z are *latent factors* that interact to generate the observed x. One example model is factor analysis (FA), which one can view as going in the opposite direction of PCA.

Unsupervised dimensionality reduction can be learned using the neural network architecture called the autoencoder (AE) [44] (Figure 3.1). The encoder part compresses x to z (as in PCA) and the decoder part generates x from z (as in FA). The two networks back-to-back are trained to reconstruct the input, that is, to minimize the difference between the output of the decoder and the input to encoder. In the simplest case, both the encoder and decoder are one-layer (i.e., linear) networks and in this case, it has been shown that the encoder spans the same subspace as PCA, but with the encoder and the decoder having more layers, the AE does *nonlinear* dimensionality reduction with z corresponding to more interesting, abstract features of the input.

Typically the encoder and the decoder are taken to be inverses of each other in terms of network architecture. For example with image data, the encoder starts with one or more convolution layers that successively down-sample followed by one or more dense layers decreasing dimensionality at each layer; the decoder starts from there and increases dimensionality at each layer starting with one or more dense layers and ending with one or more up-sampling convolutions to generate the image back again.

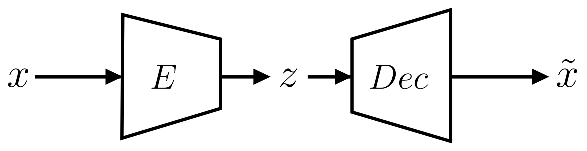


Figure 3.1. The autoencoder (AE) is composed of an encoder E and a decoder Dec. The encoder maps x to latent z and the decoder reconstructs x from z.

The autoencoder is not a generative model; for any x^t, we can find the corresponding z^t and then reconstruct x^t, but we have no way of generating new x outside of the training set. In the variational autoencoder (VAE) [45], we consider z^t as random variables sampled from a known distribution $p(z)$ (e.g., Gaussian), and we add an extra term to the reconstruction error to favor this. Once training is done, we can sample from this $p(z)$ and use the decoder to generate new x.

3.2. Bidirectional Generative Adversarial Networks

GAN proper can generate x for any z but does not have an inverse mapper for generating the corresponding z for a given x. The *Bidirectional GAN* (BiGAN) [46] and the equivalent *Adversarially Learned Inference* (ALI) [47] models were proposed independently and contain also an encoder component E mapping true x to z (in the inverse direction of the generator) (Figure 3.2). Unlike GAN where the discriminator sees only x as input, in BiGAN D sees both x and z, i.e., the observation and its latent representation together. For a true sample, x is given (it is taken from the training set) and the corresponding z is generated by the encoder E; for a fake sample, z is given (sampled from $p(z)$) and corresponding x is generated by the generator G.

The encoder E is also implemented as a deep neural network and (as in AE) its architecture is usually taken as the inverse of G. It is trained just like the generator, namely by back-propagating from the loss function defined at the output of the dis-

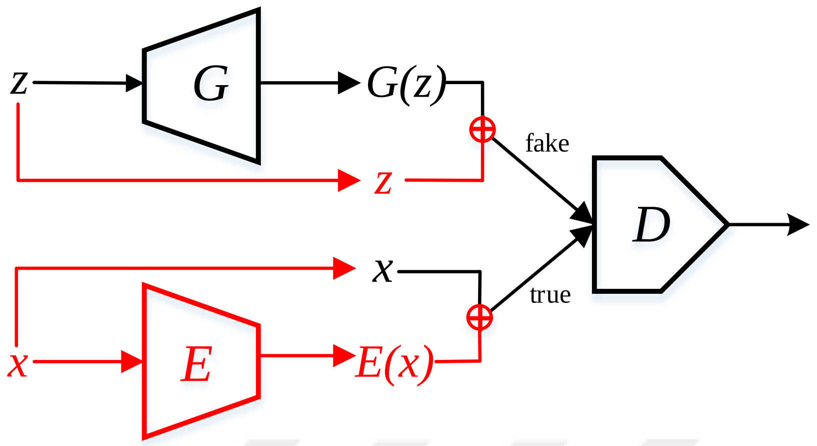


Figure 3.2. The bidirectional GAN also has an encoder E that learns to map true x to latent z. The discriminator takes the concatenated pair (shown as "$\oplus$") z, x as input. The red components show the additions to the GAN proper, shown in Figure 2.1.

```
for number of training iterations do
    for $k$ steps do
        Sample $z^t \sim p(z)$
        Choose $x^t \in \mathcal{X}$
        $\hat{z} = E(x^t)$
        $\hat{x} = G(z^t)$
        Update $D(x, \hat{z})$ and $D(\hat{x}, z)$ by maximizing Eq. 3.1
    end for
    Sample $z^t \sim p(z)$
    $\hat{x} = G(z^t)$
    Update $G(z)$ and $E(\hat{x})$ by minimizing Eq. 3.1
end for
```

Figure 3.3. BiGAN Algorithm.

criminator. Once training is complete, just like we can use the generator to predict x for new z, we can use the encoder to predict z for any x.

In the original BiGAN formulation, the same loss used for GAN is adapted:

$$\mathcal{L}_{BiGAN} = \sum_{x^t} \log D(x^t, E(x^t)) + \sum_{z^t \sim p(z)} \log(1 - D(G(z^t), z^t)) \tag{3.1}$$

except that D sees both the input and its latent representation. Again, this loss is maximized by D and minimized by G and E. We call this the *vanilla BiGAN*. The algorithm of BiGAN is given in Algorithm 3.2.

3.3. Related Work

There are many variations and applications of BiGANs in the literature. Some of these models are existing GAN architectures and variants with different optimization criteria that use a BiGAN-like structure to obtain an inverse mapping from x to z.

There are extensions of the conditional GAN that mix the class-conditional generation property with the ability to learn the inverse mapping, hence introducing the bidirectonality. Two example variants are the invertible conditional GANs [48] and the bidirectional conditional generative adversarial networks (BiCoGAN) [49]. These two models are the class-conditional extensions of BiGANs.

Invertible CGAN [48] uses two encoders, one for obtaining the latent code of an example and the other for obtaining a latent code for a label (attribute). Both outputs are fed into the generator to obtain fake examples with arbitrarily modified attributes. Two more variants are proposed in the same study as using a single encoder with shared layers and two outputs for examples and labels, and using two encoders again but conditioning the image encoder on the label encoder. It is shown that the best results were obtained using the first approach, that is using separate encoders for images and labels without conditioning one to the other.

BiCoGAN [49] is very similar to invertible CGAN. An important difference is that while invertible CGAN uses a pre-trained CGAN to learn the inverse mappings for both the images and labels, encoder of BiCoGAN is trained simultaneously with the generator and the discriminator and it is shown that it learns both the intrinsic and extrinsic factors of the inverse mappings of data samples better than the invertible CGAN.

In triangle GAN ($\triangle$-GAN) [50], two generators and two discriminators are trained to learn the two-way conditional distributions between two domains. Bidirectional mappings between the two domains are learned by the generators and the discriminators are trained to separate the true data and two kinds of fake data, coming from two different domains. These domains can be image-label, image-image, and image-attribute pairs. Thus, $\triangle$-GAN is also trained in a class-conditional manner and it can be used in a variety of applications.

Most of the applications of BiGAN and its variants are in the field of computer vision. Featurized bidirectional GAN (FBGAN) [51] trains a BiGAN and uses the latent representations learned by BiGAN to reconstruct and denoise the images that are used to exploit the vulnerabilities of classification models. These images are called adversarial examples (not related to the adversarial training approach of GANs). Since the BiGAN is assumed to have learned the data distribution in a more abstract way, it becomes more robust against noise and incoherent changes in images. The denoised images by the FBGAN then can be used safely in classification.

BiGAN-like models are also used in image translation [52], conditional image synthesis [53], and automatic colorization [54].

3.4. Wasserstein BiGAN

The motivation to use Wasserstein GAN and its formulation were given in Section 2.3 with the Equations 2.2 and 2.4. It is straightforward to adapt this for BiGAN:

$$\mathcal{L}_{WBiGAN} = \sum_{x^t, z^t \in p(z)} D(x^t, E(x^t)) - D(G(z^t), z^t) \tag{3.2}$$

where again the difference is that D sees both the input and its latent representation.

The gradient penalty term in the improved version of the Wasserstein GAN is also adapted to BiGAN with the extension of the latent space input to D [39]:

$$\mathcal{L}_{JGP} = (\|\nabla_{G(z)} D(G(z), z)\|_2 - 1)^2 \tag{3.3}$$

and the augmented Wasserstein loss for BiGAN becomes:

$$\mathcal{L}_{WBiGAN\text{-}GP} = \mathcal{L}_{WBiGAN} + \beta \mathcal{L}_{JGP} \tag{3.4}$$

where β is the hyper-parameter named the penalty coefficient that defines the trade-off between the two terms.

3.5. Loss-Sensitive BiGAN

We introduce bidirectionality and adapt the loss-sensitive GAN explained in Section 2.4 to BiGAN. If we modify Eq. 2.5 for training BiGANs, we add the latent space vectors to the input of the discriminator and also we define another margin for latent space representations. Margins can be grouped together as:

$$\Delta_{x,z} = \Delta(x, G(z)) + \Delta(z, E(x)) \tag{3.5}$$

Equation 2.5 becomes the following for loss-sensitive BiGAN (LS-BiGAN), again minimized by the discriminator and maximized by the generator:

$$\mathcal{L}_{\textit{LS-BiGAN}} = \sum_{x^t, z^t \in p(z)} C(\Delta_{x^t,z^t} + D(x^t, E(x^t)) - D(G(z^t), z^t)) \tag{3.6}$$

3.6. Hints for Improving Generation Quality

3.6.1. Motivation

Training a GAN is difficult because of a number of reasons.

(i) Though through the concept of adversarial training it is cast as a supervised problem, training a generator is in fact an unsupervised learning task and unsupervised learning is more difficult because there is less feedback.

(ii) There are two models D and G to train and hence the problem of model selection is multiplied by two. Both are typically many-layered deep networks where one needs to fine-tune the depth and width to the task.

(iii) The error at the output of D is used to update not only D but also back-propagated through it to update G, making it doubly deep.

(iv) There is no good measure that we can use to assess that a GAN has converged to a good solution. People typically generate and display a bunch of examples that are evaluated visually but that is far from ideal and cannot be automated. There are some measures that have been proposed (Section 2.5), but they come with assumptions which may not always hold for the tasks at hand.

As a result of these, making a GAN work typically requires much more trial-and-error than with other machine learning scenarios.

The approach that we take in this thesis to improve the training of GAN is by using auxiliary terms that are added to the loss function to be optimized. These terms typically define constraints on G and aim to learn a better G. They can be thought of as regularizers or they can equally be interpreted as “hints” to help the learning process converge to solutions that we believe have a higher chance of giving a better output.

The BiGAN architecture is especially suited in this regard: The fact that the additional encoder E works as the inverse of G, the two placed back to back (E followed by G in Figure 3.2), can be seen as an autoencoder: For a training instance x, we can calculate its reconstruction as $\hat{x} = G(E(x))$. The auxiliary error terms that we use are based on this implicit autoencoder; we could not have defined them on an ordinary GAN.

3.6.2. Related Work

There are GAN variants in the literature that also learn an inverse mapping and use this to improve the training with extra constraints. From the variational Bayes point of view, variational autoencoders combined with GANs present models that resemble BiGAN. The VAE-GAN [55] model combines the GAN architecture with a variational autoencoder and jointly trains GAN with an additional reconstruction loss coming from the VAE. This model does not directly minimize the reconstruction error of pixels but uses an abstract representation obtained from D. This corresponds to learning a similarity metric between true and fake data distributions by utilizing the D network. Alpha-GAN (α-GAN) [56] also combines the best of both worlds, variational training and adversarial training, with tricks such as using a hybrid loss function, adopting a synthetic likelihood instead of the intractable likelihood, and using the D network as a posterior approximator. Adversarial variational Bayes (AVB) [57] also uses a BiGAN-like structure but it differs from BiGAN in using variational training. It is reported that this difference improves the reconstruction quality. A work very similar to the AVB and VAE is the adversarial autoencoder (AAE) [58] which uses a KL divergence loss to match the arbitrary prior distribution of the latent space with the aggregated posterior distribution of the hidden code of the autoencoder. The more recently proposed VEEGAN [59] learns the inverse mapping via a reconstructor network and also includes a reconstruction loss to stabilize the adversarial training. Introspective adversarial networks [60] make use of a hybrid model of VAE and GAN to perform photo editing by combining adversarial training with the variational autoencoder training to minimize the reconstruction loss in the pixel space.

The boundary equilibrium GAN (BEGAN) [61] uses a discriminator network architecture resembling an autoencoder. A reconstruction loss derived from the Wasserstein distance is used as a lower bound for the autoencoder. Multi-Discriminator GAN (MDGAN) [62] is very recent and uses two different discriminators; one is a GAN discriminator and the other one is an autoencoder to enforce a reconstruction error that serves as an anomaly detector. High quality bidirectional GAN (HQBiGAN) [63] is an extension of BiGAN which learns a mapping from the feature space to the latent space (instead of the data space to the latent space in vanilla BiGAN). This means that the input of the E network is not the data points but the abstract features extracted from the D network. CycleGAN [23] uses a BiGAN-like inverse mapping that maps a source image to target image instead of mapping from data space to latent space. With this source-to-target mapping, CycleGAN is able to perform image-to-image translation. This inverse mapping is also used to stabilize the training by introducing a cycle consistency loss. DiscoGAN [64] and DualGAN [65] variants also propose additional reconstruction terms added to the adversarial training of the original GAN. These two studies differ from ours in terms of GAN architectures and using reconstruction loss only on the data (pixel) space.

Perhaps the most similar work in the literature to our work is ALI with conditional entropy (ALICE) [66]. This study also proposes adding a reconstruction loss term to the adversarial training of BiGANs. However, while the main purpose of ALICE is to improve the reconstruction quality of BiGAN and ALI models by adding a cycle-consistency loss term to the training, our aim and proposed methods are more driven to improve the generation quality and diversity of the models, while also improving the reconstruction quality. Therefore, we offer a different set of reconstruction criteria than ALICE to enhance the training of both the generator and the encoder. Another difference is that in our work we experiment with Wasserstein GAN and loss-sensitive GAN, while ALICE uses the vanilla GAN formulation as the adversarial training criterion.

In the following subsections, we discuss four such criteria leading to four different BiGAN variants. We define four auxiliary errors and they are added to the BiGAN loss of Equation (3.4) after multiplied by a λ term, which is adjusted using cross-validation. Note that such auxiliary error terms are hints on G (and E), and as such are used to update G (and E), to learn a better generator (and encoder), and even if they use D, they do not update D. Table 3.1 shows the summary of the proposed hints.

3.6.3. Data Space

The most straightforward method is to use the reconstruction loss in the original data space (e.g., pixels in images) as minimized by the auto-encoder: (see Figure 3.4)

$$\mathcal{L}_{DS} = \sum_{x^t} \|x^t - \hat{x^t}\|^2 = \sum_{x^t} \|x^t - G(E(x^t))\|^2 \tag{3.7}$$

where x is the actual data and $\hat{x}$ is its reconstruction using first E as the encoder and then G as the decoder. This method is similar to DiscoGAN [64], DualGAN [65], and ALICE [66].

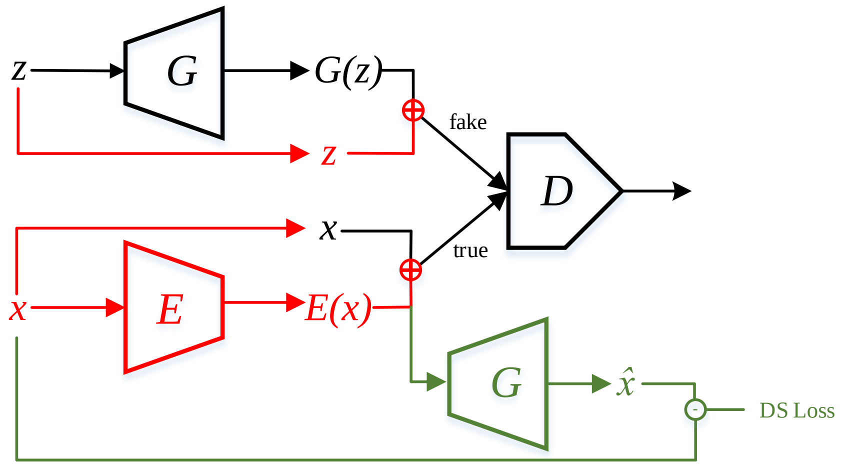


Figure 3.4. Data Space

3.6.4. Convolutional Spaces of G and E

Instead of measuring the loss in the data space, we can measure it at a more abstract level, corresponding to features higher than pixels in images. Because G and E networks are taken as the exact inverses of each other, we can define a correspondence between layers of G and E: Increasing layers of G correspond to decreasing layers of E. (See Figure 3.5)

For training instance x, Let $G_n(E(x))$ and $E_m(x)$ be the vectors of activations of at convolution layer n of G and convolution layer m of E respectively. Then we can define the following a loss in terms of their difference:

$$\mathcal{L}_{ConvGE} = \|G_n(E(x)) - E_m(x)\|^2 \tag{3.8}$$

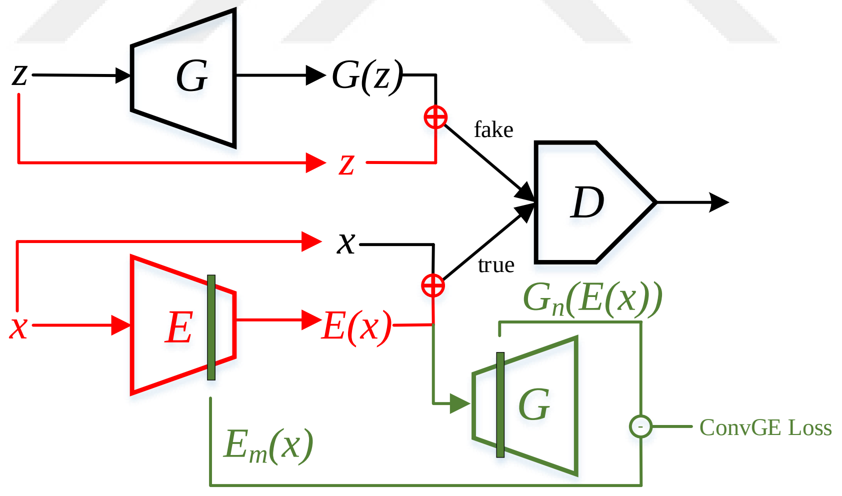


Figure 3.5. Convolutional Spaces of G and E

3.6.5. Convolutional Space of D

The discriminator D takes x as input and processes it in its many layers learning successively more abstract representations. Therefore, we can first pass x and $\hat{x} \equiv G(E(x))$ from convolutional layers of D and then compare the activations at some layer of D. (See Figure 3.6)

We denote by $D_c(x)$ the vector of activations at layer c of D, and define a loss in terms of the difference there between a training instance and its reconstruction:

$$\mathcal{L}_{ConvD} = \|D_c(x) - D_c(\hat{x})\|^2 \tag{3.9}$$

Note that we do not update D with this term, we only update G and E; we are only using all the layers of D until c for feature extraction. A similar approach is also taken in VAE-GAN [55] and α-GAN [56].

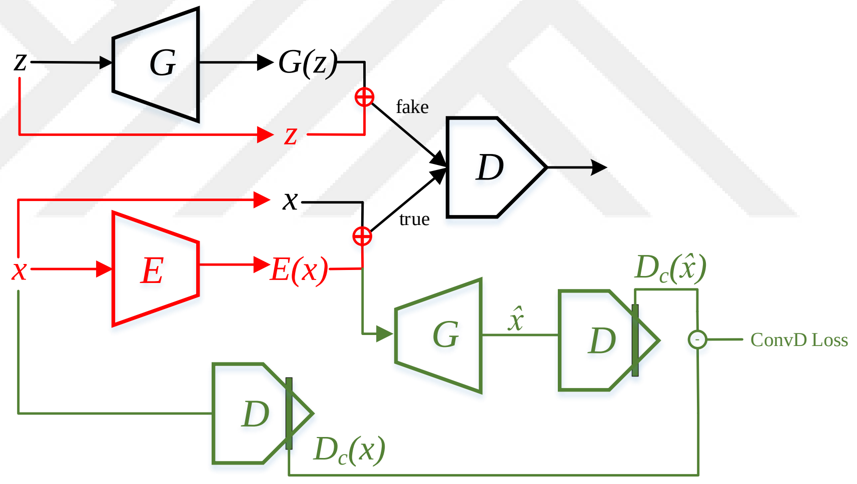


Figure 3.6. Convolutional Space of D

3.6.6. Feature Space of Inception-v3 Network

Inception-v3 network is a very deep convolutional neural network which performs well on many computer vision tasks [67]. This network is trained with the very large ImageNet data set and it is believed to have generalized well to the image domain. Because of this we can hypothesize that it has learned to detect important high-level features in its later layers which define a space in which to compare the quality of a reconstruction, x vs. $\hat{x}$. (See Figure 3.7)

Let $I_c(x)$ be the vector of activations of the Inception-v3 network at layer c. Then the reconstruction loss can be defined as:

$$\mathcal{L}_{Incpt} = \|I_c(x) - I_c(\hat{x})\|^2 \tag{3.10}$$

Again, the inception network is just used as a postprocessor whose output is used to update G and E.

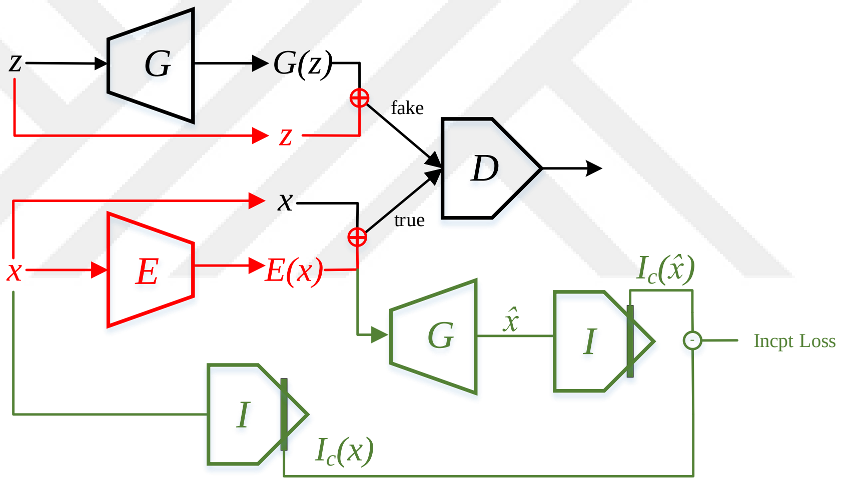


Figure 3.7. Feature Space of Inception-v3 Network

Table 3.1. Summary of Proposed Hints

	Definition	Formula
DS	Data space	$\|x - G(E(x))\|^2$
ConvGE	Convolutional spaces of G and E	$\|G_n(E(x)) - E_m(x)\|^2$
ConvD	Convolutional space of D	$\|D_c(x) - D_c(\hat{x})\|^2$
Incpt	Feature space of Inception-v3 network	$\|I_c(x) - I_c(\hat{x})\|^2$

4. EXPERIMENTS AND RESULTS

4.1. Setting

4.1.1. Data Sets

We use five well-known real-world image data sets frequently used to test GANs; they are MNIST, UT-Zap50K shoes, German Traffic Sign Recognition Benchmark (GTSRB), Cifar10, and CelebA.

The MNIST data set [68] consists of 60,000 handwritten grayscale digit images each of size 28×28. The training set contains 50,000 images and 10,000 images are in the test set. There are a total of 6,000 images for each digit from zero to nine. For architectural convenience, we resize the images in the data set to 32×32 using bilinear interpolation. We also normalize the pixel values to the range between -1 and 1. Reason of this normalization is given in Section 4.1.2. Examples from the MNIST data set are given in Figure 4.1. MNIST is probably the most popular image data set used in deep learning in general and GANs in particular.

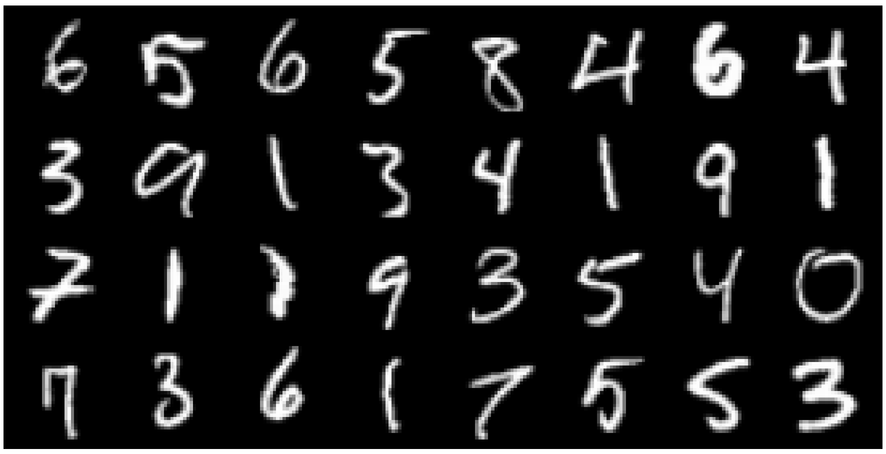


Figure 4.1. Some examples from MNIST.

The UT-Zap50K [69, 70] data set contains 50,025 shoe images in the RGB-color scheme. These are catalog images collected from an e-commerce website with four major categories; shoes, sandals, slippers, and boots. All of the images contain a white background and the shoes are centered. Images in the original data set are of varying sizes but we resize them to 32×32 for comparability with other data sets and methods. Some example images are shown in Figure 4.2. We use this data set to measure the quality and diversity of the fake samples since there are many fine-grained details in shoe images.

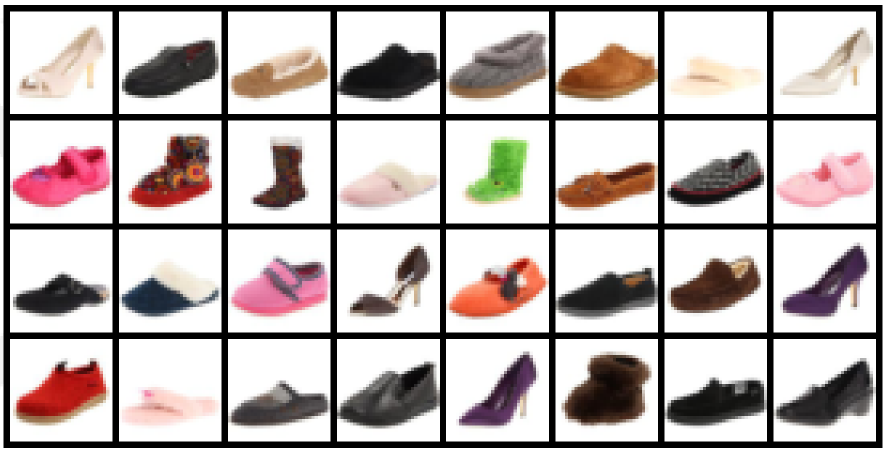


Figure 4.2. Some examples from UT-Zap50K.

The GTSRB [71] data set contains more than 50,000 images of traffic signs taken from the streets and roads in Germany. There are more than 40 classes of traffic signs in the data set. Images are collected in a realistic manner under different lighting, angles, and backgrounds. The training set contains 39,209 traffic sign images, which we again resize to 32×32 for architectural convenience and comparability. Some examples can be seen in Figure 4.3. The motivation to use this data set comes from the fact that it contains realistic images with different angles and backgrounds, therefore it is not easy for GAN to produce fake samples that look real. Nowadays, with research moving forward in autonomous vehicles we believe that working on such a data set is very important.

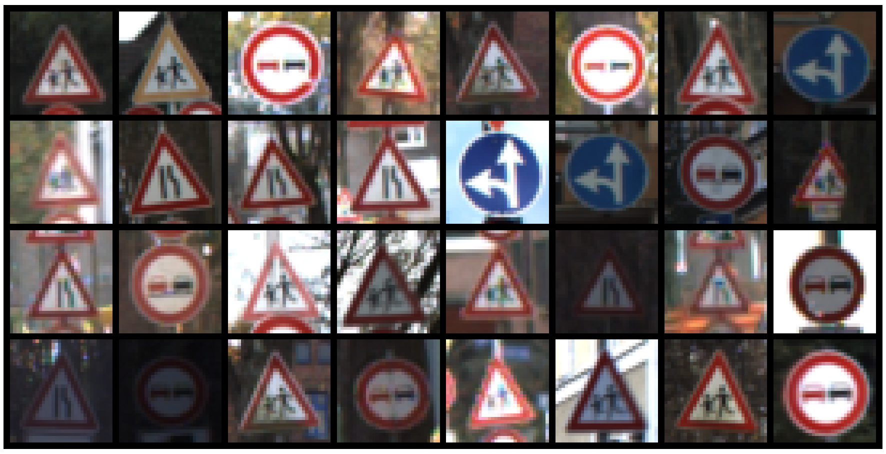


Figure 4.3. Some examples from GTSRB.

The Cifar10 data set [72] contains 50,000 training images and 10,000 test images of ten various objects and scenes. Images are in color and each of them is sized 32×32. The ten classes of images include many natural objects such as birds, cats, dogs, frogs, airplanes, and so on. Each class contains 6,000 samples in total. Generating fake data to resemble the images in this data set is still a challenge for GAN since the images contain diverse backgrounds, lighting, angles, and perspectives. Cifar10 is another very well known benchmark data set in the research field of computer vision. Figure 4.4 shows some examples. We use Cifar10 to mainly assess and demonstrate the reconstruction ability of our methods.

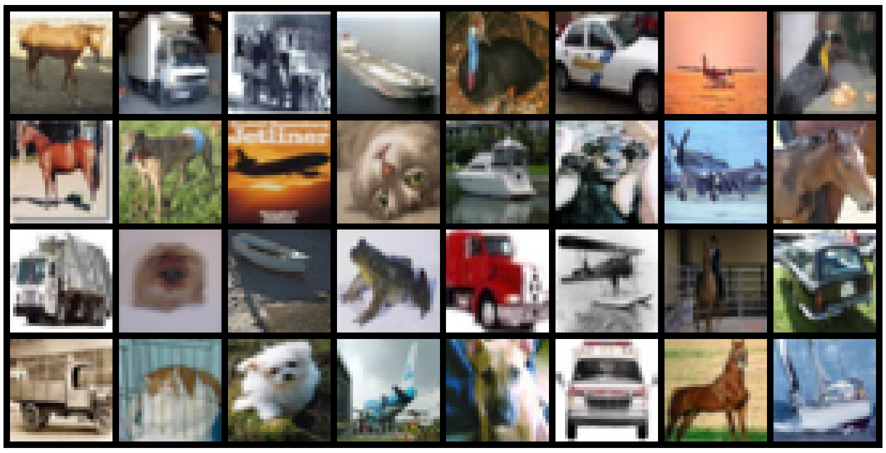


Figure 4.4. Some examples from Cifar10.

CelebA [73] is a large-scale data set of celebrity face images, consisting of 202,599 samples of 10,177 identities collected from the internet. There are also five landmark locations and 40 attribute annotations for each image. Images in the data set contain many pose variations and background clutter. The rich amount of annotations and labels provide the means to use this data set in a diverse set of computer vision applications. Original images are in color and in various sizes; we use the aligned and cropped version of the data set. Images are roughly aligned using similarity transformation according to the eye locations. We resize the images to 64×64. Some examples can be seen in Figure 4.5. CelebA data set has quickly become a widely used data set in the GAN literature and we decided to experiment on this data set for comparison to other studies and to test the semantic attributes of the latent space interpolations.

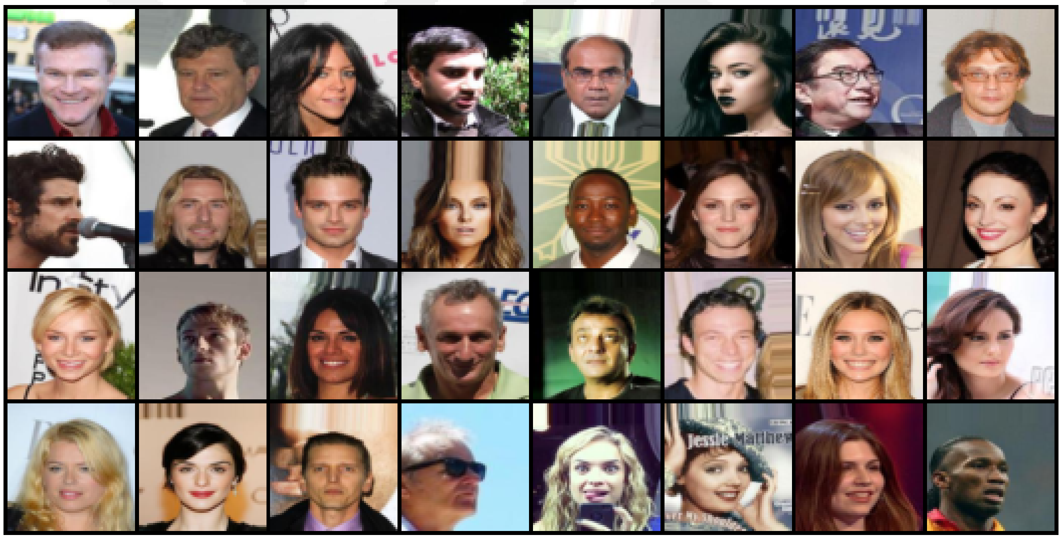


Figure 4.5. Some examples from CelebA.

4.1.2. Model Architectures

For the generator, the discriminator, and the added encoder for BiGAN, we use network architectures that are similar to the ones used in Deep Convolutional GAN (DCGAN) [74]. The architectures of the networks for the generator and the encoder are given in Tables 4.1 and 4.2, respectively. The encoder architecture is the exact inverse of the DCGAN generator. Note that the last layer of the generator performs

Table 4.1. The Generator Architecture of 32×32 Models

Layer	Input	Output	Kernel	Stride	Padding	Output Size
Input	Input size = $z \times 1 \times 1$					$z \times 1 \times 1$
ConvTranspose2D	z	128	4×4	1	0	$128 \times 4 \times 4$
BatchNorm2D	Momentum = 0.9					$128 \times 4 \times 4$
ReLU						$128 \times 4 \times 4$
ConvTranspose2D	128	64	4×4	2	1	$64 \times 8 \times 8$
BatchNorm2D	Momentum = 0.9					$64 \times 8 \times 8$
ReLU						$64 \times 8 \times 8$
ConvTranspose2D	64	32	4×4	2	1	$32 \times 16 \times 16$
BatchNorm2D	Momentum = 0.9					$32 \times 16 \times 16$
ReLU						$32 \times 16 \times 16$
ConvTranspose2D	32	3	4×4	2	1	$3 \times 32 \times 32$
Tanh						$3 \times 32 \times 32$

Table 4.2. The Encoder Architecture of 32×32 Models

Layer	Input	Output	Kernel	Stride	Padding	Output Size
Input	Input size = $3 \times 32 \times 32$					$3 \times 32 \times 32$
Conv2D	3	32	4×4	2	1	$32 \times 16 \times 16$
BatchNorm2D	Momentum = 0.9					$32 \times 16 \times 16$
ReLU						$32 \times 16 \times 16$
Conv2D	32	64	4×4	2	1	$64 \times 8 \times 8$
BatchNorm2D	Momentum = 0.9					$64 \times 8 \times 8$
ReLU						$64 \times 8 \times 8$
Conv2D	64	128	4×4	2	1	$128 \times 4 \times 4$
BatchNorm2D	Momentum = 0.9					$128 \times 4 \times 4$
ReLU						$128 \times 4 \times 4$
Conv2D	128	z	4×4	1	0	$z \times 1 \times 1$

Table 4.3. The Discriminator Architecture of 32×32 Models. *Not included in the discriminator of Wasserstein BiGAN

Layer	Input	Output	Kernel	Stride	Padding	Output Size
Input x^t	Input size = $3 \times 32 \times 32$					$3 \times 32 \times 32$
Conv2D	3	32	4×4	2	1	$32 \times 16 \times 16$
LeakyReLU	Slope = 0.2					$32 \times 16 \times 16$
Dropout2D	Rate = 0.2					$32 \times 16 \times 16$
Conv2D	32	64	4×4	2	1	$64 \times 8 \times 8$
BatchNorm2D*	Momentum = 0.9					$64 \times 8 \times 8$
LeakyReLU	Slope = 0.2					$64 \times 8 \times 8$
Dropout2D*	Rate = 0.2					$64 \times 8 \times 8$
Conv2D	64	128	4×4	2	1	$128 \times 4 \times 4$
BatchNorm2D*	Momentum = 0.9					$128 \times 4 \times 4$
LeakyReLU	Slope = 0.2					$128 \times 4 \times 4$
Dropout2D*	Rate = 0.2					$128 \times 4 \times 4$
Conv2D	128	128	4×4	1	0	$128 \times 1 \times 1$
Input z^t	Input size = $z \times 1 \times 1$, Concat with $\uparrow$					$(128 + z) \times 1 \times 1$
Dropout2D*	Rate = 0.2					$(128 + z) \times 1 \times 1$
Conv2D	$(128 + z)$	256	1×1	1	0	$256 \times 1 \times 1$
LeakyReLU	Slope = 0.2					$256 \times 1 \times 1$
Dropout2D*	Rate = 0.2					$256 \times 1 \times 1$
Conv2D	256	256	1×1	1	0	$256 \times 1 \times 1$
LeakyReLU	Slope = 0.2					$256 \times 1 \times 1$
Dropout2D*	Rate = 0.2					$256 \times 1 \times 1$
Conv2D	256	1	1×1	1	0	$1 \times 1 \times 1$
Sigmoid*						$1 \times 1 \times 1$

hyperbolic tangent (tanh) activation to produce images with pixel values ranging from -1 to 1, which is the reason to normalize all the training images in the data sets between the same range. The only difference for BiGAN is that the discriminator takes both the data point x and its latent z as input. The BiGAN discriminator passes x through some convolutional layers and then z is concatenated to the output of the last of these convolutional layers, which is then passed through two additional 1×1 convolutions (both having 256 hidden units for 32×32 inputs and 512 hidden units for 64×64 inputs) acting as fully connected layers before outputting a scalar number (in the vanilla BiGAN, a sigmoid activation is applied to this scalar), see Table 4.3 for the architecture of the discriminator for 32×32 inputs. Following the original Wasserstein GAN [17], we did not add batch normalization and dropout layers in the discriminator of Wasserstein BiGAN. We follow the same approach for also LS-BiGAN. All data sets except CelebA has 32×32 inputs; for CelebA, we add one additional convolutional layer for mapping between 64×64 and 32×32.

4.1.3. Training Details

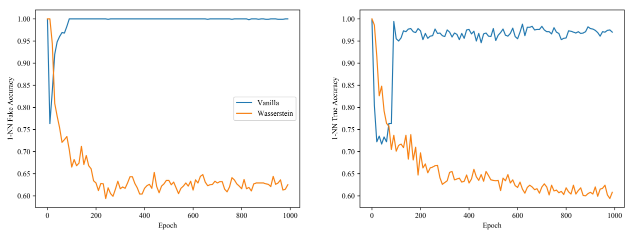
With MNIST, the latent z dimensionality is 50 and models are trained for $1,000$ epochs. For UT-Zap50K, GTSRB, and Cifar10, z dimensionality is 64 and the models are trained for 800 epochs. For CelebA models, z dimensionality is 100 and the number of epochs is 200. In all experiments, the latent z are sampled from standard normal distribution; we used 0.0005 as the learning rate and Adam optimizer. Training is done on a single Tesla V100 GPU.

4.2. Vanilla vs. Wasserstein BiGAN

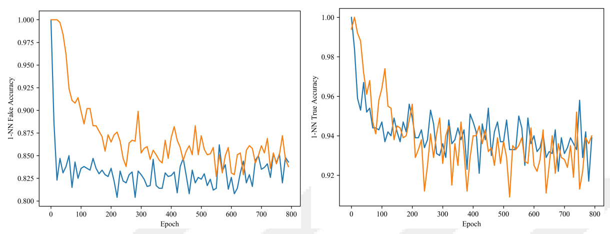
In our first set of experiments, we compare vanilla and Wasserstein BiGAN on all five data sets in terms of generated image quality and diversity. To assess quantitatively, we use the leave-one-out (LOO) 1-nearest neighbor (1-NN) classifier test [41, 43] as follows: After each training epoch, we generate $1,000$ fake samples using the generator and we take $1,000$ true samples from a held-out test set. For each sample from this $2,000$ images, we leave the sample out, fit a 1-NN classifier to the remaining samples with their class labels as true and fake, and we predict the class of the left-out sample. If the true data distribution and the generated fake distribution overlap, as we hope they do, we expect both true and fake accuracies to be around 0.5. Typically, these values start from around 1.0 and decrease during learning. Since 1-NN test is done on a held-out set of data, it also evaluates the diversity of the generated images.

In Figure 4.6, we show these on all five data sets. On MNIST, we see that Wasserstein GAN converges to almost 0.5 whereas this is not the case for vanilla BiGAN—the fake accuracy goes back up to 1.0 after 60 epochs; the fakes form a group by themselves. The true accuracy converges to around 0.8, which shows that most true samples are close to each other. These imply that the generator is able to realistically generate only one mode or some modes of the data, which is an indication of mode collapse or mode dropping. On UT-Zap50K, GTSRB, and Cifar10, Wasserstein BiGAN and Vanilla BiGAN perform very similarly. In the case of CelebA, Wasserstein BiGAN converges to around 0.5 accuracy for both true and fake samples whereas vanilla BiGAN's fake accuracy is around 0.7 and its true accuracy is around 0.4. This shows that Wasserstein BiGAN is able to generate more realistic samples for CelebA. The first two rows for each data set in Table 4.4 summarize the average fake and true accuracies respectively; vanilla BiGAN and Wasserstei BiGAN results are in the first two columns.

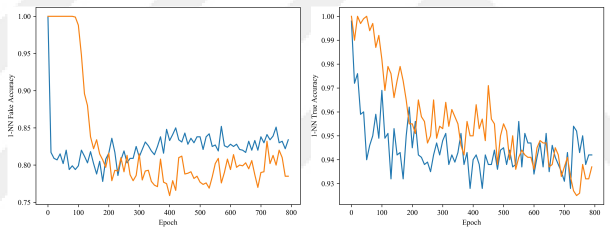
Another criterion used to assess GAN is the Fréchet-Inception distance (FID) [42] that we use here to compare Vanilla and Wasserstein BiGANs. As we discussed in Section 2.5.2, FID first passes true and fake samples through the Inception-v3 network and retrieves the activations of an (768-dimensional) intermediate layer. It then fits multivariate Gaussians to representations of true and fake samples separately and measures the distance between them. Table 4.4 (the third row for each data set) also shows the results of the FID evaluation. On MNIST and Cifar10, WBiGAN-Inception model gives the best FID scores. A reason for Inception model performing best on Cifar10 may be that the Inception-v3 network is trained by ImageNet data set which contains the classes of Cifar10 data set. In the case of UT-Zap50K and GTSRB, best FID scores are obtained by Vanilla BiGAN. On CelebA data set all the models except vanilla BiGAN



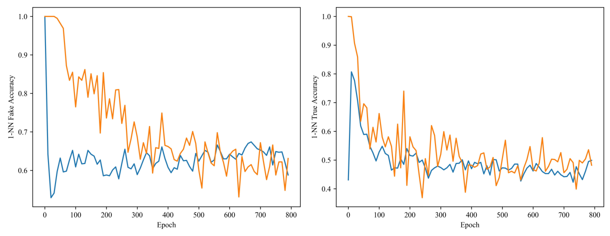
(a) MNIST



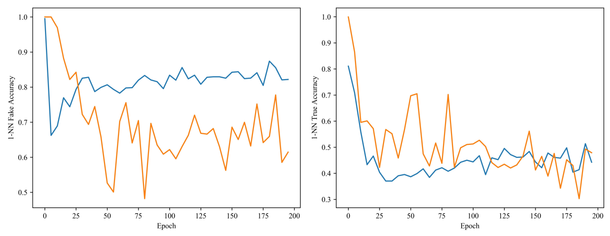
(b) UT-Zap50K



(c) GTSRB



(d) Cifar10



(e) CelebA

Figure 4.6. The evolution of the 1-NN leave-one-out fake (left) and (true) accuracies of vanilla and Wasserstein BiGANs during training.

give competitive results, the best being WBiGAN. Although FID is widely used in the GAN literature, we believe that assuming the multivariate Gaussian is too restrictive.

4.3. Vanilla vs. Loss-Sensitive BiGAN

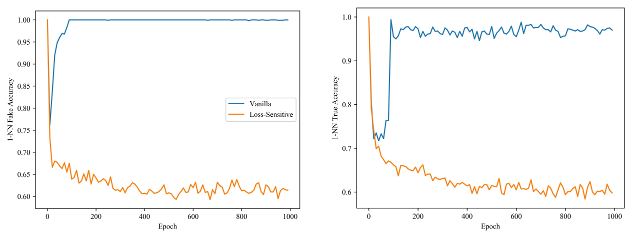
As an extension to our first set of experiments to decide the best performing BiGAN variant, we compare the Vanilla BiGAN and the loss-sensitive BiGAN on all five data sets. We measure the image generation quality and diversity using the 1-NN test. The results of the experiments are shown in Figure 4.7. It can be seen that in terms of fake and true accuracies, LS-BiGAN is closer to 50% on all of the data sets except for the GTSRB. We also see again that mode collapse does not occur on MNIST with LS-BiGAN. Looking at the results, we can infer that using LS-BiGAN instead of vanilla BiGAN is a better choice for all data sets except for GTSRB in terms of image quality and diversity. It is important to note that LS-BiGAN also stabilizes the training of the MNIST model.

Our experimental results using the loss-sensitive approach, neither for GAN nor for BiGAN, do not show any significant difference from those using Wasserstein loss.

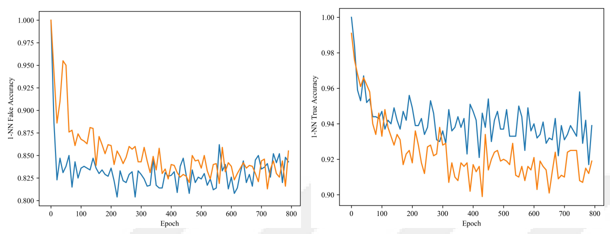
4.4. The Effect of Different Hints on Reconstruction Criteria

In our second set of experiments, we test for the effect of the auxiliary hints we proposed in Section 3.6 with Wasserstein BiGAN, leading to four variants.

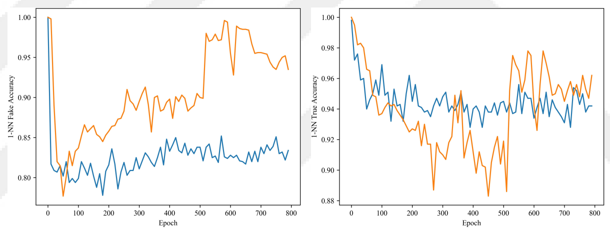
After training a BiGAN variant, we use its generator G as the decoder and put it after its encoder E to form an autoencoder. We can then use this autoencoder to assess the quality of the reconstruction learned by BiGAN as follows: We take an out-of-the-sample test instance x, pass it first the encoder to obtain a latent z, which we then pass through the generator (using it as a decoder) network to get $\tilde{x}$, and we calculate $\|x-\hat{x}\|^2$. We do this on 2,000 test samples and show how the average changes during training for all BiGAN experiments.



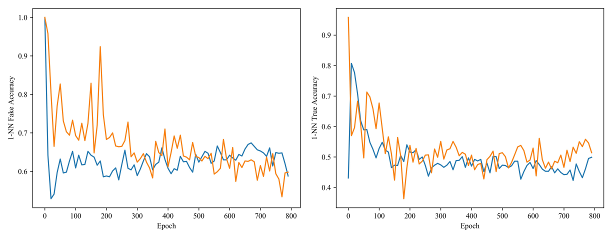
(a) MNIST



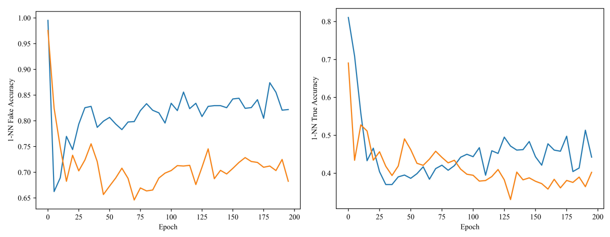
(b) UT-Zap50K



(c) GTSRB



(d) Cifar10



(e) CelebA

Figure 4.7. The evolution of the 1-NN leave-one-out fake (left) and (true) accuracies of vanilla and Loss-Sensitive BiGANs during training.

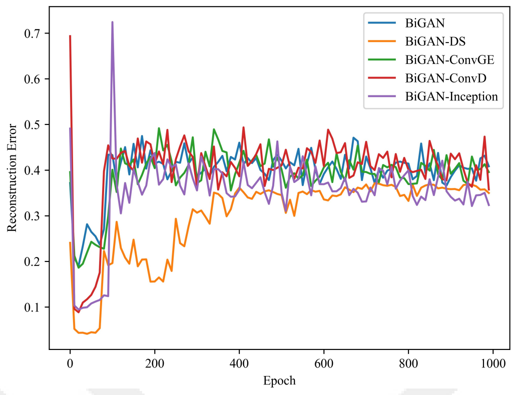
Table 4.4. Fake and true LOO 1-NN accuracies, FID scores, and reconstruction errors of different Vanilla and Wasserstein BiGAN variants. *Best scores.

		BiGAN	WBiGAN	DS	ConvGE	ConvD	Inception
MNIST	Fake	0.763	0.594	0.624	0.629	0.596*	0.610
	True	0.717	0.594	0.556	0.618	0.585*	0.597
	FID	32.03	37.11	8.75	9.66	8.89	8.48*
	Rec	0.190	0.264	0.041*	0.278	0.060	0.212
UT-Zap50K	Fake	0.804	0.827	0.767	0.787	0.750*	0.769
	True	0.917	0.909*	0.912	0.920	0.918	0.931
	FID	37.11*	38.95	41.44	36.82	37.77	45.36
	Rec	0.115	0.141	0.035*	0.193	0.051	0.081
GTSRB	Fake	0.778	0.759*	0.812	0.826	0.833	0.788
	True	0.928	0.925	0.878	0.902	0.907	0.822*
	FID	54.45*	59.23	66.79	61.94	65.04	64.54
	Rec	0.299	0.377	0.126	0.405	0.102*	0.219
Cifar10	Fake	0.529	0.531	0.496*	0.533	0.496*	0.507
	True	0.500*	0.499*	0.499*	0.500*	0.499*	0.499*
	FID	33.76	35.66	33.37	35.74	32.30	31.97*
	Rec	0.287	0.400	0.121*	0.310	0.128	0.248
CelebA	Fake	0.662	0.501*	0.503	0.508	0.580	0.475
	True	0.498*	0.498*	0.497	0.498*	0.504	0.502*
	FID	17.39	13.36*	14.12	14.49	14.09	14.36
	Rec	0.237	0.290	0.056*	0.389	0.064	0.352

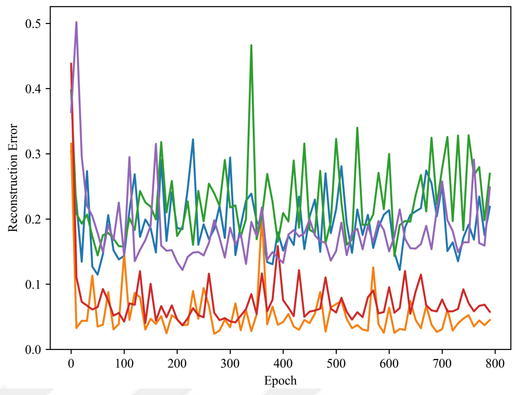
We also apply the hints to vanilla BiGAN and loss-sensitive BiGAN models and show the average reconstruction errors to compare all the variants. Figure 4.8 shows the results for the vanilla BiGAN. We can see that BiGAN-DS model gives the minimum reconstuction error closely followed by BiGAN-ConvD in all of the data sets. For Wasserstein BiGAN variants, in Figure 4.9 we see that WBiGAN-ConvD and WBiGAN-DS perform the best on all data sets. WBiGAN-Inception is competitive in models trained with UT-Zap50K and GTSRB data sets. WBiGAN-ConvGE does not significantly differ from WBiGAN and both does not perform well in terms of reconstruction error. We believe that DS model presents better results since the added loss term directly minimizes the reconstruction error. ConvD model surprisingly perform almost as well as DS and this shows that the discriminator network learns informative representations which can be obtained from its intermediate layers. For LS-BiGAN models trained with different hints, we see in Figure 4.10 that LS-BiGAN-DS model gives very good results on all data sets but surprisingly, LS-BiGAN-ConvD is the best for UT-Zap50K and GTSRB data sets.

Figure 4.11 shows the reconstruction errors of BiGAN variants trained with different loss functions, with the best performing hints for each data set. On MNIST, UT-Zap50K, Cifar10, and CelebA data sets, DS models gave the best reconstruction errors. On GTSRB data set, ConvD model performed the best. We see that the given hints work well with each loss function, except for MNIST that suffers mode collapse and performs the worst on vanilla BiGAN.

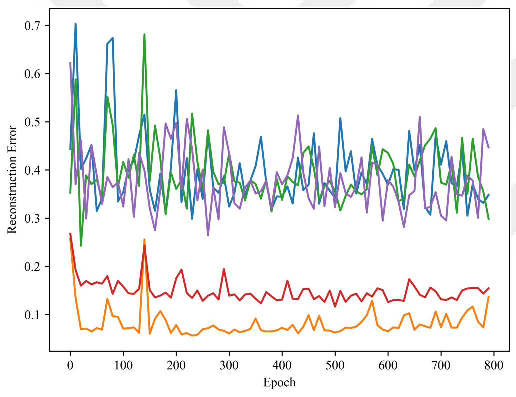
In Figure 4.12, we show randomly generated images using the best performing WBiGAN models for the five data sets; these images are generated with the models giving true and fake accuracies closest to 0.5. DS and ConvD models are used to generate the examples of MNIST and UT-Zap50K respectively. GTSRB examples are generated with WBiGAN-Inception which delivers the closest true and fake accuracies. Images generated from the CelebA data set has true and fake accuracies closest to 0.5 with WBiGAN. The only exception here is the Cifar10 data set where all models perform similarly in terms of true and fake accuracies. As a tie-breaker, we used the model with the minimum FID score to generate the examples, which is the WBiGAN-



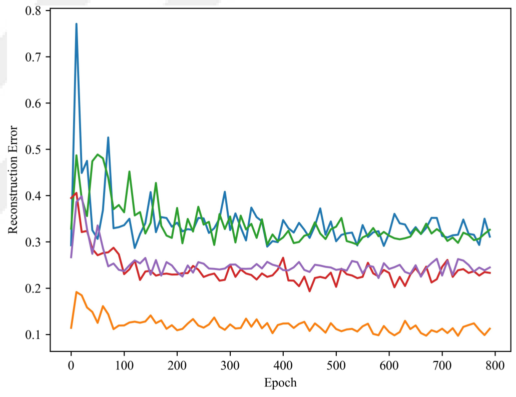
(a) MNIST



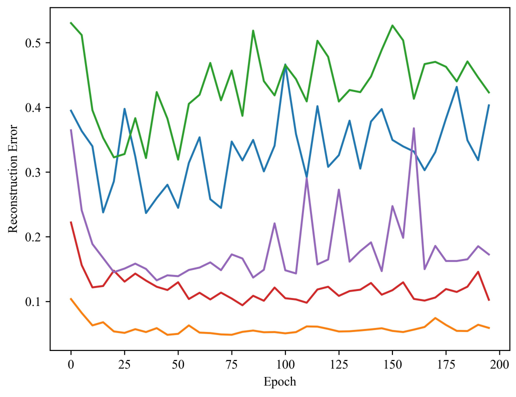
(b) UT-Zap50K



(c) GTSRB

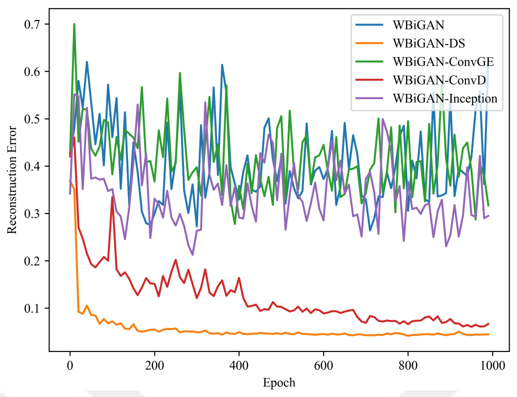


(d) Cifar10

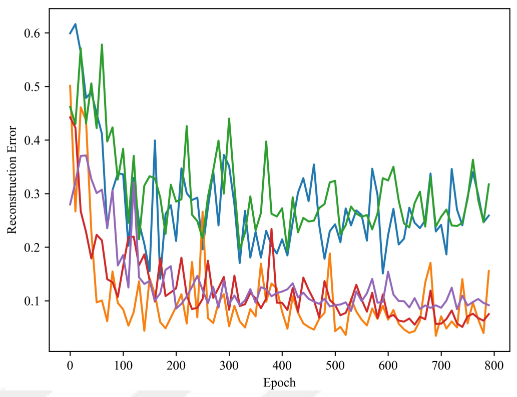


(e) CelebA

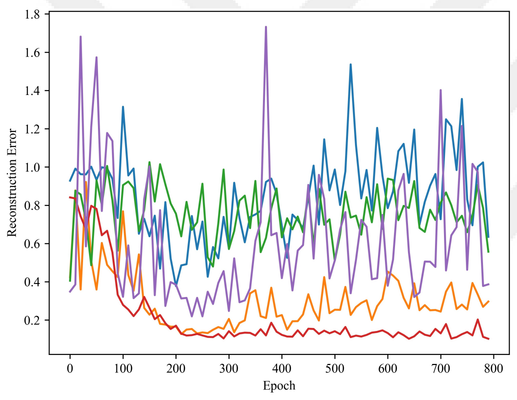
Figure 4.8. Reconstruction errors of Vanilla BiGAN models.



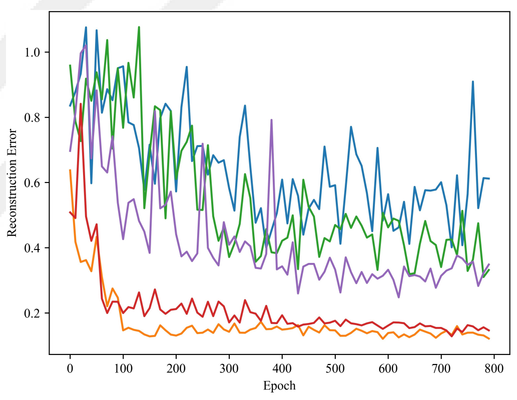
(a) MNIST



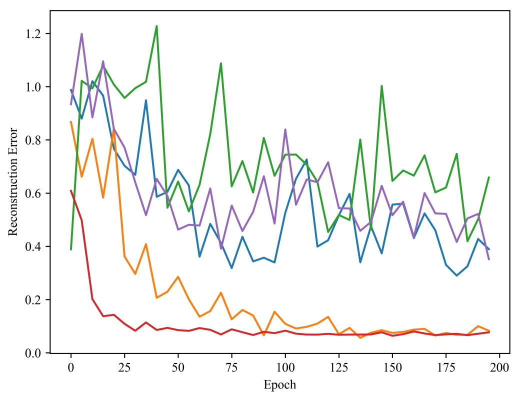
(b) UT-Zap50K



(c) GTSRB

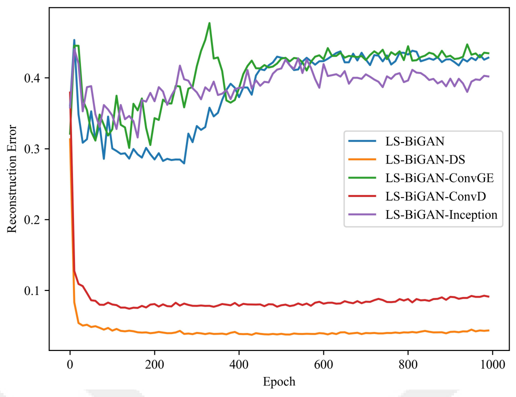


(d) Cifar10

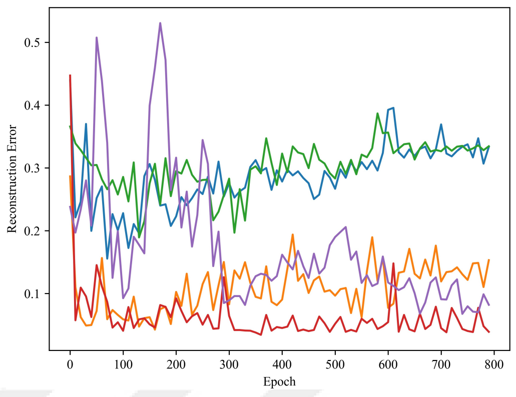


(e) CelebA

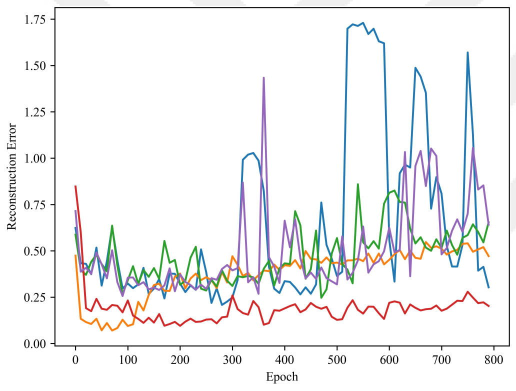
Figure 4.9. Reconstruction errors of Wasserstein BiGAN models.



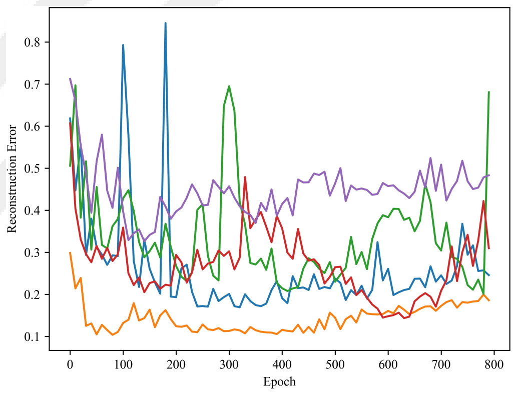
(a) MNIST



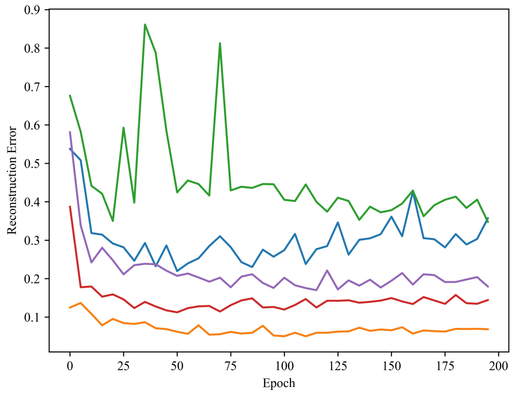
(b) UT-Zap50K



(c) GTSRB

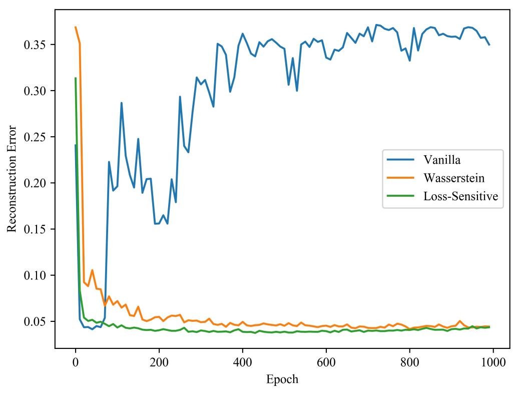


(d) Cifar10

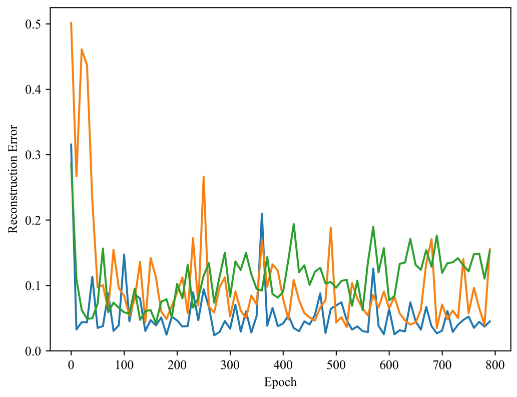


(e) CelebA

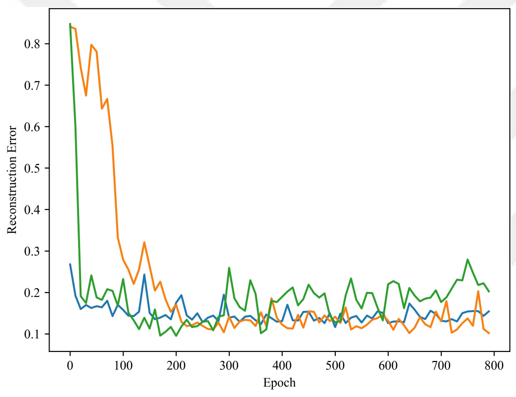
Figure 4.10. Reconstruction errors of Loss-Sensitive BiGAN models.



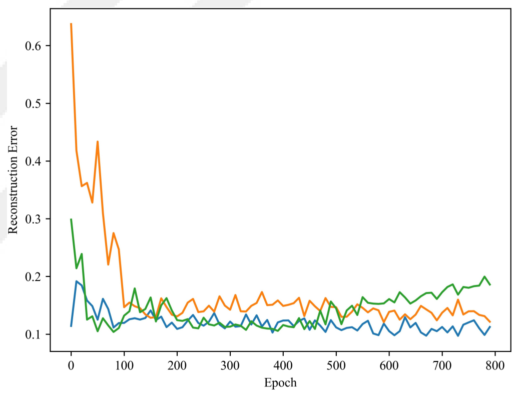
(a) MNIST - Data Space



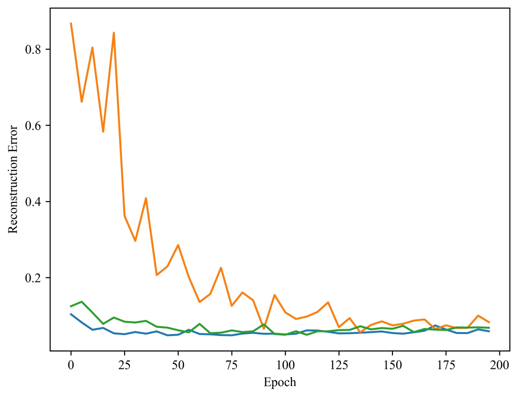
(b) UT-Zap50K - Data Space



(c) GTSRB - ConvD

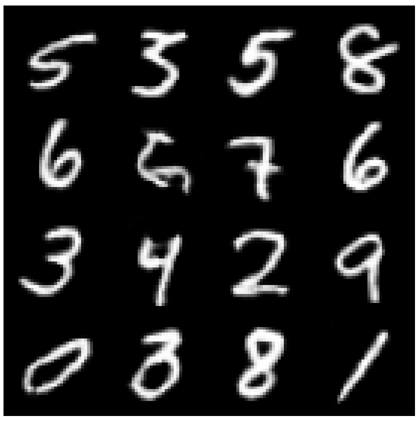


(d) Cifar10 - Data Space



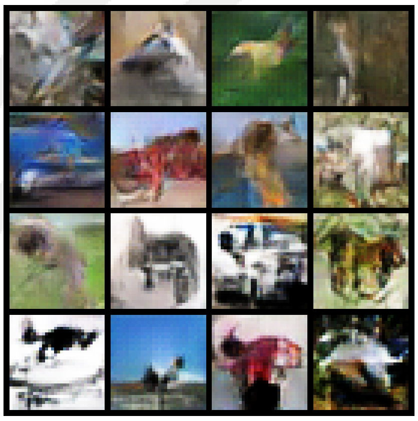
(e) CelebA - Data Space

Figure 4.11. Reconstruction errors of best performing hints on each data set with respect to loss functions.


(a) MNIST with WBiGAN-DS

(b) UT-Zap50K with WBiGAN-ConvD

(c) GTSRB with WBiGAN-Inception


(d) Cifar10 with WBiGAN-Inception

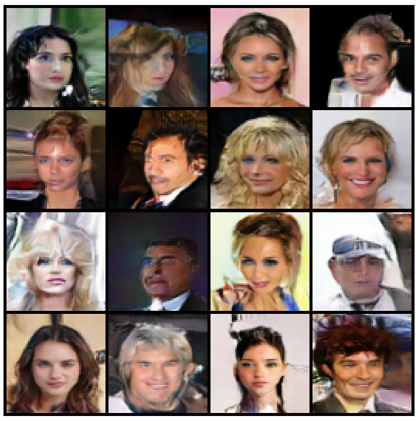

(e) CelebA with WBiGAN

Figure 4.12. Randomly generated images with best performing Wasserstein BiGAN models measured in terms of 1NN accuracy.

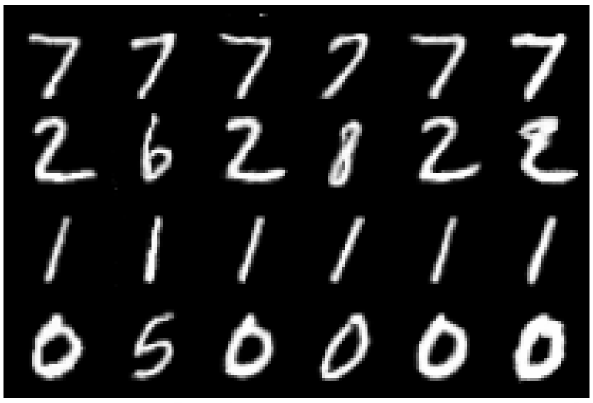
Inception.

In Table 4.4, we also show the minimum average reconstruction errors of all of the Wasserstein BiGAN models. We see that the WBiGAN-ConvD and WBiGAN-DS models greatly reduce the reconstruction error on all data sets. WBiGAN-Inception variant works well on UT-Zap50K and Cifar10 but it does not lead to a significant improvement on other data sets. We suspect that the WBiGAN-Inception model performs well with Cifar10 simply because of the fact that Inception-v3 model was trained on ImageNet which contains many classes from the Cifar10 data set. Finally, WBiGAN-ConvGE model does not improve the result on any data set. We believe that the main reason for this is that the intermediate representations in the convolutional layers of G and E networks do not provide sufficient information and hence do not define a good reconstruction criterion. WBiGAN-ConvD model takes advantage of the last convolutional layer of D and provides a better abstract representation and this makes the reconstruction of these representations more useful.

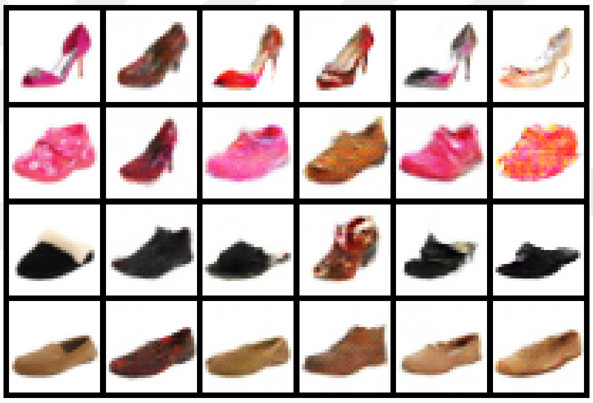
Figures 4.13 and 4.14 show example reconstructions of the same true test image with different models. Wasserstein BiGAN without any auxiliary reconstruction loss performs the worst. WBiGAN-ConvGE model also does not perform well. We can infer by visual inspection that the best models are WBiGAN-DS and WBiGAN-ConvD. The performance of the latter tells us that the discriminator learns a very good representation of the data.

4.5. Latent Space Interpolations

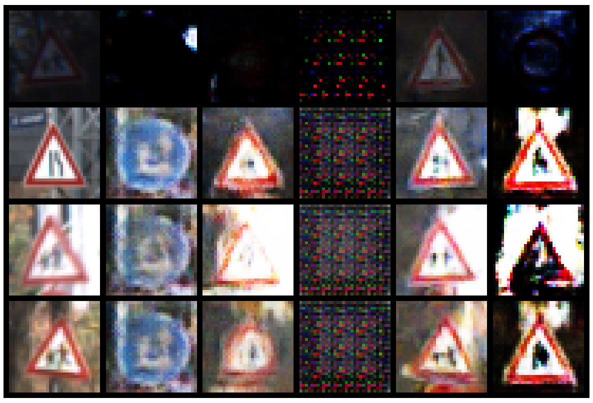
With trained BiGAN models, it is possible to interpolate in the dimensions of the latent space and observe how changes in z causes changes in x. In the case of training with image data, these interpolations can sometimes result in semantically meaningful changes. With trained BiGAN models we have an inverse mapping to the latent space which implies that we do not have to interpolate in arbitrary directions. To be more precise, we can extract the latent z_1 and z_2 of two true images x_1 and x_2 through the encoder network. Then we can generate new data by moving from z_1 to z_2 and see the



(a) MNIST

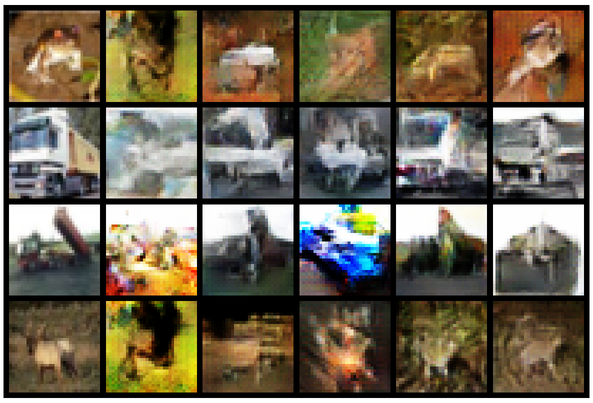


(b) UT-Zap50K

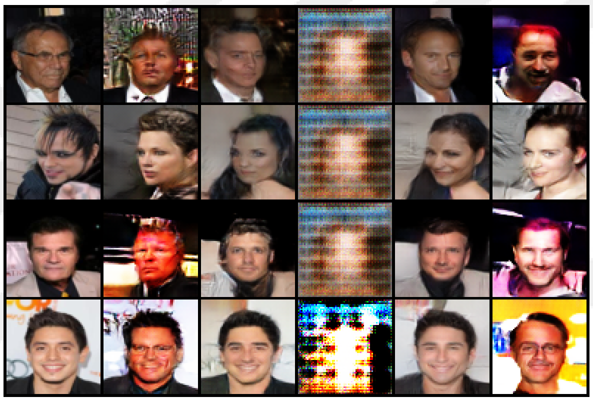


(c) GTSRB

Figure 4.13. Reconstruction examples of MNIST, UT-Zap50K and GTSRB. Models follow the order in Table 4.4 from left to right.



(a) Cifar10

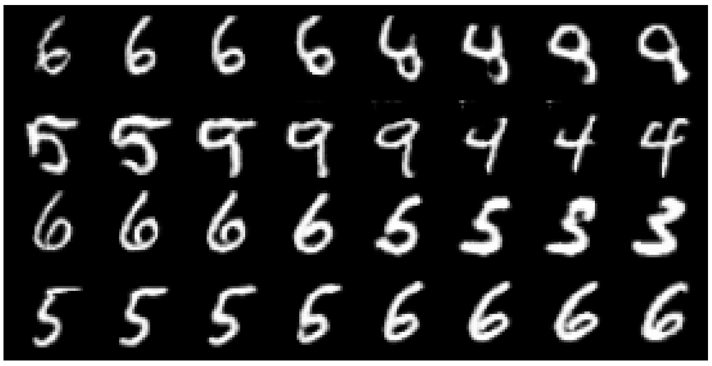


(b) CelebA

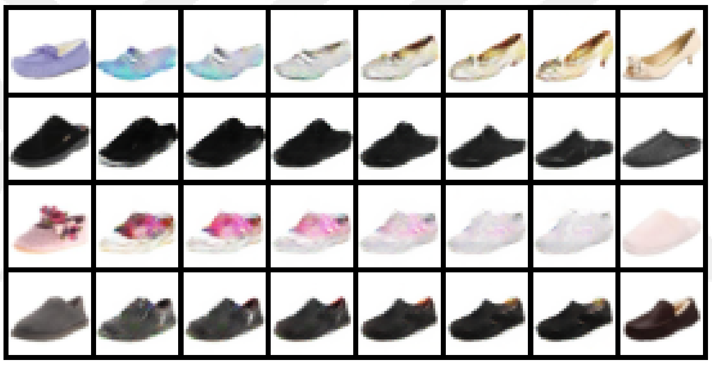
Figure 4.14. Reconstruction examples of Cifar10 and CelebA. Models follow the order in Table 4.4 from left to right.

resulting x generated in the way. If this procedure results in slowly altering semantic attributes in x, we can state that the encoder network has learned semantic features of the data. Some examples of interpolations using the BiGAN variants can be seen in Figures 4.15 and 4.16. These examples are generated using the best performing models for all data sets, measured in terms of the reconstruction error.

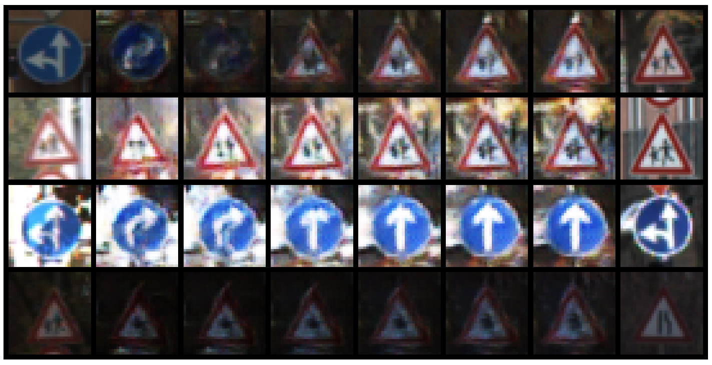
These results show that the interpolations slowly change some semantic attributes of the samples. In MNIST, when interpolating from a true image of the digit five to digit six changes the thickness and the shape of the digit. In CelebA, head orientations and genders change with interpolation with mixed attributes in the middle.



(a) MNIST with WBiGAN-DS

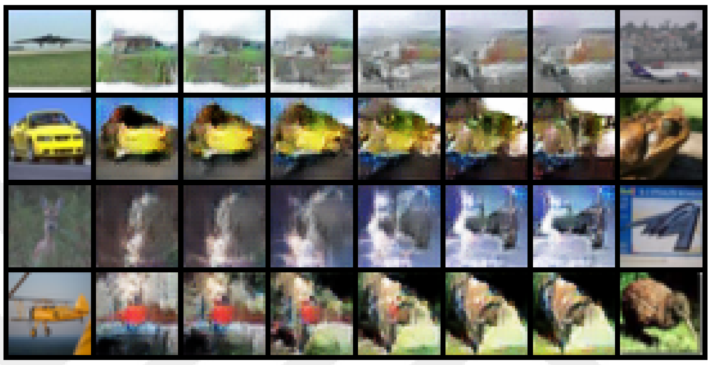


(b) UT-Zap50K with WBiGAN-DS

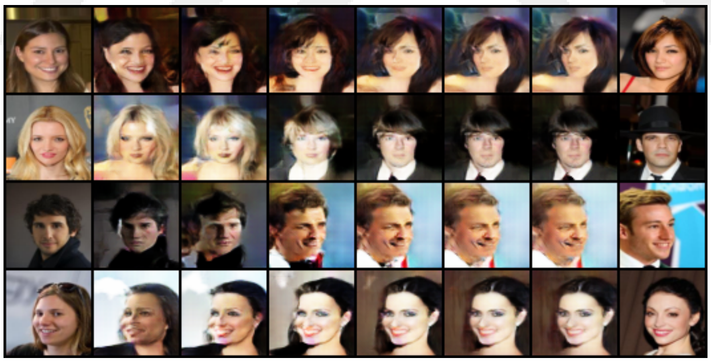


(c) GTSRB with WBiGAN-ConvD

Figure 4.15. Interpolation examples of MNIST, UT-Zap50K and GTSRB. The best performing model for each data set (in terms of reconstruction error) is used for obtaining the interpolations.



(a) Cifar10 with WBiGAN-DS



(b) CelebA with WBiGAN-DS

Figure 4.16. Interpolation examples of Cifar10 and CelebA. The best performing model for each data set (in terms of reconstruction error) is used for obtaining the interpolations.

5. CONCLUSIONS AND FUTURE WORK

5.1. Conclusions

- We generalized the bidirectional GAN by using the Wasserstein loss function that has recently been shown to work better than GAN proper. In our experiments on MNIST, UT-Zap50K, GTSRB, Cifar10 and CelebA data sets, we see that training BiGANs with Wasserstein loss leads to more stable training and generation of better quality images, when compared with vanilla BiGAN.
- We experimented with loss-sensitive GAN and adapted its loss function to BiGAN. We have seen in our experiments that while performing better than the vanilla BiGAN, loss-sensitive BiGAN did not demonstrate a significant superiority or inferiority against the Wasserstein loss.
- We evaluated the training stability, generation quality and diversity of our BiGAN models using the leave-one-out 1-NN test. We see that the 1-NN test allows us to monitor how well the true and fake samples overlap and gives us a good quantitative measure to evaluate different BiGAN variants.
- We showed that by taking advantage of the autoencoder structure of BiGANs, it is possible to reduce the reconstruction error without compromising the generated image quality and diversity. Since the purpose of BiGANs is to learn an inverse mapping from the data space to the latent space, we believe this improvement is essential for BiGANs to be able to learn better abstract representations of the data.
- Specifically, our WBiGAN-DS and WBiGAN-ConvD variants dramatically reduce the reconstruction error on all data sets when compared with the Wasserstein BiGAN without hints, paving the way to improved BiGAN models which can learn better representations and features of the data in an unsupervised manner.
- We believe that the success of WBiGAN-ConvD model shows us that the discriminator learns the structure of the data well enough to provide good intermediate representations that improve the quality of the model when used in reconstruction.

- We have not seen any improvement in reconstruction quality with WBiGAN-ConvGE model. We observed that reconstructing the hidden representations of intermediate layers of the generator and the encoder do not provide enough information to improve the reconstruction quality.
- We examined our proposed reconstruction criteria not only with Wasserstein BiGAN, also with vanilla and loss-sensitive BiGAN. We conclude from our exhaustive experiments that all three BiGAN variants are affected positively in terms of reconstruction quality when trained with the help of our proposed hints.
- We analyzed the latent space interpolations of best performing Wasserstein BiGAN models (in terms of reconstruction quality) to validate our assumption that a better reconstruction quality leads to a better learned representations of the data. We have observed semantic changes in images when interpolating from the latent space of one image to the other image.
- We presented a generator, an encoder, and a discriminator architecture and training setup for BiGAN which performs well on all five data sets without the need of exhaustive model tuning. These architectures are also easily scalable to images with higher resolutions.

To recap, we can state that for the purpose of generating high quality and diverse data, Wasserstein BiGAN works quite well. If the purpose is to train BiGANs for reconstructions or obtaining meaningful features of the data, our experimental results indicate that WBiGAN-DS or WBiGAN-ConvD variants we propose lead to improved reconsturction and meaningful semantic features of the data.

5.2. Future Work

- There are other loss functions that have been recently proposed for GANs, such as least-squares GAN [35], boundary equilibrium GAN [61], and energy-based GAN [37]; testing their BiGAN versions together with the hints we propose is one possible future research direction.

- In this work, we have used the pre-trained Inception-v3 model to obtain features for reconstructions on the feature space. Pre-trained parameters of other famous architectures, such as VGG-16 [3] or ResNet [5], can also be utilized to obtain different features.
- The scalability of our models and BiGANs in general can be examined using higher resolution images. For example, large-scale and high resolution data sets such as ImageNet [40] or CelebA-HQ [32] can be used to train BiGAN proper and our proposed variants.
- We can perform semi-supervised or supervised learning tasks such as classification using the latent features obtained by our improved models. This will indicate whether the latent z are useful in a supervised setting. Note that if the labels are present in the data, this label information can also be used to define new supervised hints in addition to, or instead of the unsupervised hint of the reconstruction loss.

REFERENCES

1. Krizhevsky, A., I. Sutskever and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

2. Rumelhart, D. E., G. E. Hinton and R. J. Williams, “Learning representations by back-propagating errors”, *Nature*, Vol. 323, No. 6088, p. 533, 1986.

3. Simonyan, K. and A. Zisserman, “Very deep convolutional networks for large-scale image recognition”, *arXiv preprint arXiv:1409.1556*, 2014.

4. Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, “Going deeper with convolutions”, *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

5. He, K., X. Zhang, S. Ren and J. Sun, “Deep residual learning for image recognition”, *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

6. Sutskever, I., O. Vinyals and Q. V. Le, “Sequence to sequence learning with neural networks”, *Advances in Neural Information Processing Systems*, pp. 3104–3112, 2014.

7. Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, “Google’s neural machine translation system: Bridging the gap between human and machine translation”, *arXiv preprint arXiv:1609.08144*, 2016.

8. Bengio, Y., R. Ducharme, P. Vincent and C. Jauvin, “A neural probabilistic language model”, *Journal of Machine Learning Research*, Vol. 3, pp. 1137–1155, 2003.

9. Kumar, A., O. Irsoy, P. Ondruska, M. Iyyer, J. Bradbury, I. Gulrajani, V. Zhong, R. Paulus and R. Socher, “Ask me anything: Dynamic memory networks for natural language processing”, *International Conference on Machine Learning*, pp. 1378–1387, 2016.

10. Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado and J. Dean, “Distributed representations of words and phrases and their compositionality”, *Advances in Neural Information Processing Systems*, pp. 3111–3119, 2013.

11. Hinton, G., L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups”, *IEEE Signal Processing Magazine*, Vol. 29, No. 6, pp. 82–97, 2012.

12. Xiong, W., J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu and G. Zweig, “Achieving human parity in conversational speech recognition”, *arXiv preprint arXiv:1610.05256*, 2016.

13. Szegedy, C., W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow and R. Fergus, “Intriguing properties of neural networks”, *arXiv preprint arXiv:1312.6199*, 2013.

14. Goodfellow, I. J., J. Shlens and C. Szegedy, “Explaining and harnessing adversarial examples”, *arXiv preprint arXiv:1412.6572*, 2014.

15. Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, “Generative adversarial nets”, *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

16. Arjovsky, M. and L. Bottou, “Towards principled methods for training generative adversarial networks”, *arXiv preprint arXiv:1701.04862*, 2017.

17. Arjovsky, M., S. Chintala and L. Bottou, “Wasserstein GAN”, *arXiv preprint*

arXiv:1701.07875, 2017.

18. Salimans, T., I. Goodfellow, W. Zaremba, V. Cheung, A. Radford and X. Chen, “Improved techniques for training GANs”, *Advances in Neural Information Processing Systems*, pp. 2234–2242, 2016.

19. Warde-Farley, D. and Y. Bengio, “Improving Generative Adversarial Networks With Denoising Feature Matching”, *International Conference on Learning Representations*, 2017.

20. Qi, G.-J., “Loss-sensitive generative adversarial networks on Lipschitz densities”, *arXiv preprint arXiv:1701.06264*, 2017.

21. Mirza, M. and S. Osindero, “Conditional generative adversarial nets”, *arXiv preprint arXiv:1411.1784*, 2014.

22. Chen, X., Y. Duan, R. Houthooft, J. Schulman, I. Sutskever and P. Abbeel, “InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets”, *Advances in Neural Information Processing Systems*, pp. 2172–2180, 2016.

23. Zhu, J.-Y., T. Park, P. Isola and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks”, *arXiv preprint arXiv:1703.10593*, 2017.

24. Dong, H., P. Neekhara, C. Wu and Y. Guo, “Unsupervised image-to-image translation with generative adversarial networks”, *arXiv preprint arXiv:1701.02676*, 2017.

25. Ledig, C., L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. P. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network.”, *Computer Vision and Pattern Recognition (CVPR)*, Vol. 2, p. 4, 2017.

26. Yeh, R. A., C. Chen, T.-Y. Lim, A. G. Schwing, M. Hasegawa-Johnson and M. N. Do, “Semantic Image Inpainting with Deep Generative Models.”, *Computer Vision and Pattern Recognition (CVPR)*, Vol. 2, p. 4, 2017.

27. Wu, H., S. Zheng, J. Zhang and K. Huang, “GP-GAN: Towards realistic high-resolution image blending”, *arXiv preprint arXiv:1703.07195*, 2017.

28. Luc, P., C. Couprie, S. Chintala and J. Verbeek, “Semantic segmentation using adversarial networks”, *arXiv preprint arXiv:1611.08408*, 2016.

29. Zhang, H., T. Xu, H. Li, S. Zhang, X. Huang, X. Wang and D. Metaxas, “Stack-GAN: Text to photo-realistic image synthesis with stacked generative adversarial networks”, *arXiv preprint arXiv:1612.03242*, 2017.

30. Reed, S., Z. Akata, X. Yan, L. Logeswaran, B. Schiele and H. Lee, “Generative adversarial text to image synthesis”, *arXiv preprint arXiv:1605.05396*, 2016.

31. Zhang, H., I. Goodfellow, D. Metaxas and A. Odena, “Self-Attention Generative Adversarial Networks”, *arXiv preprint arXiv:1805.08318*, 2018.

32. Karras, T., T. Aila, S. Laine and J. Lehtinen, “Progressive growing of GANs for improved quality, stability, and variation”, *arXiv preprint arXiv:1710.10196*, 2017.

33. Brock, A., J. Donahue and K. Simonyan, “Large scale GAN training for high fidelity natural image synthesis”, *arXiv preprint arXiv:1809.11096*, 2018.

34. Karras, T., S. Laine and T. Aila, “A style-based generator architecture for generative adversarial networks”, *arXiv preprint arXiv:1812.04948*, 2018.

35. Mao, X., Q. Li, H. Xie, R. Y. Lau, Z. Wang and S. P. Smolley, “Least squares generative adversarial networks”, *2017 IEEE International Conference on Computer Vision*, pp. 2813–2821, 2017.

36. Che, T., Y. Li, A. P. Jacob, Y. Bengio and W. Li, “Mode regularized generative

adversarial networks", *arXiv preprint arXiv:1612.02136*, 2016.

37. Zhao, J., M. Mathieu and Y. LeCun, "Energy-based generative adversarial network", *arXiv preprint arXiv:1609.03126*, 2016.

38. Metz, L., B. Poole, D. Pfau and J. Sohl-Dickstein, "Unrolled generative adversarial networks", *arXiv preprint arXiv:1611.02163*, 2016.

39. Gulrajani, I., F. Ahmed, M. Arjovsky, V. Dumoulin and A. C. Courville, "Improved training of Wasserstein GANs", *Advances in Neural Information Processing Systems*, pp. 5767–5777, 2017.

40. Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database", *IEEE Conference on Computer Vision and Pattern Recognition, 2009.*, pp. 248–255, 2009.

41. Xu, Q., G. Huang, Y. Yuan, C. Guo, Y. Sun, F. Wu and K. Weinberger, "An empirical study on evaluation metrics of generative adversarial networks", *arXiv preprint arXiv:1806.07755*, 2018.

42. Heusel, M., H. Ramsauer, T. Unterthiner, B. Nessler and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local Nash equilibrium", *Advances in Neural Information Processing Systems*, pp. 6626–6637, 2017.

43. Lopez-Paz, D. and M. Oquab, "Revisiting classifier two-sample tests", *arXiv preprint arXiv:1610.06545*, 2016.

44. Hinton, G. E. and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks", *Science*, Vol. 313, No. 5786, pp. 504–507, 2006.

45. Kingma, D. P. and M. Welling, "Auto-encoding variational Bayes", *arXiv preprint arXiv:1312.6114*, 2013.

46. Donahue, J., P. Krähenbühl and T. Darrell, "Adversarial feature learning", *arXiv*

preprint arXiv:1605.09782, 2016.

47. Dumoulin, V., I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky and A. Courville, “Adversarially learned inference”, *arXiv preprint arXiv:1606.00704*, 2016.

48. Perarnau, G., J. van de Weijer, B. Raducanu and J. M. Álvarez, “Invertible conditional GANs for image editing”, *arXiv preprint arXiv:1611.06355*, 2016.

49. Jaiswal, A., W. AbdAlmageed, Y. Wu and P. Natarajan, “Bidirectional Conditional Generative Adversarial Networks”, *arXiv preprint arXiv:1711.07461*, 2017.

50. Gan, Z., L. Chen, W. Wang, Y. Pu, Y. Zhang, H. Liu, C. Li and L. Carin, “Triangle generative adversarial networks”, *Advances in Neural Information Processing Systems*, pp. 5247–5256, 2017.

51. Bao, R., S. Liang and Q. Wang, “Featurized Bidirectional GAN: Adversarial Defense via Adversarially Learned Semantic Inference”, *arXiv preprint arXiv:1805.07862*, 2018.

52. Li, Z., W. Wang and Y. Zhao, “Image Translation by Domain-Adversarial Training”, *Computational Intelligence and Neuroscience*, Vol. 2018, 2018.

53. Odena, A., C. Olah and J. Shlens, “Conditional image synthesis with auxiliary classifier GANs”, *arXiv preprint arXiv:1610.09585*, 2016.

54. Larsson, G., M. Maire and G. Shakhnarovich, “Learning representations for automatic colorization”, *European Conference on Computer Vision*, pp. 577–593, Springer, 2016.

55. Larsen, A. B. L., S. K. Sønderby, H. Larochelle and O. Winther, “Autoencoding beyond pixels using a learned similarity metric”, *arXiv preprint arXiv:1512.09300*, 2015.

56. Rosca, M., B. Lakshminarayanan, D. Warde-Farley and S. Mohamed, "Variational approaches for auto-encoding generative adversarial networks", *arXiv preprint arXiv:1706.04987*, 2017.

57. Mescheder, L., S. Nowozin and A. Geiger, "Adversarial variational Bayes: Unifying variational autoencoders and generative adversarial networks", *arXiv preprint arXiv:1701.04722*, 2017.

58. Makhzani, A., J. Shlens, N. Jaitly, I. Goodfellow and B. Frey, "Adversarial autoencoders", *arXiv preprint arXiv:1511.05644*, 2015.

59. Srivastava, A., L. Valkov, C. Russell, M. U. Gutmann and C. Sutton, "VEEGAN: Reducing mode collapse in GANs using implicit variational learning", *Advances in Neural Information Processing Systems*, pp. 3308–3318, 2017.

60. Brock, A., T. Lim, J. M. Ritchie and N. Weston, "Neural photo editing with introspective adversarial networks", *arXiv preprint arXiv:1609.07093*, 2016.

61. Berthelot, D., T. Schumm and L. Metz, "BEGAN: boundary equilibrium generative adversarial networks", *arXiv preprint arXiv:1703.10717*, 2017.

62. Intrator, Y., G. Katz and A. Shabtai, "MDGAN: Boosting Anomaly Detection Using Multi-Discriminator Generative Adversarial Networks", *arXiv preprint arXiv:1810.05221*, 2018.

63. Bang, D. and H. Shim, "High Quality Bidirectional Generative Adversarial Networks", *arXiv preprint arXiv:1805.10717*, 2018.

64. Kim, T., M. Cha, H. Kim, J. K. Lee and J. Kim, "Learning to discover cross-domain relations with generative adversarial networks", *arXiv preprint arXiv:1703.05192*, 2017.

65. Yi, Z., H. R. Zhang, P. Tan and M. Gong, "DualGAN: Unsupervised Dual Learning

for Image-to-Image Translation.", *International Conference on Computer Vision (ICCV)*, pp. 2868–2876, 2017.

66. Li, C., H. Liu, C. Chen, Y. Pu, L. Chen, R. Henao and L. Carin, "Alice: Towards understanding adversarial learning for joint distribution matching", *Advances in Neural Information Processing Systems*, pp. 5495–5503, 2017.

67. Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception architecture for computer vision", *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

68. LeCun, Y., L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition", *IEEE*, Vol. 86, No. 11, pp. 2278–2324, 1998.

69. Yu, A. and K. Grauman, "Fine-Grained Visual Comparisons with Local Learning", *Computer Vision and Pattern Recognition (CVPR)*, June 2014.

70. Yu, A. and K. Grauman, "Semantic Jitter: Dense Supervision for Visual Comparisons via Synthetic Images", *International Conference on Computer Vision (ICCV)*, Oct 2017.

71. Stallkamp, J., M. Schlipsing, J. Salmen and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition", *Neural Networks*, Vol. 32, pp. 323–332, 2012.

72. Krizhevsky, A. and G. Hinton, *Learning multiple layers of features from tiny images*, Tech. rep., Citeseer, 2009.

73. Liu, Z., P. Luo, X. Wang and X. Tang, "Deep learning face attributes in the wild", *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3730–3738, 2015.

74. Radford, A., L. Metz and S. Chintala, "Unsupervised representation learn-

ing with deep convolutional generative adversarial networks", *arXiv preprint arXiv:1511.06434*, 2015.