Pattern Recognition 103 (2020) 107320

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Training bidirectional generative adversarial networks with hints

Uras Mutlu*, Ethem Alpaydın

Department of Computer Engineering, Boğaziçi University, TR-34342 İstanbul, Turkey

ARTICLE INFO

Article history:
Received 23 May 2019
Revised 15 January 2020
Accepted 28 February 2020
Available online 29 February 2020

MSC:
00-01
99-00

Keywords:
Generative Modeling
Generative Adversarial Networks
Unsupervised Learning
Autoencoders
Neural Networks
Deep Learning

ABSTRACT

The generative adversarial network (GAN) is composed of a generator and a discriminator where the generator is trained to transform random latent vectors to valid samples from a distribution and the discriminator is trained to separate such "fake" examples from true examples of the distribution, which in turn forces the generator to generate better fakes. The bidirectional GAN (BiGAN) also has an encoder working in the inverse direction of the generator to produce the latent space vector for a given example. This added encoder allows defining auxiliary reconstruction losses as hints for a better generator. On five widely-used data sets, we showed that BiGANs trained with the Wasserstein loss and augmented with hints learn better generators in terms of image generation quality and diversity, as measured numerically by the 1-nearest neighbor test, Fréchet inception distance, and reconstruction error, and qualitatively by visually analyzing the generated samples.

1. Introduction

1.1. Generative modeling

In generative modeling, we have a data set of $\{x^t\}_t$ drawn from an unknown probability distribution $p(x)$ and we would like to be able to generate new x that also look like they have been drawn from $p(x)$. For example, x^t may be the face images of a collection of people and we would like to be able to generate new face images; these new synthetic images would be legitimate faces but of people that do not exist.

The typical approach would be to learn some estimator to $p(x)$ (e.g., using a Gaussian distribution) and then to sample from that estimator. The approach we have in this paper defines generative modeling as a mapping task where a generator function takes some low-dimensional z drawn from a given $p(z)$ as input and transforms it into a valid instance x from $p(x)$. All the structure that $p(x)$ has (for example, all the requirements of being a face image) needs to be captured during learning so that the newly generated x also reflect those.

1.2. The autoencoder

In many real-world applications that involve, for instance, images, speech, or text, 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 much lower-dimensional z-space without any 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 goes in the opposite direction of PCA.

Unsupervised dimensionality reduction can be learned using the neural network architecture called the autoencoder (AE) (Fig. 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 the encoder. In the simplest case, both the encoder and the 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 realizes *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

* Corresponding author.
E-mail addresses: uras.mutlu@boun.edu.tr (U. Mutlu), alpaydin@boun.edu.tr (E. Alpaydın).

https://doi.org/10.1016/j.patcog.2020.107320
0031-3203/

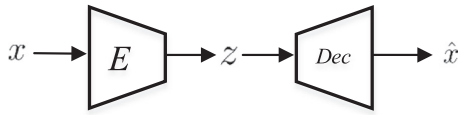


Fig. 1. The autoencoder (AE) is composed of an encoder E and a decoder Dec. The encoder maps x to latent z and the decoder reconstructs x from z.

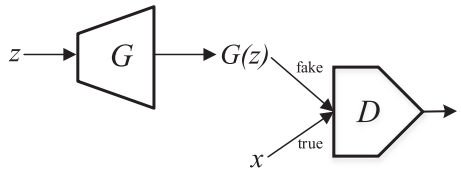


Fig. 2. The generative adversarial network (GAN) is composed of a generator G and a discriminator D. G generates fake x from z and D learns to discriminate fake $G(z)$ from true x.

image data, the encoder starts with one or more convolution layers that successively downsample 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 upsampling deconvolution layers to generate the image back again.

The autoencoder is not a generative model; for any x, we can find the corresponding z and then reconstruct x, but we have no way of generating new x outside of the training set. In the variational autoencoder (VAE) [1], 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 enforce this. Once training is done, we can sample from this $p(z)$ and use the decoder to generate new x.

1.3. The generative adversarial network (GAN)

In this paper, we extend the generative adversarial network (GAN) [2] that has recently been shown to work better than the VAE as a generative model. The original GAN model is composed of two networks, a generator G and a discriminator D (Fig. 2). Both G and D are deep neural networks with convolutional and dense layers as appropriate. The generator takes a latent vector z as input and generates an observation vector x, where 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 new z from $p(z)$ and passing them through 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 fakes so well that the discriminator cannot tell them apart from the true samples. The two networks G and D play an adversarial game and gradually improve their abilities: As G gets to generate better fakes, D gets better at detecting them, which in turn forces G to get even better, and so on.

The following log-likelihood criterion is maximized by D and minimized by G:

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

Here, 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 $p(z)$.

Since its inception, the GAN model and its many variants have been successfully used in many applications, in image, video, text and music generation; see [3] for a survey.

1.4. Extending GAN with hints

Despite their various successful applications, it has been seen that training GANs is difficult and several empirical tips and tricks have been proposed to improve convergence, such as label smoothing, mini-batch discrimination, and feature matching [4].

Our approach in this paper involves adding an auxiliary loss term to that of GAN and optimize the resulting augmented criterion in training. This added term provides a "hint" that directs the learning process towards a better generator. We propose a general framework that defines how such hints can be included in training and show four variants. To be able to define the type of hint we have, we use the bidirectional form of the GAN. The original GAN 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) [5,6] also includes an *encoder* component, and this encoder allows us to define various loss functions to train better generators. This new encoder component ($x \rightarrow z$), which is also implemented as a deep neural network, works just like the encoder of the AE, and the generator of the GAN ($z \rightarrow x$) works just like the decoder of AE, and we use this correspondence in defining the auxiliary loss functions.

In addition to training a better generator, having an encoder can also be useful in different scenarios: Once we have such a mechanism, by investigating how z changes as x is changed, we can assign meaning to the different dimensions of z [7] and this allows knowledge extraction: For example, we can do "vector algebra" where abstract features such as putting on glasses corresponds to adding a vector in the z-space. Because such an encoder works as a dimensionality reducer trained with unlabeled data, it can be used as a preprocessor before a later classifier or regressor in a semi-supervised setting.

In Section 2.1, we discuss the BiGAN model trained using the original log-likelihood criterion as well as with the Wasserstein loss introducing the Wasserstein BiGAN. We introduce the auxiliary reconstruction criteria for training BiGANs in Section 3. Our experimental results on five image data sets are given in Section 4 and we conclude in Section 5.

2. Training the bidirectional GAN

2.1. The bidirectional GAN

The original GAN can generate x for any z but does not have an inverse mapper for generating the corresponding z for any given x. The *Bidirectional GAN* (BiGAN) [5] and the equivalent *Adversarially Learned Inference* (ALI) [6] models were proposed independently and contain also an encoder component E mapping true x to z (Fig. 3). Unlike the GAN where the discriminator sees only x as input, in the 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 (it is sampled from $p(z)$) and its corresponding x is generated by the generator G.

The encoder E is also implemented as a deep neural network and (as in the 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 discriminator. 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.

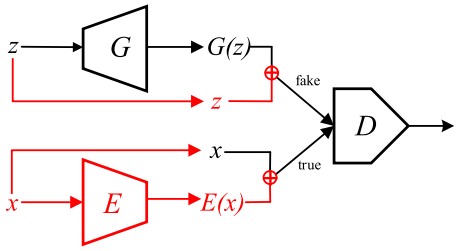


Fig. 3. The bidirectional GAN also has an encoder E that learns to map true x to latent $z \equiv E(x)$. The discriminator takes the pair z, x as input; $\oplus$ in the figure denotes vector concatenation. The red components show the additions to the original GAN.

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

$$\mathcal{L}_{BiGAN} = \sum_{x^t} \log D(x^t, E(x^t)) + \sum_{z^t \sim p(z)} \log(1 - D(G(z^t), z^t)) \quad (2)$$

except that D sees both the input and its latent representation.

2.2. The Wasserstein BiGAN

For the case of the original GAN, it has been shown that minimizing the Wasserstein distance leads to more stable training [8,9], and better quality images are produced when used for generating face images [10]. 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, which in our case, is from that of the fake samples to that of the true samples. For the original GAN, the Wasserstein loss is defined as:

$$\mathcal{L}_{WGAN} = \sum_{x^t} D(x^t) - \sum_{z^t \in p(z)} D(G(z^t)) \quad (3)$$

for true samples x^t and fake samples generated from z^t. Unlike the original GAN where D is a 0/1 classifier estimating the posterior probability that its input is a true sample, in the Wasserstein GAN D is a regressor estimating the "trueness" score of its input. In terms of implementation, the single output of D in the original GAN uses the sigmoid nonlinearity whereas that of the Wasserstein GAN is linear.

The Wasserstein loss of Eq. (3) is the difference of the trueness scores of true and fake samples. D is trained to maximize this and G is trained to minimize it. D wants its output $D(x^t)$ to be higher for true samples x^t than for generated fake samples $D(G(z^t))$, and G wants the opposite.

We adapt this for BiGAN

$$\mathcal{L}_{WBiGAN} = \sum_{x^t} D(x^t, E(x^t)) - \sum_{z^t \in p(z)} D(G(z^t), z^t) \quad (4)$$

where again the difference is that D sees both representations.

An improved version of the Wasserstein GAN includes an additional gradient penalty term to enforce a 1-Lipschitz constraint on the gradients of D [11]

$$\mathcal{L}_{GP} = \sum_{z^t \in p(z)} (\|\nabla_{G(z)} D(G(z))\|_2 - 1)^2 \quad (5)$$

which we adapt for BiGAN as

$$\mathcal{L}_{JGP} = \sum_{z^t \in p(z)} (\|\nabla_{G(z)} D(G(z), z)\|_2 - 1)^2 \quad (6)$$

An approach that is very similar to the Wasserstein loss is the *loss-sensitive GAN* (LS-GAN) proposed independently [12], which not only looks at the difference as in the Wasserstein loss but also imposes a margin. Our experimental results using the loss-sensitive approach, on GAN or BiGAN, have not shown significant difference from those using the Wasserstein loss, and for the sake of brevity we do not include those results in later sections.

Table 1
The loss terms of the hints used.

Name	Definition	Formula
DS	Data space	$\|x - G(E(x))\|^2$
FeaGE	Feature spaces of G and E	$\|G_n(E(x)) - E_m(x)\|^2$
FeaD	Feature space of D	$\|D_c(x) - D_c(G(E(x)))\|^2$
FeaInc	Feature space of Inception-v3 network	$\|I_c(x) - I_c(G(E(x)))\|^2$

3. Hints for improving generation quality

3.1. Motivation

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

1. 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 known to be more difficult because there is less feedback.
2. There are two models D and G to train and hence the problem of model selection is multiplied by two. Both are typically implemented by many-layered deep networks and the depth and width of both should be fine-tuned to the task.
3. 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.
4. There are a number of measures that have been proposed to assess the quality of a GAN solution, each making different assumptions and partially informative. Researchers typically also generate and display a bunch of examples that are evaluated visually, but that is subjective and cannot be automated.

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

Our approach 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 direct the learning process to get a better G. Both G and D are implemented as deep neural networks that are highly over-parameterized and from an optimization perspective, such additional terms can be thought of as regularizers. Equivalently, from a learning perspective, they can 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 Fig. 3), 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 the original GAN.

3.2. Auxiliary losses as hints

In the following subsections, we discuss four criteria leading to four different BiGAN variants, some of which overlap with or are similar to existing methods. The basic idea is to define an implicit autoencoder using the generator and the added encoder, and with that autoencoder we can define a reconstruction error. The four variants differ in the space where this reconstruction is done.

These reconstruction errors define four auxiliary loss terms; see Table 1. They are added to the BiGAN loss, which in our case is the Wasserstein loss, and the resulting augmented loss is minimized. Note that such auxiliary losses are hints on G (and E), and as such

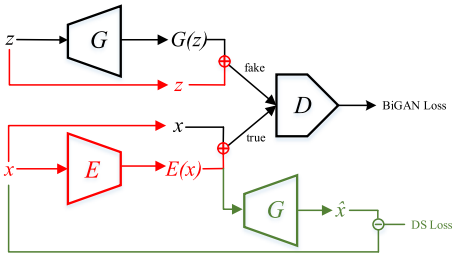


Fig. 4. The training instance x is passed through first E then G to get its reconstruction $\hat{x}$. The L_2-norm of the difference between x and its reconstruction $\hat{x}^t \equiv G(E(x^t))$ is used as the data-space hint (DS). The green components show the additions to BiGAN to calculate the hint.

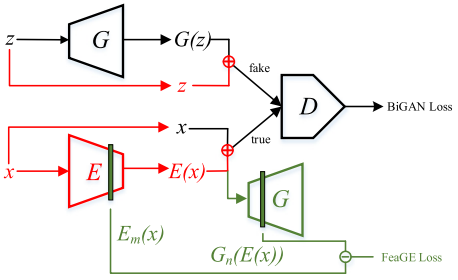


Fig. 5. The training instance x is passed through E and then G to obtain $\hat{x}$. Instead of comparing raw x and $\hat{x}$, the activations at intermediate layer m of E and the corresponding layer n of G are compared.

are used to update G (and E), to learn a better generator (and encoder); even if they use D, they do not cause any update on D.

3.3. Data space

The most straightforward is to use the reconstruction loss in the original data space (e.g., pixels in images), equal to what is minimized by the autoencoder:

$$\mathcal{L}_{DS} = \sum_{x^t} \|x^t - \hat{x}^t\|^2 = \sum_{x^t} \|x^t - G(E(x^t))\|^2 \tag{7}$$

where x is a true training instance and $\hat{x}^t \equiv G(E(x^t))$ is its reconstruction using first E as the encoder and then G as the decoder (see Fig. 4).

3.4. Feature 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 higher-order features. Because G and E networks are taken as the exact inverses of each other architecturally, we can define a correspondence between the intermediate layers of G and E: Increasing layers of G correspond to decreasing layers of E; one can use a convolution or a dense layer for this purpose.

For a training instance x^t, let $G_n(E(x^t))$ and $E_m(x^t)$ be the vectors of activation at the convolutional layer n of G and the corresponding convolutional layer m of E respectively. Then we can define the following a loss in terms of their difference (see Fig. 5):

$$\mathcal{L}_{FeaGE} = \sum_{x^t} \|G_n(E(x^t)) - E_m(x^t)\|^2 \tag{8}$$

3.5. Feature space of D

The discriminator D takes x as input and processes it in its many layers learning successively more abstract representations. So another possibility is to use the early layers of D as a preprocessor to get such an abstract representation. We pass x and $\hat{x} \equiv G(E(x))$ through early layers of D and then compare the activations in an intermediate layer of D.

$D_c(x)$ denotes the vector of activations at layer c of D and we define a loss in terms of the difference calculated for a training instance and its reconstruction (see Fig. 6):

$$\mathcal{L}_{FeaD} = \sum_{x^t} \|D_c(x^t) - D_c(\hat{x}^t)\|^2 = \sum_{x^t} \|D_c(x^t) - D_c(G(E(x^t)))\|^2 \tag{9}$$

3.6. Feature space of inception-v3 network

Inception-v3 network is a very deep convolutional neural network trained on the very large ImageNet data set and we have reason to believe that it has generalized well to the image domain [13]. Because of this we can hypothesize that it has learned to detect important high-level features in its layers and we can those as a preprocessor and use the vector of activations at a higher layer as an abstract representation.

Let $I_c(x^t)$ and $I_c(\hat{x}^t)$ denote the vector of activations of the Inception-v3 network at layer c for training instance x^t and its reconstruction $\hat{x}^t$. Then the reconstruction loss can be defined as (see Fig. 7):

$$\mathcal{L}_{FeaInc} = \sum_{x^t} \|I_c(x^t) - I_c(\hat{x}^t)\|^2 = \sum_{x^t} \|I_c(x^t) - I_c(G(E(x^t)))\|^2 \tag{10}$$

3.7. Related work

There are GAN variants in the literature that learn an inverse mapping and use this to improve training by defining extra constraints. The VAE-GAN [14] model combines GAN with a VAE and jointly trains GAN with an additional reconstruction loss defined by the VAE using an abstract representation obtained from D; this corresponds to a hint in the feature space of D.

There have since been a number of different versions of this VAE-GAN combination, such as the adversarial autoencoder (AAE) [15], introspective adversarial networks [16], α-GAN [17], adversarial variational Bayes (AVB) [18], and InjectionGAN [19] for image-to-image translation.

VEEGAN [20] learns an inverse mapping via a reconstructor network using a reconstruction loss proposed to stabilize the adversarial training. The boundary equilibrium GAN (BEGAN) [21] uses a discriminator architecture resembling an autoencoder; a reconstruction loss derived from the Wasserstein distance is used as a lower bound.

ALI with conditional entropy (ALICE) [22] also proposes adding a cycle-consistency loss term to the adversarial training of BiGAN to improve the reconstruction quality. CycleGAN [23] uses a BiGAN-like inverse mapping to map a source image to the target image instead of mapping from the data space to the latent space; this inverse mapping stabilizes training by introducing a cycle-consistency loss. DiscoGAN [24] and DualGAN [25] variants also propose additional reconstruction terms added to the adversarial training of the original GAN; these two models and ALICE use hints that are reconstruction losses defined in the original data (pixel) space.

Multi-Discriminator GAN (MDGAN) [26] uses two different discriminators; one is a GAN discriminator and the other one is an autoencoder that enforces a reconstruction error serving as an anomaly detector. High-quality bidirectional GAN (HQBiGAN) [27] is an extension of BiGAN that learns a mapping from the feature space extracted by D to the latent space, instead of the data space as used in the original BiGAN.

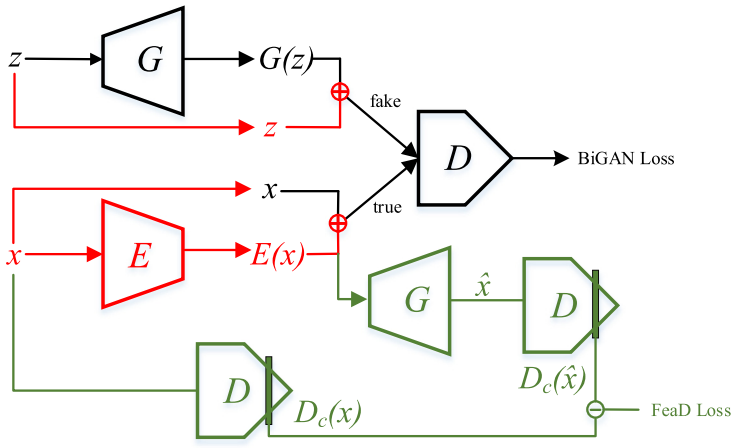


Fig. 6. We use the early layers of D as a preprocessor and compare the activations at some later layer c for the original input x and its reconstruction $\hat{x}^t \equiv G(E(x^t))$.

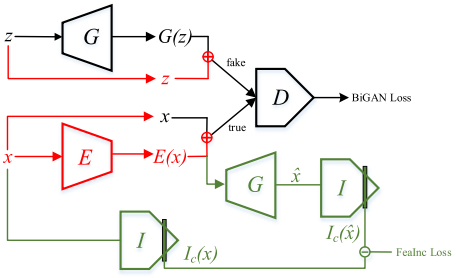


Fig. 7. We use the early layers of the Inception network trained on the very large ImageNet data as a preprocessor and compare the activations at some layer c for the original input x and its reconstruction $\hat{x}^t \equiv G(E(x^t))$.

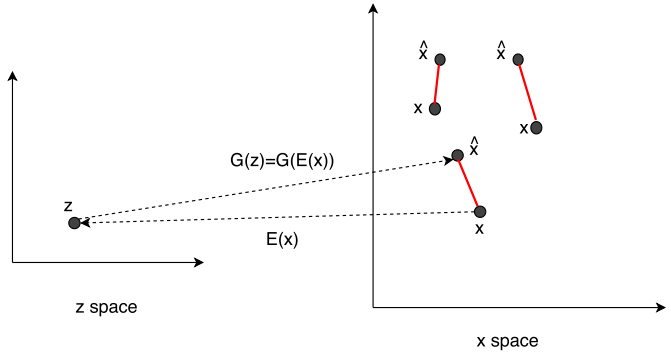


Fig. 8. Reconstruction error. For any x from the test set, $z \equiv E(x)$ is its latent representation and $\hat{x} \equiv G(z)$ is its reconstruction back in the input space. The difference $\|x - \hat{x}\|^2$ is the reconstruction error. If E and G are well-trained, this is expected to be small.

4. Experiments and results

4.1. Setting

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. MNIST consists of 60,000 handwritten grayscale digit images each of size 28 × 28, which we resize to 32 × 32 for convenience. UT-Zap50K data set contains 50,025 shoe images in RGB; images are of varying sizes and we resize them to 32 × 32. The training set of GTSRB contains 39,209 traffic sign images which we again resize to 32 × 32. Cifar10 contains 50,000 training images of various objects and scenes, also of size 32 × 32. CelebA contains celebrity face images, consisting of 202,599 samples; we use the aligned and cropped 64 × 64 version to better preserve details.

For the generator, the discriminator, and the added encoder for BiGAN, we use network architectures similar to the Deep Convolutional GAN (DCGAN) [7]. The encoder architecture is the exact inverse of the DCGAN generator. 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.

Following the original Wasserstein GAN [8], we do not have batch normalization or dropout in the discriminator of the Wasserstein BiGAN. All data sets except CelebA has 32 × 32 inputs; for CelebA, we add one additional convolutional/deconvolutional layer to map between 32 × 32 and 64 × 64. There is no pooling layer in any of the networks and resizing is done by downsampling convolutions or upsampling deconvolutions.

The latent z dimensionality value used for MNIST is 50. For UT-Zap50K, GTSRB, and Cifar10, which contain somewhat more detail, z dimensionality is set to 64, and for CelebA of face images that contain even more detail, z dimensionality is set to 100. In all experiments, the latent z are sampled from standard normal distribution; we used 0.0005 as the learning rate and the Adam optimizer. Training is done on a single Tesla V100 GPU.

4.2. Evaluation measures

To evaluate the quality of learned generators, we use three measures:

1. *Reconstruction error* (ERR): Using the implicit autoencoder composed of the encoder E and the generator G, for 1000 held-out test instances, we calculate the total reconstruction error in the original image space (see Fig. 8).
2. *Fréchet Inception distance* (FID) [28]: After each training epoch, using 1000 fake samples generated by G and 1000 true samples from the held-out test set, we pass the true and fake samples through the Inception-v3 network and retrieve the activations of an intermediate layer. We then fit multivariate Gaussians to these representations for true and fake samples separately and calculate the means and covariance matrices for true and fakes, namely, m_t and S_t for true

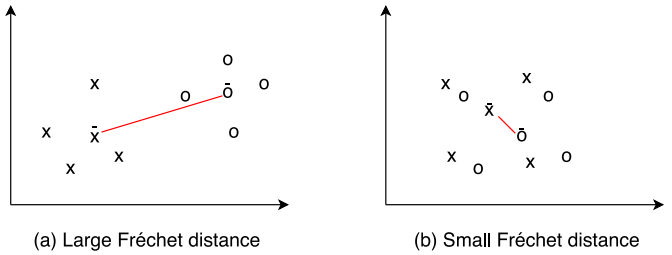


Fig. 9. Fréchet Inception distance FID). True instances, denoted by 'x' and generated fake instances, denoted by 'o', are mapped to an abstract space defined by an intermediate layer of the Inception network. FID is the distance between the two means, denoted by $\bar{x}$ and $\bar{o}$, taking also into account the covariance matrices. (a) If the generator is not good, the true and fake samples are separated and FID is large. (b) If the generator is good, then the true and fake samples overlap and FID is small.

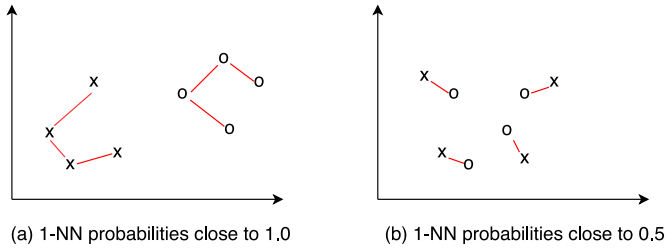


Fig. 10. 1-NN measure. In the original data space, we look for the nearest neighbor of true instances, denoted by 'x' and fake instances, denoted by 'o'. If the generator is not good, fake samples are far from true samples and the nearest neighbor of each sample is of the same type and the 1-NN probabilities will be close to 1.0. If the generator is good, the two type of samples overlap and the nearest neighbor of a sample can be of either type and the 1-NN probabilities will be closer to 0.5.

samples, and m_f and S_f for the fakes. FID is the distance between the two means taking the covariance matrices into account (see Fig. 9):

$$\text{FID} = \|m_t - m_f\|^2 + \text{Tr}(S_t + S_f - 2(S_t S_f)^{1/2}) \tag{11}$$

where Tr is the trace operator.

3. *Leave-one-out (LOO) 1-nearest neighbor (1-NN) test* [29,30]: Again using 1000 fake samples generated by the generator and 1000 true samples from a held-out test set, for each sample from this 2000 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. We calculate and report two accuracies *True* and *Fake*, respectively for true and fake instances. If the true and the generated fake distributions overlap, as we hope they do, we expect both accuracies to be around 0.5 (see Fig. 10). 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, this measure also evaluates the diversity of the generated images. FID makes the parametric assumption that the true and fake samples are approximately multivariate normal; 1-NN is a nonparametric measure and makes no such assumption.

In addition to these quantitative measures, we also use the trained generator to generate a number of new samples from each data set, which allows manual visual inspection.

4.3. The original vs. Wasserstein BiGAN

In our first set of experiments, we compare the original and our proposed Wasserstein BiGAN. In Fig. 11, we show the evolution of leave-one-out 1-NN accuracies for true and fake instances during training on all five data sets. On MNIST, we see that Wasserstein

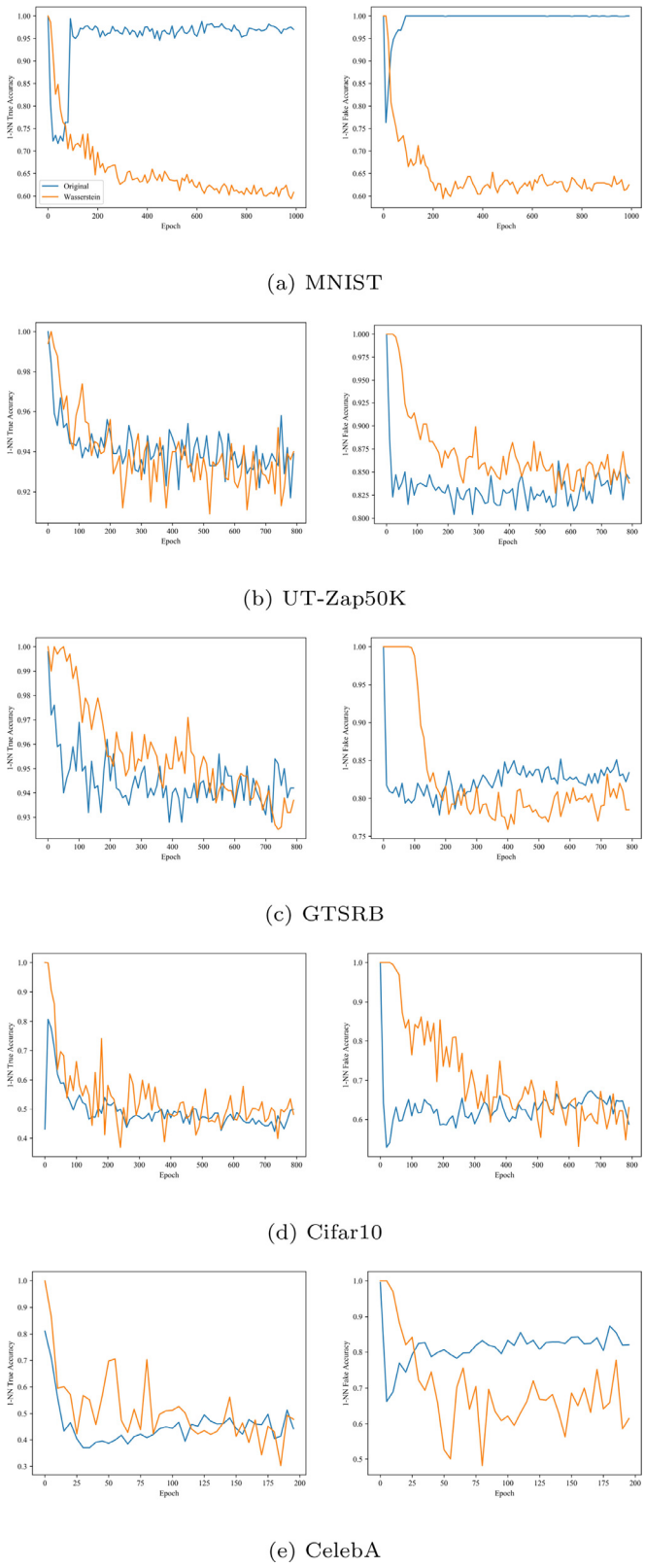


Fig. 11. The evolution of the leave-one-out 1-NN true (left) and fake (right) accuracies of the original and Wasserstein BiGANs during training.

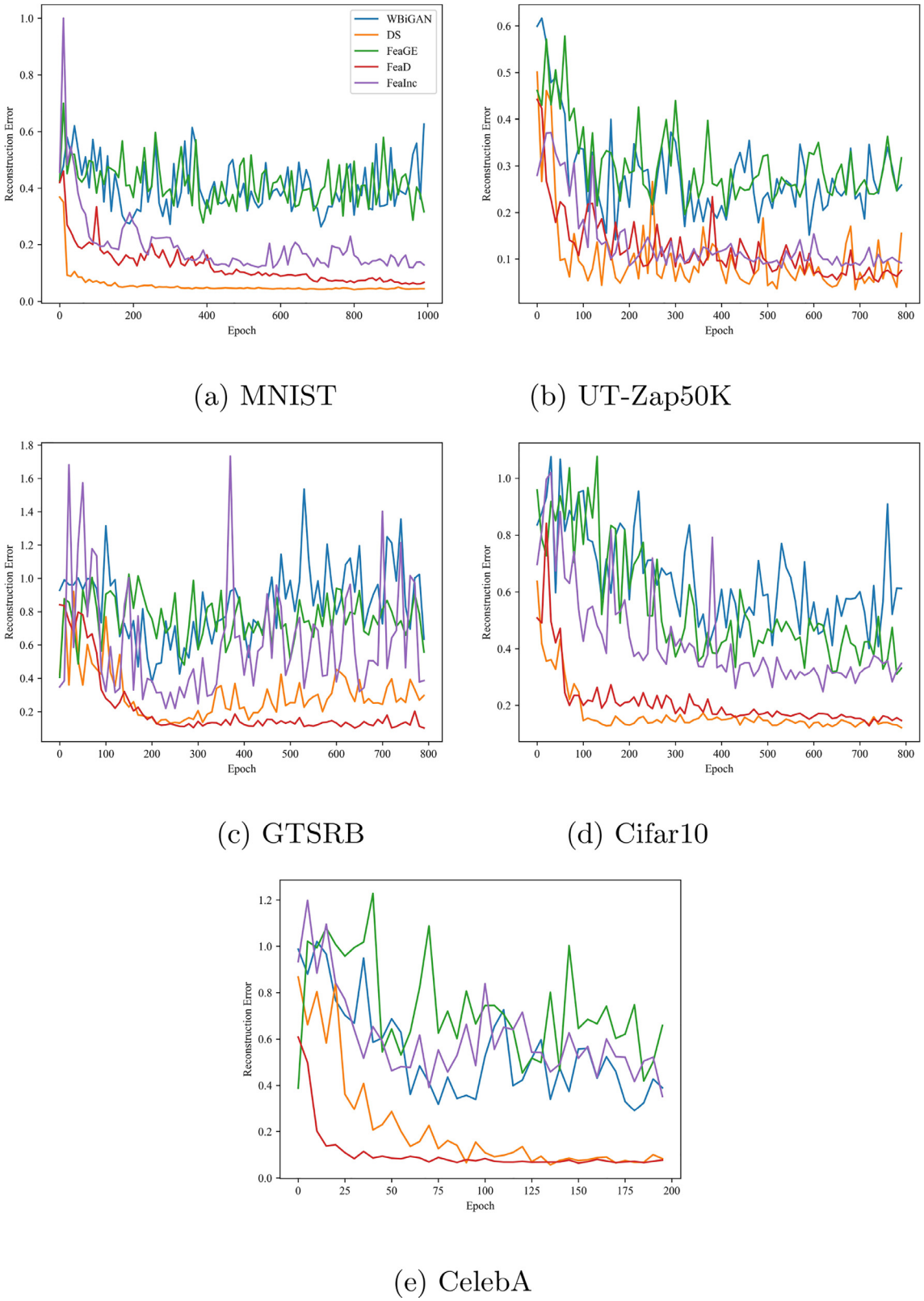


Fig. 12. Reconstruction errors of Wasserstein BiGAN models on held-out test instances.

BiGAN converges to almost 0.5, which is the best possible; this is not the case for the original BiGAN where both the accuracy of fakes and true instances go back up to 1.0, which is an indication that the fakes and true instances form separate groups by themselves. On UT-Zap50K, GTSRB, and Cifar10, the original BiGAN and Wasserstein BiGAN perform very similarly; because of the gradient penalty, the convergence of the Wasserstein BiGAN is slower but it eventually may get better performance. We also see this in the case of CelebA where Wasserstein BiGAN achieves better performance on the true instances. Overall, the Wasserstein BiGAN seems

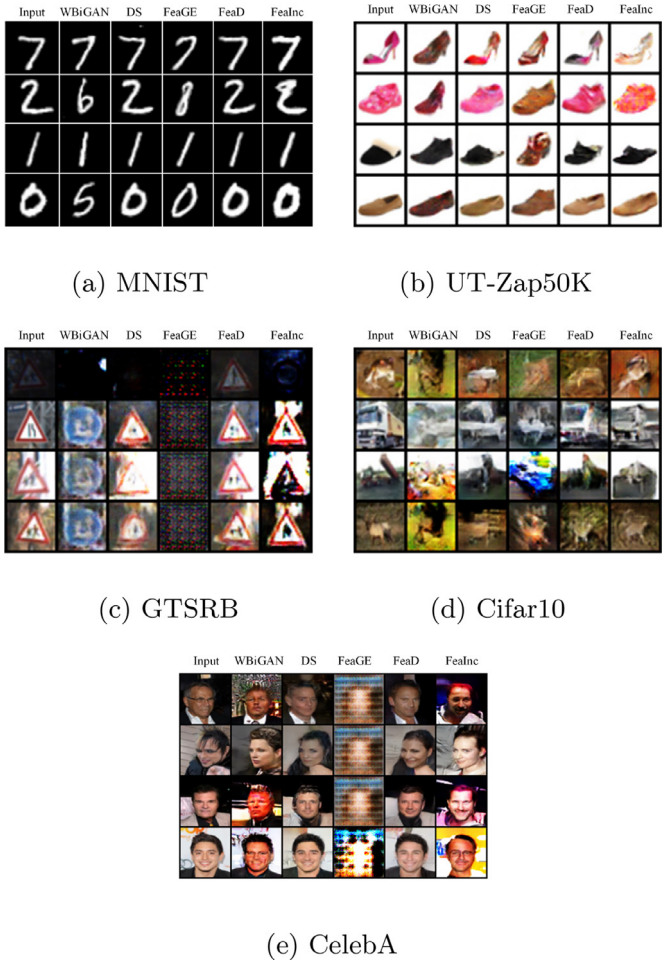


Fig. 13. Reconstruction examples. From left to right: The original test input and its reconstruction using Wasserstein BiGAN, WBiGAN-DS, WBiGAN-FeaGE, WBiGAN-FeaD, WBiGAN-FeaInc.

to achieve better performance than the original BiGAN and that is why we kept it as our basis BiGAN model in later experiments.

4.4. Comparison of the Wasserstein BiGAN variants

In our second set of experiments, we test the different Wasserstein BiGAN variants that use the different auxiliary losses we discussed in Section 3. Our results on the five data sets are summarized in Table 2, which shows the true and fake 1-NN accuracies, the Fréchet FID scores, and reconstruction errors for the original BiGAN, Wasserstein BiGAN and its four proposed variants with different hints. These are the best values on the held-out test set achieved during training; the best value in each row (for a data set and measure pair) is shown in bold.

On MNIST, we already saw that that the Wasserstein BiGAN, even without any hint, is better than the original BiGAN in terms of 1-NN accuracies; the variants with hints may lead to further improvement. The improvement due to the added hints is more striking in terms of the FID score where we see that all variants with hint are better than one with no hint. This is also true in terms of reconstruction error when FeaDS or FeaD is used.

We get similar results on UT-Zap50K and GTSRB where performance measured in terms of 1-NN fake accuracy, FID, or reconstruction error improves with added hints. On Cifar10 and CelebA, it seems as if all methods have comparable performance with a slight advantage of FeaDS when reconstruction error is the criterion.

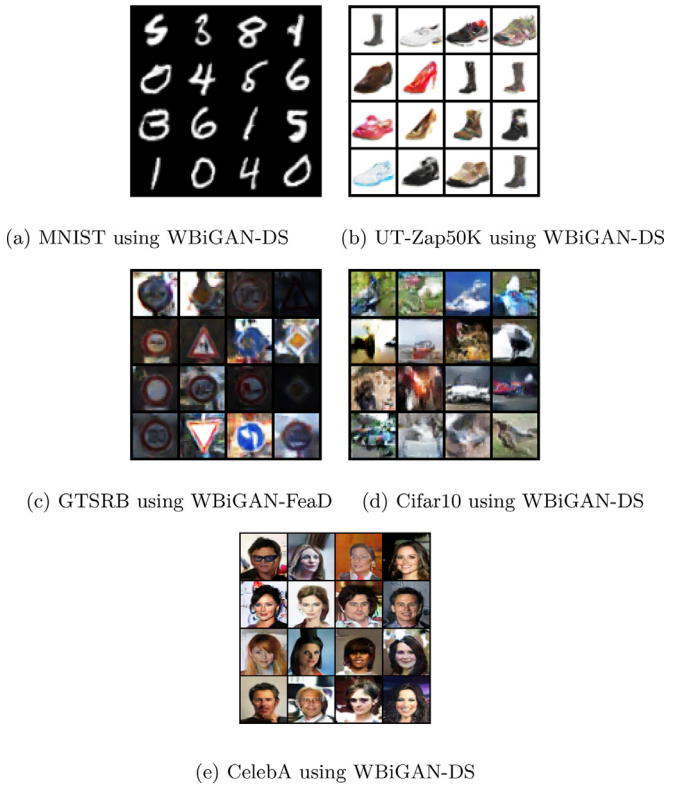
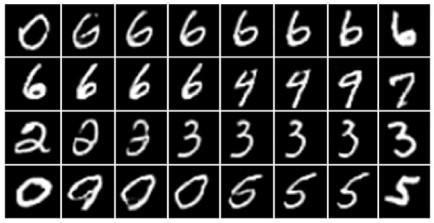


Fig. 14. Randomly generated images with the best performing Wasserstein BiGAN variant chosen in terms of reconstruction error.

Table 2
True and fake LOO 1-NN accuracies, Fréchet FID scores, and reconstruction errors of original BiGAN, Wasserstein BiGAN and its four proposed variants. These are best values achieved on the held out test set during training. The best score in each row (for a data set and measure pair) is shown in *bold*.

Data		BiGAN	WBiGAN	DS	FeaGE	FeaD	FeaInc
MNIST	True	0.717	0.594	**0.556**	0.618	0.585	0.597
	Fake	0.763	**0.594**	0.624	0.629	0.596	0.610
	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	True	0.917	**0.909**	0.912	0.920	0.918	0.931
	Fake	0.804	0.827	0.767	0.787	**0.750**	0.769
	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	True	0.928	0.925	0.878	0.902	0.907	**0.822**
	Fake	0.778	**0.759**	0.812	0.826	0.833	0.788
	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	True	**0.500**	**0.499**	**0.499**	**0.500**	**0.499**	**0.499**
	Fake	0.529	0.531	**0.496**	0.533	**0.496**	0.507
	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	True	**0.498**	**0.498**	0.497	**0.498**	0.504	**0.502**
	Fake	0.662	**0.501**	0.503	0.508	0.580	0.475
	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

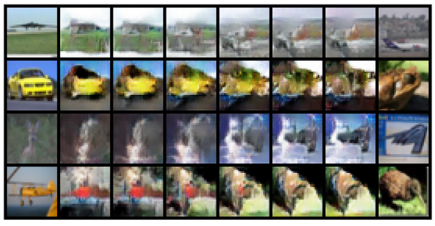
The evolution of the reconstruction errors on 2000 held-out test examples during training using all Wasserstein BiGAN variants on all five data sets are shown in Fig. 12. We see that the variants that use abstract features extracted by D (FeaD) or the original image pixels (DS) perform the best on all data sets, indicating that they lead to more faithful generators. The fact that the variant using D performs almost as well as DS shows that the discriminator network has learned informative representations in its intermediate layers. The variant that uses Inception network as a preprocessor (FeaInc) is competitive on UT-Zap50K and GTSRB; this probably is



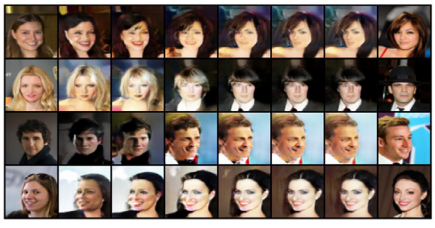
(a) MNIST using WBiGAN-DS

(b) UT-Zap50K using WBiGAN-DS

(c) GTSRB using WBiGAN-FeaD



(d) Cifar10 using WBiGAN-DS



(e) CelebA using WBiGAN-DS

Fig. 15. Interpolation examples using the best performing Wasserstein BiGAN variant for each data set in terms of reconstruction error. The first and last image in each row are randomly picked from the data; images in between them are interpolated in the latent space and are novel images synthesized by the trained generator.

an indication that such data are part of the ImageNet data set over which the Inception network is trained. The variant that uses the G and E in parallel (FeaGE) does not perform better than WBiGAN without any hint.

Some example reconstructions using these models are shown in Fig. 13 for visual inspection; these are reconstructions of held-out test instances not seen during training. Again we see the advantage of hints here: Wasserstein BiGAN without hint is not always good; the best models seem to be FeaDS and FeaD. That WBiGAN-FeaGE model does not perform well is also seen in these results.

4.5. Sampling from the Wasserstein BiGAN generators

Once training is complete, we can use the trained G to generate completely new instances, by sampling randomly from $\tilde{z} \sim p(z)$ and calculating $G(\tilde{z})$. We do this using G trained with different WBiGAN variants on the five data sets; on each data set, we choose the variant that is best in terms of the reconstruction error. Examples are shown in Fig. 14. We see that the trained generators are able to produce novel and realistic examples from the same distribution indicating that they have learned the main characteristics of the data; note that the image resolution on complex images, e.g., Cifar10 and CelebA, can always be improved by using deeper networks for G.

4.6. 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 cause changes in x. In the case of image data, these interpolations may result in semantically meaningful changes. With a trained BiGAN, we have the encoder E that maps an image to the latent space and we can interpolate in there as follows. We pick two training images x_1 and x_2 and pass them through E and get the corresponding z_1 and z_2. We then move from z_1 to z_2 in small steps in the latent space and use G to generate the corresponding x images on the way.

Some examples of this type of interpolation in the latent space using the BiGAN variants are shown in Fig. 15. These examples are generated using the best performing model for each data sets in terms of reconstruction error. In each row, the first and the last image are real examples from the training set; all images in between are synthesized novel and realistic images. These results also show that the interpolations slowly change some semantic attributes of the samples. On MNIST for example, as we interpolate from one digit to another, the thickness and the slant change gradually. On CelebA, the head orientation changes during interpolation.

5. Conclusions

We applied the Wasserstein loss to BiGAN and also extended it by using additional loss criteria. After our experiments on MNIST, UT-Zap50K, GTSRB, Cifar10 and CelebA data sets, we have reached the following findings:

- The autoencoder structure of BiGAN allows defining a reconstruction error which can be used to define different loss criteria as hints. We see that suitably defined hints lead to improved quality in generation.
- We find that the variant that works in the original image space (DS) and one that works in the feature space learned by the discriminator (FeaD) dramatically reduce the reconstruction error on all data sets with respect to the Wasserstein BiGAN without any hint.
- The success of WBiGAN-FeaD shows 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-FeaGE model, indicating that the hidden representations of intermediate layers of the generator and the encoder do not provide enough complementary information to improve the reconstruction quality.

- The trained model can be used to sample new instances as well as to sample in the latent space learned by the encoder. When the hint is chosen appropriately, we see that the model can synthesize novel and realistic images. We have also observed semantic changes in images when interpolating in the latent space.

There are other loss functions that have been recently proposed for GANs, such as least-squares GAN [31], boundary equilibrium GAN [21], and energy-based GAN [32]; testing their BiGAN versions together with the hints we propose is one possible future research direction.

Acknowledgements

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

References

[1] D. P. Kingma, M. Welling, Auto-encoding variational Bayes, arXiv:1312.6114 (2013).
[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Advances in Neural Information Processing Systems 27, 2014, pp. 2672–2680.
[3] Y. Hong, U. Hwang, J. Yoo, S. Yoon, How generative adversarial networks and their variants work: an overview, arXiv:1711.05914 v10 (2019).
[4] M. Arjovsky, L. Bottou, Towards principled methods for training generative adversarial networks, in: International Conference on Learning Representations, 2017.
[5] J. Donahue, P. Krähenbühl, T. Darrell, Adversarial feature learning, arXiv:1605.09782 (2016).
[6] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, A. Courville, Adversarially learned inference, arXiv:1606.00704 (2016).
[7] A. Radford, L. Metz, S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, arXiv:1511.06434 (2015).
[8] M. Arjovsky, S. Chintala, L. Bottou, Wasserstein generative adversarial networks, in: International Conference on Machine Learning, 2017, pp. 214–223.
[9] A. Atapour-Abarghouei, S. Akcay, G.P. de La Garanderie, T.P. Breckon, Generative adversarial framework for depth filling via Wasserstein metric, cosine transform and domain transfer, Pattern Recognit. 91 (2019) 232–244.
[10] T. Karras, T. Aila, S. Laine, J. Lehtinen, Progressive growing of GANs for improved quality, stability, and variation, arXiv:1710.10196 (2017).
[11] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A.C. Courville, Improved training of Wasserstein GANs, in: Advances in Neural Information Processing Systems 30, 2017, pp. 5767–5777.
[12] G.-J. Qi, Loss-sensitive generative adversarial networks on Lipschitz densities, arXiv:1701.06264 (2017).
[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.
[14] A.B.L. Larsen, S.K. Sønderby, H. Larochelle, O. Winther, Autoencoding beyond pixels using a learned similarity metric, arXiv:1512.09300 (2015).
[15] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, B. Frey, Adversarial autoencoders, arXiv:1511.05644 (2015).
[16] A. Brock, T. Lim, J.M. Ritchie, N. Weston, Neural photo editing with introspective adversarial networks, arXiv:1609.07093 (2016).
[17] M. Rosca, B. Lakshminarayanan, D. Warde-Farley, S. Mohamed, Variational approaches for auto-encoding generative adversarial networks, arXiv:1706.04987 (2017).
[18] L. Mescheder, S. Nowozin, A. Geiger, Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks, arXiv:1701.04722 (2017).
[19] W. Xu, K. Shawn, G. Wang, Toward learning a unified many-to-many mapping for diverse image translation, Pattern Recognit. 93 (2019) 570–580.
[20] A. Srivastava, L. Valkov, C. Russell, M.U. Gutmann, C. Sutton, VEEGAN: reducing mode collapse in GANs using implicit variational learning, in: Advances in Neural Information Processing Systems 30, 2017, pp. 3308–3318.
[21] D. Berthelot, T. Schumm, L. Metz, BEGAN: boundary equilibrium generative adversarial networks, arXiv:1703.10717 (2017).
[22] C. Li, H. Liu, C. Chen, Y. Pu, L. Chen, R. Henao, L. Carin, ALICE: towards understanding adversarial learning for joint distribution matching, in: Advances in Neural Information Processing Systems 30, 2017, pp. 5495–5503.
[23] J.-Y. Zhu, T. Park, P. Isola, A.A. Efros, Unpaired image-to-image translation using cycle-consistent adversarial networks, arXiv:1703.10593 (2017).
[24] T. Kim, M. Cha, H. Kim, J.K. Lee, J. Kim, Learning to discover cross-domain relations with generative adversarial networks, arXiv:1703.05192 (2017).
[25] Z. Yi, H.R. Zhang, P. Tan, M. Gong, DualGAN: unsupervised dual learning for image-to-image translation., in: International Conference on Computer Vision, 2017, pp. 2868–2876.
[26] Y. Intrator, G. Katz, A. Shabtai, MDGAN: boosting anomaly detection using multi-discriminator generative adversarial networks, arXiv:1810.05221 (2018).
[27] D. Bang, H. Shim, High quality bidirectional generative adversarial networks, arXiv:1805.10717 (2018).
[28] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, S. Hochreiter, GANs trained by a two time-scale update rule converge to a local Nash equilibrium, in: Advances in Neural Information Processing Systems 30, 2017, pp. 6626–6637.
[29] D. Lopez-Paz, M. Oquab, Revisiting classifier two-sample tests, arXiv:1610.06545 (2016).
[30] Q. Xu, G. Huang, Y. Yuan, C. Guo, Y. Sun, F. Wu, K. Weinberger, An empirical study on evaluation metrics of generative adversarial networks, arXiv:1806.07755 (2018).
[31] X. Mao, Q. Li, H. Xie, R.Y. Lau, Z. Wang, S.P. Smolley, Least squares generative adversarial networks, in: IEEE International Conference on Computer Vision, 2017, pp. 2813–2821.
[32] J. Zhao, M. Mathieu, Y. LeCun, Energy-based generative adversarial network, arXiv:1609.03126 (2016).

Uras Mutlu recevied his B.Sc. degree on computer engineering from Istanbul Technical University in 2016 and his M.S. degree on computer engineering from Boğaziçi University in 2019. He is currently a Ph.D. student with research interests such as generative adversarial networks, computer vision, natural language processing, and deep learning in general.

Ethem Alpaydın received his Ph.D. degree from Ecole Polytechnique Fédérale de Lausanne, Switzerland, in 1990, and was a postdoc at the International Computer Science Institute, Berkeley in 1991. He is a Professor in the Department of Computer Engineering, Boğaziçi University, Istanbul and a Member of the Science Academy, Istanbul. He was a Visiting Researcher at MIT in 1994, IDIAP, in 1998, and TU Delft, in 2014. He was a Fulbright scholar, in 1997. The third edition of his book Introduction to Machine Learning was published by The MIT Press, in 2014.