Generative Adversarial Autoencoder Network

Ngoc-Trung Tran, Tuan-Anh Bui, and Ngai-Man Cheung
Singapore University of Technology and Design (SUTD)
{ngoctrung_tran,tuananh_bui,ngaiman_cheung}@sutd.edu.sg

Abstract

In this paper, we introduce techniques to improve the original Generative Adversarial Networks. Our goal is to benefit the latent obtained from inference model to stabilize the training of the generation model. We achieve this goal by first introducing our basic model, a simple/effective way to use auto-encoders into original GAN framework with the help of gradient penalty. The gradient penalty was demonstrated by using with Wasserstein distances. We empirically explore that it can also stabilize our model by adding this term to discriminator objectives. Based the introduced model, we analyze some observation on encoded latent and we then propose the auto-encoders regularization terms, which is helpful to improve further GANs model. This term is to enforce the distances between two different spaces that direct the generators better avoiding the mode collapse and stabilize the convergence. Finally, we infer the interpolated latent variables from the latent and prior noise to improve further the convergence and stability of our method. We empirically demonstrate our model on synthetic, MNIST and CelebA datasets. Results show that our method can infer well multi-modal distribution and perform better than state-of-the-art methods on synthetic and MNIST datasets, and can generate many quality faces on CelebA dataset.

1. Introduction

Recently, Generative Adversarial Networks [12] (GANs) has been being a dominant approach for learning generative models as it can produce very visually appealing samples, but requires few assumption about the model. The core idea behind its huge success is the approach can produce samples *without* explicitly estimating data distribution P_{x}, *e.g.* in analytical forms. GANs uses the generator $G(\mathrm{z})$ to generate data samples with given random noise z sampled from a prior uniform distribution P_{z}. Simultaneously, it uses a discriminator D as an "arbitrator" to distinguish whether samples are drawn by the generator distribution P_G or data distribution P_{x}. Training GANs is an adversarial process that while the discriminator D learns to better discriminate real or fake samples, the generator G attempts to confuse the discriminator D into accepting its outputs as being real. Another point-of-view is that the discriminator is like a "teacher" instructing the generator how to improve. Despite their success, GANs is known to be hard to train, because it is difficult to reach the Nash equilibrium of the mini-max game [11] with current optimization algorithms. Therefore, the imbalance between discriminator and generator often leads to convergence issues. Gradient vanishing, or mode collapse are important aspects among of them. Gradient vanishing is when the discriminator provides no informative gradient for the generator to improve. It often occurs when the discriminator can distinguish so well between the real and "fake" samples before the generator distribution can reach the data distribution. Mode collapse is in which the parameter setting of the generator is collapsed that always generate low diversity of samples. In addition, generating meaningful/realistic outputs is still an open problem.

One class of GANs variants aims to address these problems are auto-encoder based GANs. Auto-encoders [14, 19, 7] is another prominent class of generative models, are attractive for learning inference model and generative model and can lead to better log-likelihoods [21]. Therefore, many recent works applied it successfully as an inference model to improve GANs training [10, 9, 16], or use auto-encoders to define the objective function of discriminators [22, 6]. There are also other attempts at combining auto-encoders and GANs [17, 15]. Those fusions are interesting; however, their the underlying principles remains unclear, *e.g.* what is the impact of auto-encoders? or can the model discourage the mode collapse problem?

In this work, we follow this fusion direction, but introduce a model explaining more clearly how auto-encoders can help to stabilize GANs training, *e.g.* avoiding the mode collapse problem, and approximating better data distribution. Our main contributions are two-fold: (i) We propose a new form of unifying auto-encoders and GANs with the help of gradient penalty (ii) We introduce and new regularization terms on auto-encoders to improve the data generation of the model and interpolate the latent code to achieve

slightly better results. (iii) Comparing to state of the art on synthetic and benchmark datasets, we achieve better stability, balance, and competitive standard scores.

2. Related Works

The issue of non-convergence still remains open for research direction, in which gradient vanishing, mode collapse is its most common forms [11, 3]. Many important variants of GAN have been proposed to address its limitations by improving GAN training. Improved GANs [20] introduced some techniques, *e.g*. feature matching, minibatch discrimination, and historical averaging, to drastically reduce the mode collapse. Unrolled GAN [18] tried to change optimization process to address convergence and mode collapse. WGAN [5] leveraged the Wasserstein distance, which provides convergence properties optimized by GAN. They require the discriminators must lie on the space of 1-Lipschitz functions, and enforce norm critics to the discriminator by weight-clipping trick. WGAN-WP [13] stabilize WGAN by alternating the weight-clipping by penalizing the norm of the gradient with respect to its input.

An alternative approach is to benefit the inference model or latent variables. AAE [17] learns the inference by matching the latent vector of autoencoder to the prior. There is no consideration on generator excepts the reconstruction constraint, which does not guarantee the generator can well approximate data distribution. VAE/GAN [15] combines VAE and GANs into one model and use feature-wise errors to capture better data distribution. InfoGAN [8] learns the disentangled representation by maximizing the mutual information for inducing latent codes. EBGAN [22] introduces the energy-based model, in which the discriminator is considered as energy function minimized via reconstruction errors. BEGAN [6] extends EBGAN by optimizing Wasserstein distance between auto-encoder loss distributions. ALI [10] and BiGANs [9] encode the data into latent and learn jointly the data/latent in GAN framework. This model can learn Autoencoders model via training although no explicit form of Autoencoders introduced.

3. Proposed method

We first propose our model (namely Generative Adversarial Autoencoder Networks, or GAANs) of unifying Autoencoders and GANs. Relying on observations on latent variables from this model, we propose some techniques to improve the model.

3.1. Generative Adversarial Autoencoder Networks

First, we introduce a simple way to integrate autoencoders into GAN framework. In our model, we consider the decoder of an auto-encoder as the generator. The goal is to direct the generator generating samples like real ones in training set. We use auto-encoder to derive the generator has two main benefits: (i) It can guide the generator the real targets, and (ii) It provides us more information (*e.g*. latent points), and more data (reconstructed samples), and based on that, we propose the regularization of Autoencoders to stabilize the convergence and reduce the mode collapse problem. The objective of our regularized Autoencoders can be written:

$$\min_{E,G} \mathcal{R}(E,G) + \lambda_{\mathrm{r}} \mathcal{W}(E,G) \tag{1}$$

where $\mathcal{R}(E,G) = ||\mathrm{x} - G(E(\mathrm{x}))||_2^2$ is the objective of traditional Autoencoders, $\mathcal{W}(E,G)$ is our proposed regularization will be discussed later and λ_{p} is regularization term. The Autoencoders consists of the encoder E and the decoder G, the input data x can be reconstructed by the Autoencoder: $G(E(\mathrm{x})) = \mathrm{x}+\varepsilon$, where ε is noise. Assume that the capacity of E and G are large enough so that the noise ϵ is small that means it's possible to assume those reconstructed samples as "real" samples (plus noise). However, in practice, using element-wise reconstruction are not good for high-quality images, can cause the blur issue. To avoid this, we use feature-wise distance [18] or similarly feature matching [20], hence $\mathcal{R}(E,G) = ||\Phi(\mathrm{x}) - \Phi(G(E(\mathrm{x})))||_2^2$ when we work on image data. In our implementation, $\Phi(\mathrm{x})$ is the feature output from an last convolution layer of discriminator D. Our framework is shown in Fig. 1. We propose to train encoder E, generator G and discriminator D following the order: (i) fixing D and train E and G to minimize the reconstruction Eqn. 1 (ii) fixing E, G, and train D to maximize (Eqn. 2), and (iii) fixing E, D and train G to maximize (Eqn. 2). $\mathrm{z_e} = E(\mathrm{x})$ is the latent code obtained from the encoder. The generator objective is exactly the same as GANs, but we modify the discriminator objective as discussed in the next section.

3.1.1 Discriminator objective

The objective function to train our discriminator is written in Eqn. 2. It is different from original discriminator of GANs at two main points. First, we indicate the reconstructed samples as "real", represented by the term $\mathbb{E}_{\mathrm{x}\sim P_{\mathrm{x}}} \log D(G(\mathrm{z_e}))$). By this way, we can push the boundary decision towards closer the generator distribution, since the distribution of reconstructed samples is nearly identical the distribution of generated samples at early epochs, then it is closer and closer to the data distribution as the encoder and generator become better. Or in other words, this way can slowdown systematically convergence of the discriminator in order to avoid partly the problem of vanishing gradient. Second, we apply the gradient penalty for the discriminator objective (Eqn. 2), where λ_{p} is penalty coefficient, and $\hat{\mathrm{x}} = \epsilon \mathrm{x} + (1-\epsilon)G(\mathrm{z})$, ϵ is a uniform random

number $\epsilon \in U[0, 1]$. This penalty was used to enforce Lipschitz constraint of Wasserstein-1 distance [13]. In this work, we explore the benefit of this term as it can provide the good gradient to direct the learning of the generator. So, we empirically found that it makes our model better as adding this term to our model. However, using this gradient penalty alone doesn't solve common issues of GAN, *e.g.* mode collapse. It's true when we conducted experiments on MNIST dataset. Therefore, we propose the new regularization term of Autoencoders $\mathcal{W}(E, G)$, using the gradient penalty and this new term in combination makes our model stable.

3.1.2 Autoencoders regularization

The regularization $\mathcal{W}(E, G)$ comes from our observation of latent variables created by our model. We run our GAANs model (with gradient penalty, without auto-encoders regularization $\lambda_{\mathrm{r}} = 0$) on MNIST dataset and look the encoded latent variables. Fig. 2a shows 10K latent codes of standard MNIST digits on 2D space $[-1, 1]$ (the prior distribution of our model is uniform) by obtained from the encoder. The most left figure shows that latent code of our model (without auto-encoders regularization) does not fill out entirely the latent space, and its distribution doesn't look like the uniform (the prior distribution). We see the regions covered by MNIST digits are unequal in this latent space, and have serious overlapping between some digits. The empty regions are places probably cannot be well-handled by the auto-encoders, and those regions of latent codes can still cause partly mode collapse. For example, if the random noise z falls into empty regions on the right of digits '1' (blue points), it can likely generate images looks like '1' (with some noise), and as a result, the model can generate more digits '1' than others. Intuitively, in order to avoid the mode collapse and achieve the mode balance, the auto-encoders has to handle entire latent space, any region that random noise z can fall into. In other words, the latent variables need to be spread out the entire space and occupy similar areas in this space, so that the probability that random noise z fall in these digit regions is equal to generate the equivalent number of digits.

We know that the mode collapse occurs when the generator creates the low diversity of samples in data space given different noise inputs. In other words, those samples are "close" in the data space, even though the random noise may be "far" in its space (latent). Therefore, by above observation, we introduce a new regularization $\mathcal{W}(E, G)$ that direct the encoder and generator generate samples not suffering the mode collapse, and also achieve the mode balance. Through this term, we expect if the generated samples that are close each other in data space, their input noise/latent inputs must be relatively close each other in the latent space, and vice versa. We describe our words in the following formula:

$$\mathcal{W}(E, G) = ||f(G(\mathrm{z_e}), G(\mathrm{z})) - g(\mathrm{z_e}, \mathrm{z})||_2^2 \quad (3)$$

where f and g are distance functions computed in data space and latent space. In practice, rather than using the direct distance functions, *e.g.* Euclidean, L1 norm, etc, we measure the distribution similarity in data and latent spaces. We apply the Wasserstein-1 distance [3] for both f and define the similar function for g:

$$f(G(\mathrm{z_e}), G(\mathrm{z})) = \mathbb{E}_{\mathrm{x}\sim P_{\mathrm{x}}} D(G(\mathrm{z_e})) - \mathbb{E}_{\mathrm{z}\sim P_{\mathrm{z}}} D(G(\mathrm{z})) \quad (4)$$

$$g(\mathrm{z_e}, \mathrm{z}) = \mathrm{mean}(\mathbb{E}_{\mathrm{x}\sim P_{\mathrm{x}}} \mathrm{z_e} - \mathbb{E}_{\mathrm{z}\sim P_{\mathrm{z}}} \mathrm{z}) \quad (5)$$

where $\mathrm{mean}(.)$ computes the average of all dimensions of latent variables. Intuitively, we can see that if the generator is collapsed, $D(G(\mathrm{z}))$ gets the same output for any given z that leads $f(G(\mathrm{z_e}), G(\mathrm{z}))$ to zero, otherwise $g(\mathrm{z_e}, \mathrm{z})$ has different values rather than zeros. Therefore, enforcing f and g is close can reduce drastically the mode collapse problem. Fig. 3 shows the mapping directions between latent, data and discriminator spaces. The term $\mathcal{W}(E, G)$ enforce mapping distance between latent space and the discriminator space, the reconstruction handles the relationship between latent and data space. Implicitly, the discriminator penalty tends to avoid so high density of generated samples in data space when minimizing the logarithm difference between the data distribution and generator's distribution.

3.1.3 Latent interpolation

To improve further the convergence of our method, we propose to interpolate more latent variables during training. We create more interpolated latent variables $\mathrm{z_i}$ based on the random noise z and latent code $\mathrm{z_e}$. Using these latent variables to generate more "fake" samples to fool the discriminator D:

$$G(\mathrm{z_i}) = G((1 - \alpha)\mathrm{z_e} + \alpha \mathrm{z}) \quad (6)$$

Therefore, to train the generator G for each step, we need to maximize the objective function:

$$\max_G \mathbb{E}_{\mathrm{z}\sim P_{\mathrm{z}}} \log(D(G(\mathrm{z}))) + \mathbb{E}_{\mathrm{z_i}} \log(D(G(\mathrm{z_i}))) \quad (7)$$

Similarly, we also add $\mathbb{E}_{\mathrm{z_i}} \log(1 - D_{\mathrm{x}}(G(\mathrm{z_i})))$ to the discriminator objective. Using interpolated latent slightly accelerate the convergence of our model. The regularization term (Eqn. 3) enforces the latent variables have zero-center. Once we infer the latent variables from encoded latent and the prior, $\mathrm{z}_i = (1 - \alpha)\mathrm{z_e} + \alpha \mathrm{z}$ (*e.g.* $\alpha = 0.5$), we generate more data systematically around the encoded latent

$$\mathcal{L}(E,G,D) = \mathbb{E}_{\mathrm{x}\sim P_{\mathrm{x}}} \log D(\mathrm{x}) + \mathbb{E}_{\mathrm{x}\sim P_{\mathrm{x}}} \log D(G(\mathrm{z}_{\mathrm{e}}))) + \mathbb{E}_{\mathrm{z}\sim P_{\mathrm{z}}} \log(1 - D(G(\mathrm{z}))) + \lambda_{\mathrm{p}} \mathbb{E}_{\hat{\mathrm{x}}\sim P_{\hat{\mathrm{x}}}} (||\nabla_{\hat{\mathrm{x}}} D(\hat{\mathrm{x}})||_2 - 1)^2 \quad (2)$$

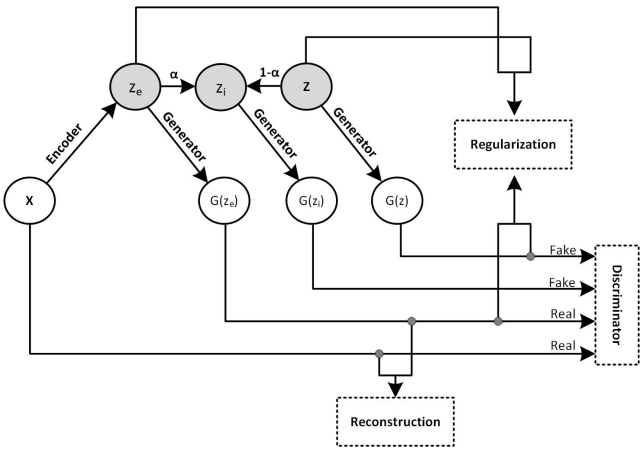

Figure 1. Our model architecture.

that during training can guide the generated samples closer to the real data (reconstructed from encoded latent). However, as fixing the α, the interpolated latent can be denser around the center of latent space, and it likely causes some imbalance between modes (Fig. 2c). Empirically, we find that interpolated data is helpful to direct the generator distribution to data distribution at early epochs of training. To avoid the imbalance problem at the end, we decay α for each iteration to reduce effects of latent code at later stages, for instance, $\alpha = \alpha * \nu^{\mathrm{epoch}}$ (Fig. 2d). The objective of our method is defined as the minimax objective of multiple players $\mathcal{J}(D,E,G)$ as in Eqn. 8. The diagram of our network architecture is illustrated in the Fig. 1 and the detailed algorithm is presented in Alg. 1.

$$\min_{E,G} \max_{D} \mathcal{J}(E,G,D) = \mathcal{L}(E,G,D) + \mathbb{E}_{\mathrm{z_i}} \log(1 - D(G(\mathrm{z_i}))) \quad (8)$$

4. Experimental Results

In this section, we demonstrate our method that can improve the mode coverage on our synthetic datasets and the MNIST datasets and can generate quality faces on CelebA dataset. All experiments are for the unsupervised tasks.

4.1. Synthetic data

The purpose of experiments on synthetic data is the see how well the generator can approximate the data distribution. For that, we create our synthetic dataset has 25 Gaussian modes in grid layout similar to [10], Fig 5. Our dataset contains 50K training points in 2D, and we draw 2K generated samples for testing. For fair comparison on this dataset,

	d_{in}	d_{out}	N_{h}	d_{h}
Encoder (E)	2	2	3	128
Generator (G)	2	2	3	128
Discriminator (D)	2	1	3	128

Table 1. The network structures are used for evaluating on synthetic data.

we use the equivalent architectures and setup all methods in the same condition if possible, similar architecture and network size like proposed in [18] on the 8-Gaussian dataset, but here we use one more hidden layer. We use only fully-connected layers for both generator and discriminator, and use Rectifier Linear Unit (ReLU) activation for input and hidden layers, sigmoid for output layers. The network structure of encoder, generator and discriminator are presented in Table 1, where d_{in}, d_{out}, d_{h} are dimension of input, output and hidden layers respectively. N_{h} is the number of hidden layers. The output dimension of the encoder is the dimension of the latent variable. Our prior noise is uniform distribution of $[-1, 1]$. We use Adam optimizer with learning rate $\mathrm{lr} = 0.001$, and the exponent decay rate of first moment $\beta_1 = 0.8$. The learning rate is decayed very $10K$ steps with a base of 0.9. The mini-batch size is 128. The training stops after 500 epochs. To be fair, we carefully fine-tune other methods (and use weight decay during training if it gets better results) to ensure they achieve their as best as possible results on the synthetic data. For evaluation, a mode is collapsed if there are less than 20 generated samples registered into this mode, which is measured by its mean and variance of 0.01 [16, 18]. For all experiments, we fix the parameters $\lambda_{\mathrm{r}} = 0.1$ (Eqn. 1), $\lambda_{\mathrm{p}} = 0.1$ (Eqn. 2), and initial $\alpha = 0.5$ (Eqn. 6) with decay rate $\nu = 0.98$. We conduct eight runs for one method and the final results for comparison are their average.

At first, we highlight the capability of our model to approximate the distribution P_{x} through experiments on our synthetic dataset. We evaluate our proposed method in different settings to understand the influence of each proposed component to the overall performance of our method. We define Setting 1 of our method (GAAN_1) of using the reconstruction term of autoencoders and the gradient penalty. Setting 2 (GAAN_2) improves from GAAN_1 by adding the autoencoders regularization. Setting 3 (GAAN_3) improves the GAAN_2 by interpolating fake latent for the generator to fool discriminator D. The quantitative results are shown in the first row of Fig. 4. The left figures are the number registered modes changing in the training. GAAN_3 cover all 25 modes as fast as GAAN_2, and GAAN_2 is slightly faster

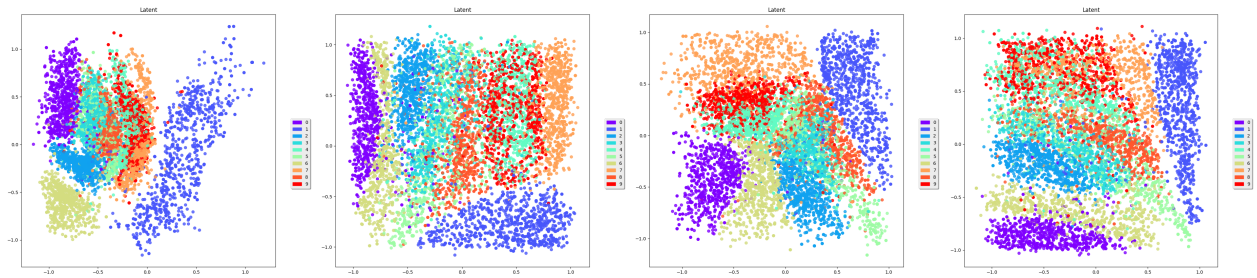

Figure 2. The 2D latent code obtained from 10K test images of MNIST by our method. To determine which one is better, we compute KL-divergence between latent distribution and prior distribution because we expect our latent distribution is similar to the prior distribution. From left to right (a) Setting 1, KL = 1.03 (b) Setting 2, KL = 0.33, (c) Setting 3 (fixed α), KL = 0.46 (d) Setting 3 (decrease α), KL = 0.21. (These settings are defined in the experiment section.)

Algorithm 1 Generative Adversarial Autoencoder Network

Initialize discriminators, encoder and generator D, E, G
repeat
  $\mathrm{x}^{\mathrm{m}} \leftarrow$ Random minibatch of m data points from dataset.
  $\mathrm{z}^{\mathrm{m}} \leftarrow$ Random m samples from noise distribution P_{z}
  // Learn encoder and generator on x^{m} *and* z^{m} *by Eqn. 1*
  $E, G \leftarrow \min_{E,G} \mathcal{R}(E, G) + \lambda_{\mathrm{r}} \mathcal{W}(E, G)$
  // Interpolate latent variables by Eqn. 6
  $\mathrm{z}_{\mathrm{i}}^{\mathrm{m}} \leftarrow (1-\alpha)\mathrm{E}(\mathrm{x}^{\mathrm{m}}) + \alpha \mathrm{z}^{\mathrm{m}}$ *// where* $\alpha \in [0, 1]$
  // Learn discriminators of data and latent by Eqn. 8 on $\mathrm{x}^{\mathrm{m}}, \mathrm{z}^{\mathrm{m}}, \mathrm{z}_{\mathrm{i}}^{\mathrm{m}}$
  $D \leftarrow \max_D \mathcal{J}(E, G, D)$
  // Learn once more the generator by Eqn. 7
  $G \leftarrow \max_G \mathbb{E}_{\mathrm{z} \sim P_{\mathrm{z}}} \log(D(G(\mathrm{z}^{\mathrm{m}}))) + \mathbb{E}_{\mathrm{z}_{\mathrm{i}}} \log(D(G(\mathrm{z}_{\mathrm{i}}^{\mathrm{m}})))$
until
return E, G, D

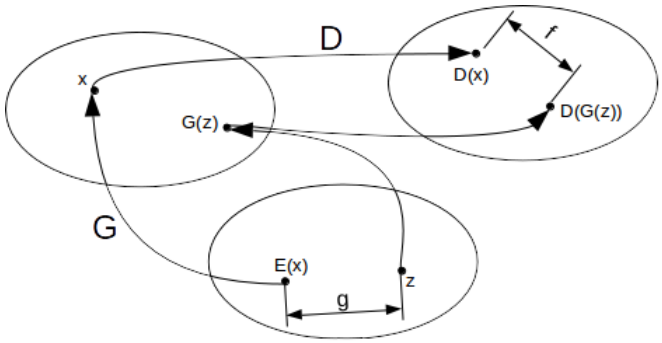

Figure 3. Relationship between spaces: data, latent and discriminators.

than GAAN_1. All three settings can cover all the modes after about 100 epochs. Because they don't miss the mode, it's fair to compare also on the number of registered points as on the right figures. GAAN_3 achieves the best number of registered points, slightly better than GAAN_2. GAAN_2 demonstrates the effectiveness of using auto-encoder constraint over the basic model of GAAN_1.

We compare our method to previous works, where ALI [10], and DAN-2S [16] are recent works using auto-encoder in their method, VAE-GAN [15] introduce a similar model, and WGAN-GP [13] is one of the current state of the art of GAN. The number of covered modes and registered points is presented in Fig 4, and the quantitative numbers are in Table 2. In this table, we report also Total Variation scores to measure the mode balance. All numbers are the average of eight runs for each method. Our method outperforms GAN [12], DAN-2S [16], ALI [10], and VAE/GAN [15] on both the number of registered samples and covered modes. While WGAN-GP sometimes misses one mode, we show our stability as not suffering any mode collapse on all eight runs, furthermore we achieve a higher number of registered samples than WGAN-GP and all others. Our method is also better than all others in Total Variation (TV) [16]. Fig. 5 visualizes the samples generated by the generators and other compared methods. Fig. 7 gives the detail of how many samples (in ratio) of 25 modes.

Interestingly, we try to apply our auto-encoders regularization and the latent interpolation to other models, such as ALI and VAE/GAN. We see that the use of two terms can improve these models, Fig. 6, where "-Reg" and "-Int" indicates the method use auto-encoder regularization and in-

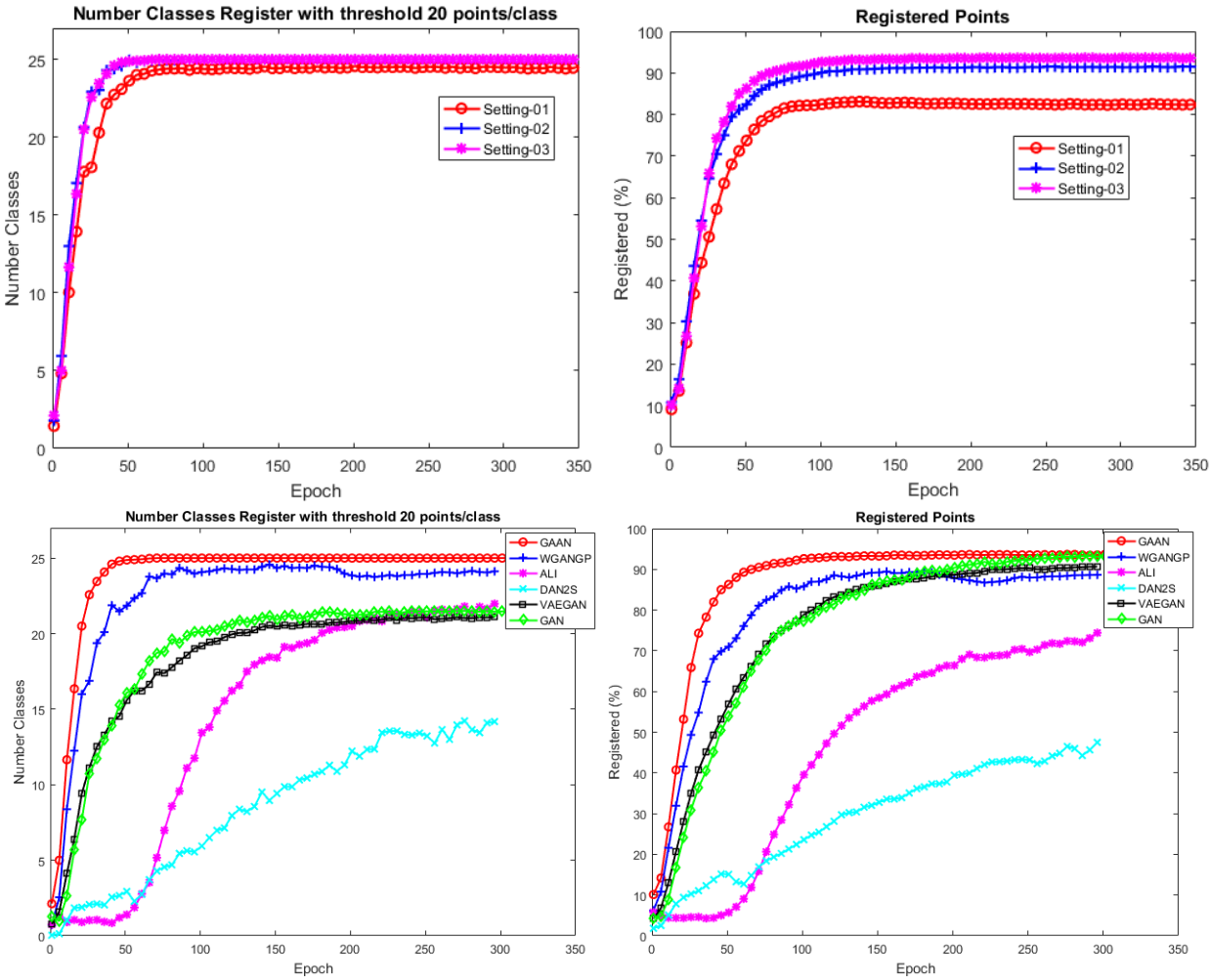

Figure 4. Comparison the number of registered modes (left) and points (right) on the synthetic dataset. The first row indicates the influence of each of our components. The second row compares to other methods.

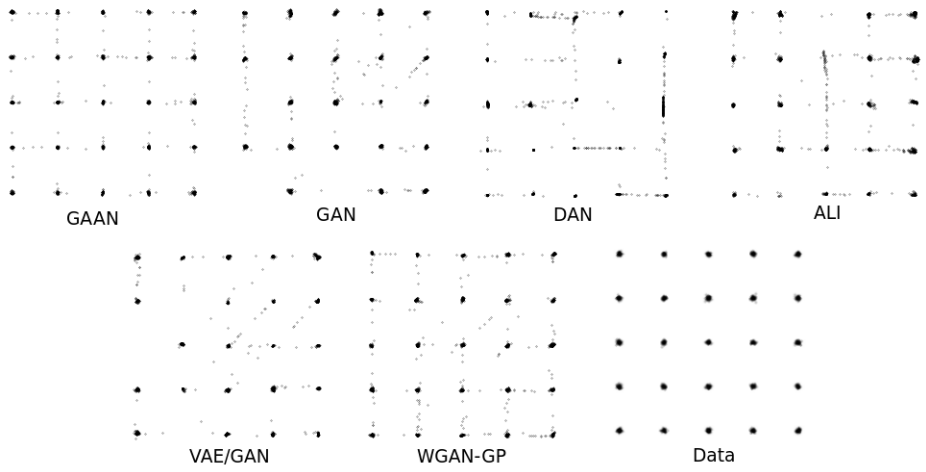

Figure 5. Visualizing samples generated by compared methods on the synthetic dataset.

Method	#registered modes	#registered points	TV (True)	TV (Differential)
GAN [12]	21.50 ± 2.00	1875.63 ± 42.53	1.00 ± 0.00	0.94 ± 0.02
DAN-2S [16]	14.07 ± 2.85	986.33 ± 160.80	0.99 ± 0.02	0.70 ± 0.09
VAE-GAN [15]	21.13 ± 2.33	1816.19 ± 61.23	0.98 ± 0.15	0.93 ± 0.21
ALI [10]	22.00 ± 2.63	1490.25 ± 194.38	0.93 ± 0.11	0.68 ± 0.15
WGAN-GP [13]	24.04 ± 1.16	1766.25 ± 79.48	0.54 ± 0.35	0.43 ± 0.31
$GAAN_1$	24.52 ± 1.00	1652.89 ± 148.22	0.47 ± 0.26	0.31 ± 0.19
$GAAN_2$	25.00 ± 0.00	1827.38 ± 25.60	0.20 ± 0.03	0.12 ± 0.02
$GAAN_3$	25.00 ± 0.00	1865.63 ± 23.75	0.15 ± 0.02	0.09 ± 0.01

Table 2. The network structures are used for evaluating synthetic data. Columns indicate the number of covered modes, and the number of registered samples among 2000 generated samples, and two types of Total Variation (TV).

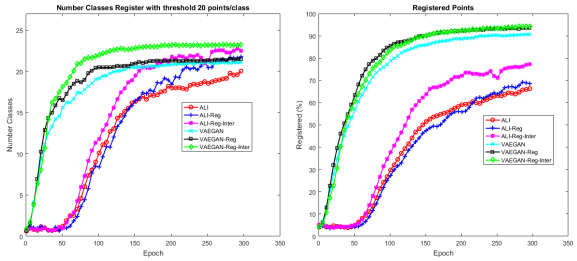

Figure 6. Improving other models by our regularization terms. Left images are a number of modes/classes registered during training, and the number of registered points is on the right. Using our terms both VAE/GAN and ALI can improve their performance.

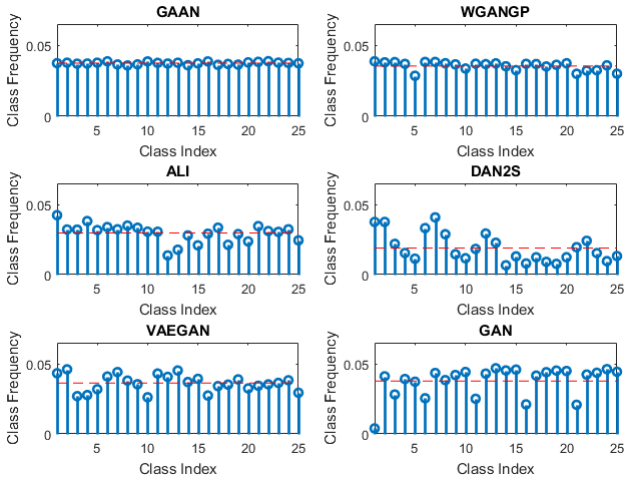

Figure 7. The mode balance obtained by compared methods.

terpolation respectively. This demonstrates the potential of our proposed regularization, not only for our model but also useful to improve other auto-encoders based GAN methods.

4.2. MNIST-1K

Our implementation on MNIST based on the open source published by WGAN-GP [13]. Fig. 8 is the generated samples and mode frequency of each digit generated by our method on standard MNIST. It confirms that our method can approximate well the MNIST digit distribution. The standard MNIST seems trivial to compare methods on mode collapse. Hence, we follow a more challenging technique [18] to construct 1000-class MNIST (MNIST-1K) dataset. This dataset is built by stacking three random digits to form an RGB image (one digit per a channel). It can be assumed to have 1000 modes from 000 to 999. The digits are classified by a pre-train classifier (0.4% in our case). We create a total of 25,600 images and measure methods by counting the number of modes is covered (at least one sample). We also compute KL divergence. To be fair, we adopt the equivalent network architecture (low-capacity generator and two crippled discriminators K/4 and K/2) as proposed by [18]. Table 3 presents the mode number and KL divergence of our method comparing to other published methods. Compare to others on MNIST-1K, our method outperforms all others at a number of covered modes, especially with low-capacity discriminator (K/4 architecture), that is higher about 150 modes than the next one. Our method achieves the small gap between two architectures (*e.g.* differ about only ~ 60 mode), and this gap is small as compared with other works. Our method achieves similar KL divergence as the state-of-the-art method, WGAN-GP. All results support the demonstration of our capability to handle a large number of modes and the stability of our training even in case of the imbalance between generator and discriminator.

5. CelebA dataset

We extend experiments of our GAANs on CelebA dataset and compare to DCGAN and WGAN-GP. Our implementation is based on the open source [1, 2], which provided some available results of DCGAN and WGAN-GP. Fig. 9 shows some samples generated by our GAAN, DCGAN, and WGANGP. While DCGAN encounters a slight mode collapse at epoch 50. Our method doesn't suffer this problem and generates realistic samples. While WGAN-GP sometimes generates broken faces (not look like human faces), all of our generated faces are recognizable and many of them looks realistic.

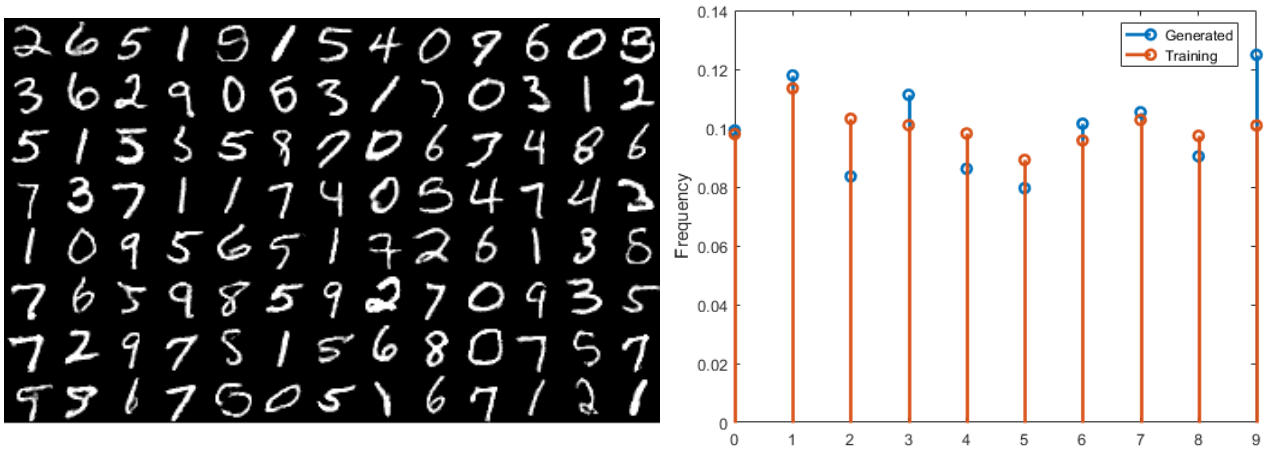

Figure 8. Our qualitative results and the number of generated samples of each class of our method on the standard MNIST dataset. We compare our frequency of generated samples to the ground-truth via KL divergence: KL = 0.01.

Architecture	GAN	Unrolled GAN	WGAN-GP	GAAN
K/4, #	30.6 ± 20.7	372.2 ± 20.7	640.1 ± 136.3	859.5 ± 68.7
K/4, KL	5.99 ± 0.04	4.66 ± 0.46	1.97 ± 0.70	1.04 ± 0.29
K/2, #	628.0 ± 140.9	817.4 ± 39.9	772.4 ± 146.5	917.9 ± 69.6
K/2, KL	2.58 ± 0.75	1.43 ± 0.12	1.35 ± 0.55	1.06 ± 0.23

Table 3. The comparison on MNIST-1K of methods. We follow the setup and network architectures from Unrolled GAN.

6. Conclusion

In this paper, we propose a GAN method by leveraging the autoencoders and its latent variables. First, we introduce the model can simply/effectively integrating autoencoders into GAN with the support of gradient penalty. Then, we introduce to use the regularization of auto-encoders to stabilize the model and achieve better mode balance. In addition, we propose to interpolate more latent during training can give slightly better results. Our proposed complete method does not suffer mode collapse issues as evaluating on both the synthetic, MNIST-1K and CelebA datasets. Furthermore, we achieve better results the previous works on MNIST-10K datasets. We demonstrate the efficiency of using good regularizations and gradient penalty can direct better the generator of GAN to produces better samples. Throughout experiments, we show that our method can approximate well the multi-modal distribution of training data on synthetic and MNIST-1K. We also show the potential of our regularization terms once applying it leads some improvements in other methods.

References

[1] `https://github.com/carpedm20/DCGAN-tensorflow`.

[2] `https://github.com/LynnHo/DCGAN-LSGAN-WGAN-WGAN-GP-Tensorflow`.

[3] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[4] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[5] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[6] D. Berthelot, T. Schumm, and L. Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[7] Y. Burda, R. Grosse, and R. Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

[8] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.

[9] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

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

[11] I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NISP*, pages 2672–2680, 2014.

[13] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

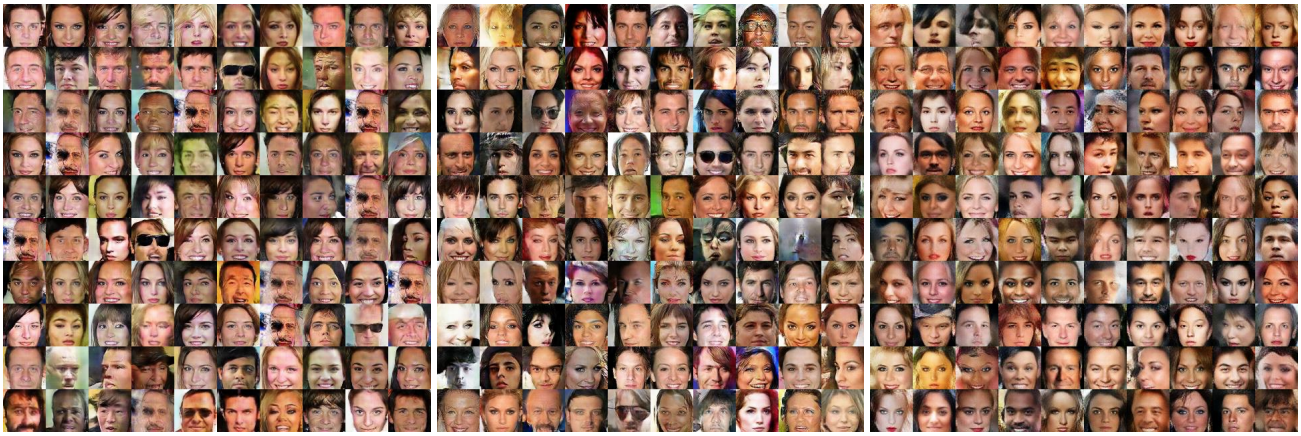

Figure 9. Generated samples of DCGAN (50 epochs, courtesy of [2]), WGAN-GP (50 epochs, courtesy of [2]) and our GAAN (50 epochs).

[14] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

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

[16] C. Li, D. Alvarez-Melis, K. Xu, S. Jegelka, and S. Sra. Distributional adversarial networks. *arXiv preprint arXiv:1706.09549*, 2017.

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

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

[19] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, pages 1278–1286, 2014.

[20] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *NISP*, pages 2234–2242, 2016.

[21] Y. Wu, Y. Burda, R. Salakhutdinov, and R. Grosse. On the quantitative analysis of decoder-based generative models. *arXiv preprint arXiv:1611.04273*, 2016.

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