PMGAN: Paralleled Mix-Generator Generative Adversarial Networks with Balance Control

Xia Xiao[✉] and Sanguthevar Rajasekaran

Computer Science and Engineering Department, University of Connecticut, Storrs, CT 06269, USA
{xia.xiao, sanguthevar.rajasekaran}@uconn.edu

Abstract. A Generative Adversarial Network (GAN) is an unsupervised generative framework to generate a sample distribution that is identical to the data distribution. Recently, mix strategy multi-generator/discriminator GANs have been shown to outperform single pair GANs. However, the mixed model suffers from the problem of linearly growing training time. Also, imbalanced training among generators makes it difficult to parallelize. In this paper, we propose a balanced mix-generator GAN that works in parallel by mixing multiple disjoint generators to approximate the real distribution. The weights of the discriminator and the classifier are controlled by a balance strategy. We also present an efficient loss function, to force each generator to embrace few modes with a high probability. Our model is naturally adaptive to large parallel computation frameworks. Each generator can be trained on multiple GPUs asynchronously. We have performed extensive experiments on synthetic datasets, MNIST1000, CIFAR-10, and ImageNet. The results establish that our model can achieve the state-of-the-art performance (in terms of the modes coverage and the inception score), with significantly reduced training time. We also show that the missing mode problem can be relieved with a growing number of generators.

Keywords: Deep learning · Generative adversarial networks
Parallelization

1 Introduction

Generative Adversarial Networks were proposed by [8], where two neural networks, the generator and the discriminator, are trained to play a minimax game. The generator is trained to fool the discriminator while the discriminator is trained to distinguish fake data from real data. When Nash Equilibrium is reached, the generated distribution P_G will be identical to the real distribution P_{real}. Unlike Restricted Boltzmann Machine or Variational Auto-encoder that

This work has been supported in part by the NSF grants 1447711 and 1743418.

explicitly approximate data distribution, the approximation of GAN is implicit [7]. Training a GAN is challenging due to various potential problems such as gradient vanish [1], missing mode [5,10,11,15,16], mode collapse [2,7], equilibrium [3,4], etc.

Recently, the authors of [3,9] have used a set of generators to replace a single complex generator. Each generator only captures a part of the real distribution. In this case, the distance between the mix-generated distribution and the real distribution should be minimized. A new classifier is added to separate each pair of generators. The generated image using this approach obtained the highest score (Inception Score of about 15% better than the average of the second and the third competitors). Note that the overlapping penalty from the classifier and an unrealistic penalty from the discriminator may conflict during training. More specifically, we observe two problems in practice: (1) competition: multiple generators try to capture one mode, but are hampered by a strict boundary. The competition happens when the total number of generators K is greater than the actual number of modes of P_{real}. (2) One beats all: One or a few of the generators are too strong to capture all the modes, while the other generators are forced to move away from the data distribution since the penalty of the classifier is stronger than the penalty of the discriminator. In this paper, we offer novel and efficient techniques to solve the imbalance problems and effectively parallelize the multi-generator model.

Our idea is to dynamically balance between two penalties, based on the stage where each generator stands. To control this competition, we propose a balance term β, where all the training information from all the generators are collected, the current progress of each generator is evaluated, and a decision is made based on the overall stage of all the generators. To further improve and speed up the model, we propose a reverse KL divergence loss function instead of JS Divergence as the generator loss, to avoid mode collapse and improve the generator's ability to capture all the modes. Moreover, our model can allow parallelized training among generators, with synchronized or asynchronized updates for the discriminator, which significantly reduces the training time. Another advantage of our parallelization framework is robustness and extensibility. Increasing or decreasing the number of processors will not hamper the training process. The framework can dynamically adapt to the change. Experimental results show that our model can solve the missing mode problem and generate diverse images by adding generators into the model.

2 Related Works

Recently, many researchers have started focusing on designing a mixture of generators to beat the discriminator. The authors of [13] train different generators to capture different granularities of the image and generate a high-resolution image. The paper [17] uses the idea of Adaboost, where the weight of the misclassified data is increased, and the final model is a mixture of all the weak learners trained in previous steps. A mixture model where multiple generators are trained to play against the discriminator is given in [3]. Given enough number and complexity of generators, a Nash Equilibrium can also be achieved, and the discriminator tends to lose the game. The authors of [9] follow this idea and achieve the state-of-the-art generated quality (inception score). However, these two methods suffer from the imbalance and competition problems mentioned in the previous section. Our method extends this idea in a novel manner. We exploit the fact that given enough generators, with balance control, all the modes can be captured and the mixed generators can finally win.

To understand and solve the missing mode problem, [1] proves that any proxy loss function that contains the reverse Kullback Leibler divergence (KL divergence [12]) term tends to capture a single or few modes of P_{data}, while ignoring the other modes. It has been claimed that an imbalance in data points for different modes may cause the missing mode problem [5]. The generation manifold tends to move to modes with dominating data points while ignoring modes with only a few data points. Chey et al. [5] propose to use an autoencoder to map the data points back to the prior distribution z, and let the generator sample from the mapped distribution prior instead of a simple Gaussian. This paper also introduces an evaluation metric to measure both the generated quality and the ability to handle the missing mode problem, which is not highlighted in the traditional Inception Score measurement ([15]). Unrolled GAN, where copies of the discriminators are made, and back-propagation is done through all of the discriminators, while the generator is updated based on the gradient update of those discriminators has been presented in [10]. In [6], the authors propose a multi-discriminator model, where weak discriminators are trained using parts of the data, and the gradients from all the discriminators are passed to the generator. The authors of [16] have used another reconstructor network to learn the reverse mapping from generated distribution to prior noise. If the support of the mapped distribution is aggregated to a small portion, then the missing mode problem is detected. A dual discriminator model where KL and reverse KL divergence are controlled by two discriminators is offered in [11]. In this model, the weights of the two discriminators are controlled by a neural network.

3 Our Method

The original generative adversarial network was first proposed in [8], and can be formulated as a minimax game between a discriminator D and a generator G, where the loss function can be defined as:

$$\begin{aligned}
\mathcal{J}^D_{\theta^D} &= \mathbb{E}_{x\sim P_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))] \\
\mathcal{J}^G_{\theta^G} &= \mathbb{E}_{z\sim p_z(z)}[\log(D(G(z)))] \\
\theta^G &= \underset{\theta^G}{\operatorname{argmin}}\ \max_{\theta^D} \mathcal{J}^D_{\theta^D}
\end{aligned} \tag{1}$$

For the generator, the optimal discriminator at each step is $D^* = \frac{P_{data}}{P_{data}+P_g}$. When convergence is reached, we can obtain $P_g = P_{data}$. The procedure is equivalent to minimizing the Jensen Shannon Divergence $JSD(P_g||P_{data})$. As discussed in [1], the zero sum loss results in the gradient vanish problem where generator can learn nothing since the gradient is zero. Thus heuristic/proxy loss for the generator G is proposed. As is proved in [1], the gradient of the heuristic loss is equivalent to the gradient $\nabla_{\theta^G}[KL(P_g||P_{data}) + JSD(P_g||P_{data})]$.

3.1 Loss Functions

In our work, we design a multi-player game by dividing one generator into K generators, and adding another classifier to the original minimax game. The loss function for every single generator is:

$$\begin{aligned}
\mathcal{J}^D(G,D) &= \mathbb{E}_{x\sim P_{data}}[\log D(x)] + \mathbb{E}_{x\sim P_G}[\log(1-D(x))]\\
\mathcal{J}^C(G,C) &= \mathbb{E}_{x\sim P_{g_{-k}}}[\log C(x)] + \mathbb{E}_{x\sim P_{g_k}}[\log(1-C(x))]\\
\mathcal{J}^{G_k}(G_k,C,D) &= \mathbb{E}_{x\sim P_{g_k}}[1+\log D(x) - \log(1-D(x))]\\
&\quad - \beta_k\,\mathbb{E}_{x\sim P_{g_k}}[\log(1-C(x))]
\end{aligned} \tag{2}$$

Since the loss function $\mathcal{J}^{G_k}(G_k, C, D)$ is not bounded, we need to truncate $\mathcal{J}^{G_k}$ if $D > t$ to avoid the gradient explosion problem, where t is a threshold value. The goal is to solve the multi-player minimax game. If we take a closer look at the loss functions, we will notice that: (1) The discriminator loss $\mathcal{J}^D$ is nothing but the loss from the original GAN paper, which minimizes the Jensen-Shannon Divergence (JSD) between the mixture of generators and P_{real}; (2) the classifier loss $\mathcal{J}^C$ is actually another discriminator that treats G_{-k} as real samples, G_k as fake samples, and separates each generator G_k from all the other generators G_{-k} maximizing $JSD(G_k||G_{-k})$. The output of the classifier C is a softmax layer with size K; and (3) each generator is trained according to the gradient provided by both the discriminator D and a weighted classifier C.

We can show that the distance we are minimizing is $D_{KL}(P_{g_k}||P_{data})$ and $-D_{JSD}(P_{g_k}||P_{g_{-k}})$. From [8], the optimal discriminator, given the current generator G, has a close form $D_G^* = \frac{P_{data}(x)}{P_{data}(x)+P_g(x)}$. Since the loss function of C is fairly close to D, we can obtain the optimal C given that the current G is $C_G^* = \frac{P_{G_{-k}}(x)}{P_{G_{-k}}+P_g(x)}$. Next, we will analyze the loss of the generator when we fix $D = D^*$ and $C = C^*$.

Proposition 1. *Given optimal D^* and C^*, minimizing the loss for generator in Eq. 2 is equivalent to minimizing:*

$$D(P_{g_k}, P_{data}, P_{g_{-k}}) = D_{KL}(P_{g_k}||P_{data}) - \beta D_{JSD}(P_{g_k}||P_{g_{-k}}).$$

Proof. We first show that minimizing the first term is equivalent to minimizing $D_{KL}(P_{g_k}||P_{data})$. If we take the partial derivative of the reverse KL divergence:

$$\frac{\partial}{\partial\theta}D_{KL}(P_{g_k}(\theta)||P_{data}) = \frac{\partial}{\partial\theta}\int P_{g_k}(\theta)\log\frac{P_{g_k}(\theta)}{P_{data}}\,dx.$$

We can use Leibniz integral rule to switch integral and derivative, if we assume that the function inside the integral satisfies: 1. continuity, 2. continuous derivative, and 3. $\lim_{x\to\infty} f(x) = 0$. We obtain:

$$\frac{\partial}{\partial\theta} D_{KL}(P_{g_k}(\theta)||P_{data}) = \int \frac{\partial P_{g_k}(\theta)}{\partial\theta} \log\frac{P_{g_k}}{P_{data}} + P_{g_k}\frac{\partial P_{g_k}(\theta)}{\partial\theta}\, dx.$$

Substituting D with optimal D^*, $\mathcal{J}^{G_k}(G_k, C, D)$ can also be rewritten as:

$$\mathcal{J}^{G_k}(G_k, C, D^*) = \mathbb{E}_{x\sim P_G}[1 + \log(\frac{1-D^*}{D^*})] = \mathbb{E}_{x\sim P_G}[1 + \log\frac{P_{g_k}(\theta)}{P_{data}}]$$

$$= \frac{\partial}{\partial\theta}\int \log\frac{P_{g_k}}{P_{data}} P_{g_k}(\theta) + P_{g_k}(\theta)\, dx = \int \log\frac{P_{g_k}}{P_{data}}\frac{\partial P_{g_k}(\theta)}{\partial\theta} + P_{g_k}\frac{\partial \log P_{g_k}(\theta)}{\partial\theta}\, dx,$$

which is equivalent to the gradient of the reverse KL divergence. Note that we assume that $\frac{P_{g_k}}{P_{data}}$ is a constant when optimal D^* is obtained. The second term in the generator loss is the same as the zero-sum loss, which is equivalent to minimizing the Jensen Shannon Divergence $D_{JSD}(P_{g_k}||P_{g_{-k}})$. $\square$

3.2 The Balance Term

For the loss function in the previous section, both the discriminator and the classifier provide gradient to the generator, i.e., the unrealistic error and overlapping error. Note that the two directions may conflict in practice. Based on the information gathered from all the other generators, one should decide whether to focus on minimizing the unrealistic error, or the overlapping error. The information includes: how the generator k performs against all the other generators; how the generator k performs against an ideal generator; and how much overlap is detected by the classifier C. We define these three terms as relative performance w , absolute bias d, and absolute overlap c for the generator k, assuming the total number of generators is K:

$$w_k = \frac{\exp \mathcal{J}_k^D}{\sum_{i=1}^K \exp \mathcal{J}_i^D},\ d_k = \sigma(\mathcal{J}_k^D),\ c_k = \mathcal{J}_k^C \tag{3}$$

The balance term β is constructed based on w, d and c, considering the intuition:

1. c_k and d_k are both high or both low, the generator k is either in the initial stage or in a stable stage, and there is no need to increase or decrease β.
2. c_k is low but d_k is high, the generator k runs outward in a wrong direction, β needs to be reduced to pull P_{G_k} back to P_{data}.
3. c_k is high but d_k is low, the generator k captures a certain mode of P_{data} while conflicting with another generator. β has to be increased to separate the two joint generators.

Synthesizing all the criteria above, we can construct β as:

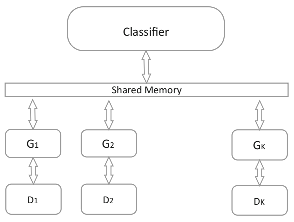
$$\beta_k = w_k \exp(-(\frac{c}{d} - \lambda)) = \frac{\exp \mathcal{J}_k^D}{\sum_{i=1}^K \exp \mathcal{J}_i^D}\exp(-(\frac{\mathcal{J}_k^C}{\sigma(\mathcal{J}_k^D)} - \lambda)) \tag{4}$$

The final β will be renormalized using $\beta_k = \frac{\exp \beta_k}{\sum_{i=1}^{K} \exp \beta_i}$. Note that the sigmoid in the expression is to map the discriminator loss to $R^{(0,1)}$. λ can be interpreted as 'diversity factor' to control the separation among the generators. β will decrease sharply if $c/d > \lambda$. A higher λ will cause a higher penalty from overlapping error which results in a higher generated diversity.

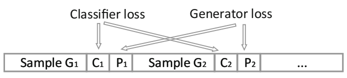
In practice, we also multiply an extra term t decaying with time, where $t = \exp(-\alpha t)$, if the expected number of modes is small or the number of generators is high. Adding the term t forces the overlapping error shrink over time, and reduces the three players game back to two players game, where the convergence is guaranteed.

3.3 Structure of PMGAN

The structure of PMGAN is shown in Fig. 1. All generators and the classifier are connected by shared memory. The communication among them only happens through the shared memory. The shared memory has K slots, where K is the number of generators. Each slot contains three subslots: a sample part where the samples generated by the generator k are stored, a validation part where the value of the classifier is stored, and a progress part where the loss of the generator is stored. Thus the total size of the shared memory is $k(batchsize + 3)$. During training, generator k will store its generated sample in the sample part of k^{th} slot, and continue training. Once the validation or progress slot is updated, the generator will recalculate the overlapping loss or β_k. Classifier C will update once all the sample slots are updated, and store the softmax output to validation slots for each k. Note that it is not necessary that the generator should stop and wait for the response from the classifier or progress from the others since the generator will not go far away from the previous update, and the training process is totally distributed and asynchronized.



(a) structure of PMGAN



(b) structure of shared memory

Fig. 1. Illustration of our proposed PMGAN

4 Experiments

In this section, we demonstrate the practical effectiveness of our algorithm through experiments on synthetic datasets and real datasets. The set up for all the experiments is: (1) Learning rate = 0.0002, (2) Minibatch size = 128 for the generator, the discriminator, and the classifier, (3) Adam optimizer with first-order momentum = 0.5, (4) β is set to 1 at the beginning, with decay $\beta = \exp^{-\lambda t}$, and (5) Activation function is LeakyReLU, weight initialization is from DCGAN [14]. All the codes have been implemented in Pytorch (Inception Score in Tensorflow).

4.1 Synthetic Datasets

Synthetic datasets are a mixture of 8 Gaussians without any overlaps. We have used two settings: 8 generators and 10 generators. First, we train exactly 8 generators with random initialization. In Fig. 2, we show the results for every 5 K steps (discriminator steps).

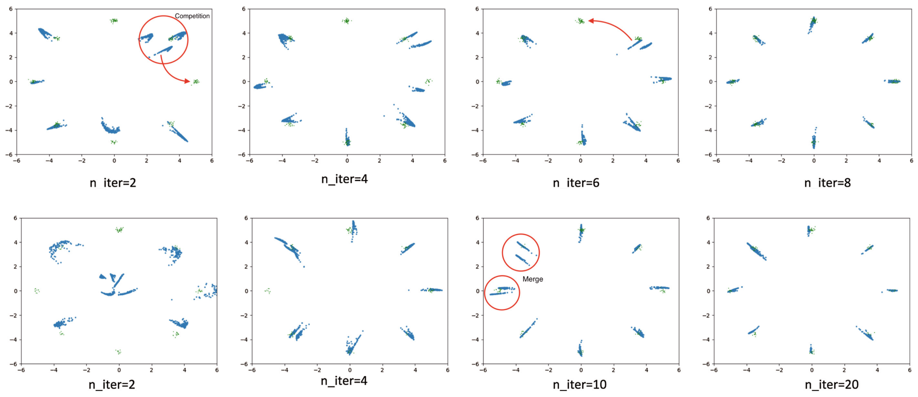


Fig. 2. Evaluation on synthetic datasets. Top: 8 generators, 8 modes. Bottom: 10 Generators, 8 modes

From the results, we see that all the generators are spread out at the beginning. The overlapping penalty and the generators proceeding in the same direction will be divided after certain number of steps. Since the number of modes is exactly the same as the number of generators, the property of the reverse KL divergence will keep each generator stay stationary. When competition happens, the other generators will be pushed to other un-captured modes. Finally, all the 8 modes are captured by different generators.

We have then increased the number of generators to 10. The result is shown in Fig. 3. In the beginning, the situation is the same as in the previous setting, but the strong penalty will hamper the mode captured by two generators.

The two generators are competing for the same mode. This illustrates that the function of the balance term with decay is to 'mediate' the competition among the generators. Two or more generators can collapse to the same mode and reach final convergence after several epochs(determined by the decay factor).

4.2 Real World Data

In this section, we use three popular datasets, MNIST1000, CIFAR-10 and Imagenet. Note that the difference between MNIST and MNIST1000 is that the latter one is constructed using $1,000$ channels to evaluate the missing mode of the model. To evaluate the quality of the generated samples, we use the Inception Score proposed in [15], where the score is calculated by the expectation of KL divergence $\mathbb{E}[D_{KL}p(y|x)||p(y)]$, where we calculate the distance between the conditional label and the real label.

MNIST1000 Dataset: The MNIST dataset contains $1,000$ classes. We ran our model with different numbers of generators ranging from 1 to 16. The result is shown in Tables 1 and 2. Note that by increasing the number of generators, the modes captured by the mixed generator increased, while the distance between generators decreased. Comparing to other models, our model captures all the $1,000$ modes, and obtains the lowest distance between the generated distribution and the real data distribution.

Table 1. MNIST-1000 results for different models

Missing mode evaluation				
Model	GAN	UnrolledGAN	DCGAN	PMGAN
Modes covered	628.0 ± 140.9	817.4 ± 37.9	849.6 ± 62.7	**1,000**
$D_{KL}(model\|\|data)$	2.58	1.43	0.73	**0.06**

Table 2. MNIST-1000 results for different numbers of generators

Num of generator	1	4	8	12	16
Mode covered	140	488	732	977	1,000

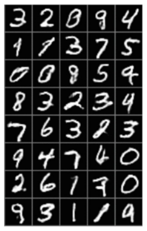
CIFAR-10 and ImageNet Dataset: We trained 1 to 20 generators for CIFAR-10 and ImageNetdataset. From the results, we can conclude that the inception score increases with the number of generators, while it gradually gets saturated. From our observation, the threshold depends on the complexity of the dataset, model capacity, and the classifier. The highest score we get is 8.17 and 9.08, with more than 12 generators, which is very close to the sequential MGAN model. See Table 3.

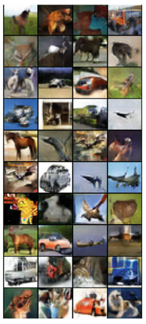
4.3 Training Time

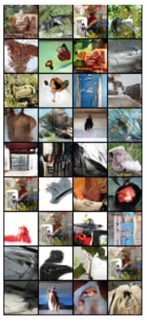
The training time for the sequential mix generator model for CIFAR-10 dataset is 115.4 min in our setting. To obtain around the same score, the PMGAN with 4 generators takes 54% of the time, 8 generators takes 42%, 12 generators takes 37%, and 16 generators takes 35%. The inception scores are 7.02, 8.03, 8.73, 9.01, and 9.08, respectively. From Fig. 3(d), we can observe that with the significantly reduced training time, the inception scores remain unchanged(only with slightly decrease).

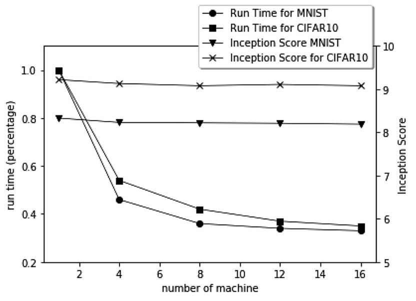
Table 3. Real world data results

Inception score		
Model	CIFAR-10	ImageNet
Real data	11.24 ± 0.16	25.78 ± 0.47
Wasserstein GAN [2]	3.82 ± 0.06	
MIX+WGAN [3]	4.04 ± 0.07	
DCGAN [14]	6.40 ± 0.05	7.89
D2GAN [11]	7.15 ± 0.07	8.25
MGAN [9]	**8.33 ± 0.10**	**9.22**
PMGAN(Our work)	**8.19 ± 0.16**	**9.08**


(a) MNIST


(b) CIFAR-10


(c) ImageNet



(d) Runtime

Fig. 3. (a): Random pick from the mix generator for MNIST dataset. (b): Random pick from the mix generator for CIFAR-10 dataset. (c):Inception score for mix generators and single generator for both datasets. (d): Runtimes and Inception Scores for different numbers of machines

5 Conclusions

In this paper, we propose a novel balanced mixed generator GAN. Our algorithm is parallelizable and can be scaled to large platforms. To resolve the competition and one-beat all problems in the mix generator model, we have designed the reverse KL divergence loss function, and a carefully designed balance term to produce a stable, converging, and fast training method. Experimental results show that we can handle the situation when the generators compete for the same mode even when the number of generators is greater than the number of modes. The empirical results reveal that our method achieves the state-of-the-art performance on the quality of the generated distribution (in terms of the inception score). Also, we show that our model solves the missing mode problem on the MNIST1000 dataset.

More works have to be done in this multi-player game. First, the balance method can also be improved if we can have a better heuristic for β. We can also train to learn β, and to achieve a balance between competition and convergence. The parallelization scheme that we propose can be utilized with other multi-generator models such as the one in [13], to generate better resolution and complex images.

References

1. Arjovsky, M., Bottou, L.: Towards principled methods for training generative adversarial networks. arXiv preprint arXiv:1701.04862 (2017)
2. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein gan. arXiv preprint arXiv:1701.07875 (2017)
3. Arora, S., Ge, R., Liang, Y., Ma, T., Zhang, Y.: Generalization and equilibrium in generative adversarial nets (gans). arXiv preprint arXiv:1703.00573 (2017)
4. Berthelot, D., Schumm, T., Metz, L.: Began: Boundary equilibrium generative adversarial networks. arXiv preprint arXiv:1703.10717 (2017)
5. Che, T., Li, Y., Jacob, A.P., Bengio, Y., Li, W.: Mode regularized generative adversarial networks. arXiv preprint arXiv:1612.02136 (2016)
6. Durugkar, I., Gemp, I., Mahadevan, S.: Generative multi-adversarial networks. arXiv preprint arXiv:1611.01673 (2016)
7. Goodfellow, I.: Nips 2016 tutorial: Generative adversarial networks. arXiv preprint arXiv:1701.00160 (2016)
8. Goodfellow, I., et al.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)
9. Hoang, Q., Nguyen, T.D., Le, T., Phung, D.: Multi-generator gernerative adversarial nets. arXiv preprint arXiv:1708.02556 (2017)
10. Metz, L., Poole, B., Pfau, D., Sohl-Dickstein, J.: Unrolled generative adversarial networks. arXiv preprint arXiv:1611.02163 (2016)
11. Nguyen, T.D., Le, T., Vu, H., Phung, D.: Dual discriminator generative adversarial nets. arXiv preprint arXiv:1709.03831 (2017)
12. Nowozin, S., Cseke, B., Tomioka, R.: F-GAN: training generative neural samplers using variational divergence minimization. In: Advances in Neural Information Processing Systems, pp. 271–279 (2016)

13. Okadome, Y., Wei, W., Aizono, T.: Parallel-pathway generator for generative adversarial networks to generate high-resolution natural images. In: Lintas, A., Rovetta, S., Verschure, P.F.M.J., Villa, A.E.P. (eds.) ICANN 2017. LNCS, vol. 10614, pp. 655–662. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68612-7_74
14. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)
15. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans. In: Advances in Neural Information Processing Systems, pp. 2234–2242 (2016)
16. Srivastava, A., Valkov, L., Russell, C., Gutmann, M., Sutton, C.: Veegan: Reducing mode collapse in gans using implicit variational learning. arXiv preprint arXiv:1705.07761 (2017)
17. Tolstikhin, I., Gelly, S., Bousquet, O., Simon-Gabriel, C.J., Schölkopf, B.: Adagan: Boosting generative models. arXiv preprint arXiv:1701.02386 (2017)