MGGAN: Improving sample generations of Generative Adversarial Networks

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Generative adversarial networks (GANs) are powerful generative models that are widely used to produce synthetic data. This paper proposes a Multi-Group Generative Adversarial Network (MGGAN), a framework that consists of multiple generative groups for addressing the mode collapse problem and creating high-quality samples with less time cost. The idea is intuitive yet effective. The distinguishing characteristic of MGGAN is that a generative group includes a fixed generator but a dynamic discriminator. All the generators need to combine with a random discriminator from other generative groups after a certain number of training iterations, which is called regrouping. The multiple generative groups are trained simultaneously and independently without sharing the parameters. The learning rate and the regrouping interval are adjusted dynamically in the training process. We conduct the extensive experiments on the synthetic and real-world dataset. The experimental results show the superior performance of our MGGAN in generating high quality and diverse samples with less training time.

1 Introduction

Generative Adversarial Network (GAN)[1] is a well-known model for generating persuasive and agile samples.Though GANs can generate the samples for data augmentations, it suffers from a problem called mode collapse [2][3][4][5][6], in which the generator collapses and only generates the samples with limited variety. If the discriminator identifies the samples produced by the generator as genuine, the generator will always produce similar distributions. It is considered a major challenge to extract the feature distributions from complicated datasets with multiple object classes (e.g., ImageNet[7]) since it is difficult for GAN to converge and sometimes the Nash equilibrium does not even exist. In this case, mode collapse becomes a prominent problem. The mode collapse problem is even more severe when using GAN to generate faces. This is because the initial layers in the model mostly learn the same fine-grained feature distributions for a specific type of dataset. In this phase, the generator is limited to targeting a particular pattern even though the discriminator offers the favourable feedback. There are two widely used approaches to deal with the puzzle: 1) reinforcing the GANs' learning ability [2][5][6], and 2) ensuring the GANs can extract a variety of modes from different data distributions [3][4][8]. This work takes the second approach.

In this work, we propose a framework called MGGAN (Multi-Group Generative Adversarial Nets) to effectively solve the mode collapse problem and increase the diversity of the generated samples, even for the complicated dataset such as face images. MGGAN trains a set of generative groups simultaneously, each of which is constituted by a generator and a discriminator. During the training, the generators and the discriminators will be regrouped. A generator will be paired with a discriminator from another randomly selected generative group. The regrouping gives the system the opportunity of inheriting the network parameters from the previous group, which prevents the system from dropping the distributions that have been learned.

The challenge of introducing multiple generative groups and performing regrouping is as follows. After each regrouping instance, a new network is formed in each generative group. The new network may produce worse results in the initial period of a regrouping instance because the network starts to capture new features, and then recover gradually to generate better samples. This may generate many local optimums in the loss curve. We made careful observations to the loss trend during the training in this multi-group setting, and propose a learning rate adjustment strategy to allow the network to jump out of the local minimums swiftly. Further, we propose a strategy to determine the regrouping interval dynamically (i.e., the number of training iterations between two consecutive regrouping instances in MGGAN). The learning rate adjustment strategy combined with the regrouping strategy can facilitate MGGAN to learning disparate modes and sharing the network parameters efficiently among multiple generative groups.

We conduct the extensive experiments with MGGAN on a synthetic dataset and three realistic datasets (MNIST, CIFAR-10, Lfw). In the experiments, DCGAN [9] is implemented with our MGGAN framework and is evaluated in terms of the metrics of IS (Inception Score) and FID (Frechet Inception Distance). The experimental results show that MGGAN can generate high quality samples while reducing the training time. Moreover, MGGAN can be applied to other GAN variants. Namely, we can train several different types of GANs simultaneously by employing MGGAN in the same training procedure..

2 Related Work

GANs have shown impressive success in generating realistic, high-quality samples when being trained on the class-specific datasets (e.g.Faces[10]).However, GANs suffer from the mode collapse problem. Many efforts have been made to attack this problem from the following different perspectives.

2.1 Covering diverse modes

Many methods were proposed to solve mode collapse by improving the diversity of the generated data[11] [12]. For example, VEEGAN adds a reconstruction network by introducing an implicit variational encoder to map from data to noise[11]. GDPP uses the determinantal point process to enforce the generated data to have a similar distribution of real data[12].

2.2 Enhancing the training process of the network

The Unrolled GAN proposes a novel method to tackle the instability by defining an unrolled optimization of the discriminator[5], which shows the reduction of mode collapse. WGAN presents a new cost function based on the Wasserstein distance that has a smoother gradient [13]. This method can improve the generation's convergence.

2.3 Using multiple generators and discriminators.

Another way to reduce mode collapse is by adopting more than one generator or discriminator to capture various modes. The MAD-GAN [14] combines multiple generators with one discriminator. The system encourages each generator to capture its mode. CoGAN[8] proposes a extension to model pairs of corresponding images in two different domains. The model combines two GANs and shares the weights of the higher layers of both generator and discriminator. D2GAN [15] propose to couple two discriminators with one generator to reduce the mode collapse by minimizing the lower bounds of KL and reverse KL. [16] propose a framework to parallel many networks and pick the best one to cover modes.

However, these methods have some disadvantages, which might result in the following problems. i) The methods[16] and [14] will make the discriminators much strong, which lead to the vanishing gradient problem. The generators can not receive enough gradient to make progress from an optimal discriminator. A powerful discriminator does not provide enough information for parameter updating. ii) Some methods increase the complexity of the network structure and loss function. It might increase the computation time or make the model even harder to converge [8]. iii) Some approaches [5] and [13] have limited ability of generalization. The methods are based on some particular training models, which means it is difficult to apply them to other training networks.

3 The MGGAN Framework

3.1 GANs and Generative Group

In this subsection, we first present the background of Generative Adversarial Network (GAN) briefly[1] and the notations we will use in this paper, and then present the generative groups to be used in MGGAN.

GAN aims to model high-dimensional distributions from realistic data. It defines a game of two players: a G (Generator) and a D (Discriminator). A generator tries to map a vector z from the noise space Z to the realistic data space, generating a sample $G(z)$ to simulate the real data and fool the discriminator. The discriminator D takes an input data x and outputs a probability, denoted by $D(x)$, that x is a piece of real data. $D(x)$ is computed by determining the probability that x is from the real data distribution (denoted by $P_{data}(x)$) and the probability that x is from the samples produced by the generator G. For example, when G is fixed, the optimal discriminator uses Eq.1 to compute $D(x)$.

$$D_G(x) = \frac{P_{data}(x)}{(P_{data}(x) + P_G(x))} \tag{1}$$

The GAN framework can be modelled as a minimax two-player game, in which G and D are trained jointly via solving Eq.2[1], in which the first term calculates the expectation of the output of D (i.e., $D(x)$) over all data following the real data distribution while the second term calculates the expectation of the output of D over all noise data z that are used by the generator to produce the samples (i.e., $G(z)$).

$$\begin{aligned}\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \\ \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]\end{aligned} \tag{2}$$

In Eq. 2, minimizing $V(D,G)$ over G can be transformed to minimize the Jensen-Shanon (JS) divergence between the two probability distributions: $P_{data}(x)$ and $P_G(x)$, denoted by $D_{js} = (P_{data}(x) \parallel P_G(x))$. At the Nash Equilibrium, the distribution of the generated samples is similar to the real data distribution, i.e., $P_{data}(x) = P_G(x)$. The discriminator gets $D(x) = 0.5$ for all x, indicating it can not tell whether its input data is real or generated.

To tackle the mode collapse problem in the GAN training, we propose a new GAN training framework called MGGAN. The standard GAN consists of a generator and a discriminator, which interacts in the same network. In MGGAN, multiple generative groups are generated, each of which consists of a generator and a discriminator. After a certain number of training iterations (called regrouping interval T) in MGGAN, a generator is regrouped with a discriminator randomly selected from a generative group. Between two consecutive regrouping instances, the generative groups are trained simultaneously without sharing the parameters.

Empirically, the generator accepts the data from the noise space, which are of low dimension, to produce the samples, while the real data are usually high dimensional data. The high dimensional data convey much more information than low dimensional data. Thus after a regrouping instance, the samples produced by the generator in a new group will be distinct (new) for the discriminator. The regrouping presents the opportunity to improve the diversity of the generated samples because a generator needs to produce various samples to fool the new discriminators after each regrouping. However, the regrouping also challenges the discriminators' recognition ability because they have to identify the samples produced by different generators, which may slow down the training process or even make the training difficult to converge. This is the reason why we propose the strategy to adjust the learning rate dynamically.

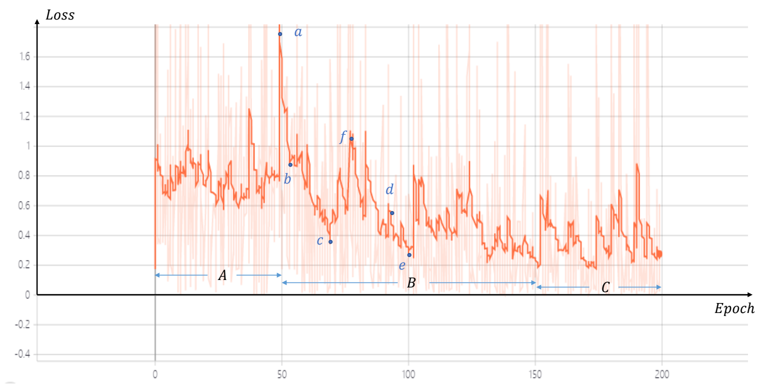


Figure 1: The loss trace of a generator with the G-D regrouping.The graph generated by 2 generative group based on the DCGAN and CIFAR-10 with E=200, T=100.

3.2 The Strategy of Adjusting the Learning Rate

Our MGGAN can work with any GAN model based on a single (G, D) group (which we call the *base GAN model*), such as DCGAN [9]. When we implement MGGAN to work with a base GAN model, multiple (G, D) groups are generated with each group trains with the base GAN model. During the training, regrouping is performed according to our strategy to be presented in this section. Note that different generative groups can also train with different base GAN models.

As we have discussed in previous section, regrouping offers the opportunity to increase the diversity of the generated samples. But after each regrouping instance, Generators and Discriminators will face new partners and need to capture new features, which may cause the loss to increase after regrouping and consequently increase the training time. We conducted the experiments to observe the trend of loss as the training progresses with regrouping being performed from time to time. Fig.1 shows the loss trace of a generator in a randomly selected generative group over epochs (up to 200 epochs) when training two generative groups on CIFA-10 with DCGAN as the base GAN model. After the careful observations and analysis to the loss trace, we found that the training process with regrouping can be divided into three periods, which are labelled as A, B and C in Fig. 1.

Period A is the period before the first regrouping instance. In period A, the loss may oscillate in the early stage of the train and then begin to decrease. Period B starts with a dramatic increase in loss (point *a* in Fig. 1). Then the following general pattern occurs throughout period B: the loss decreases fast to some point (point c in Fig. 1), increases dramatically again (to point *f* in Fig. 1) and then decreases to a point (point *e* in Fig. 1), which may be lower than the previous low point (i.e., point *c*). Our experimental records show that the dramatic increases in loss during period B are due to the regrouping in the training. This phenomenon is reasonable, because in the initial period of each regrouping instance, the samples generated by a generator is new and distinct to the discriminator in the new group. If the generator could learn the new features effectively after a regrouping, the loss may decrease to a point lower than the previous low point. In Period C, the loss becomes fluctuating without showing the decreasing pattern seen in period B. Based on the above observations, we propose the following strategy to set the learning rate as the training progresses.

In period A, we use the learning rate set by the base GAN model. When the loss does not show the obvious decrease, we set it as the end of period A and performs the first regrouping. Now the training process enters period B. We adjust the learning rate in each generative group back to the value at the beginning of period A. This is because after regrouping, each generative group needs to learn new features. After the dramatic increase in loss after a regrouping, the loss will typically decrease fast (e.g., from point *a* to *c*). This is because the network has learned some features in previous training iterations. A problem of the regrouping scheme is that many deep local optimums (such as point c) appear in period B since the loss decreases fast after last regrouping instance and will increase drastically when the next regrouping instance is performed. Without appropriate handling, the training may be stuck in a local optimum. This is the reason why the loss may not decrease to

Algorithm 1 Adjusting the learning rate in Period B

```
Input: current iteration ($i$), the iteration at which the next regrouping is performed ($i^T$), Learning
       rate adjusting region ($\alpha$), upper limit of learning rate ($\overline{L}$), learning rate adjusting ratio ($\delta$),
       Variation Ratio of the Loss ($\gamma$), threshold of VRL ($\overline{\gamma}$), current learning rate ($L$), learning rate
       at the beginning of period A ($L_0$)
Output: the adjusted learning rate ($L^{'}$)
1 if ( $i^T - \alpha \leq i \leq i^T + \alpha$  and  $\gamma \leq \overline{\gamma}$) then
2     $L+ = L * \delta$;
3     if $L \geq \overline{L}$ then
4         $L = \overline{L}$;
5     end
6 end
7 if ($i = i^T + \alpha + 1$) then
8     $L$=$L_0$
9 end
```

a point lower than the previous low point after a regrouping. In order to address this situation, we adopt the strategy outlined in Algorithm 1 to adjust the learning rate in period B.

Assume the current regrouping interval is T (i.e., T iterations need to be run between the past regrouping instance and the next regrouping). With T, we can know at which iteration (denoted by i^T) the next regrouping instance will be performed. When the iteration index is in the range $[i^T - \alpha, i^T + \alpha]$ (Line 1), we increase the learning rate by a fraction of δ each time (Line 2 in Algorithm 1), which aims to help the training escape the local optimum. The training may not be stable if the learning rate is too high. Therefore, a upper limit ($\overline{L}$) is used to cap the learning rate (Lines 3-4). When the training comes out of the range of $[i^T - \alpha, i^T + \alpha]$ (e.g., α is 10 iterations), it means that a new grouping is performed, we adjust the learning rate back to the initial value at the beginning of period A (L_0) to allow the base GAN model to learn new modes. There are the variables γ and $\overline{\gamma}$ in the algorithm. We will present the meaning of them in the next subsection. In this algorithm, α, δ $\overline{L}$ and $\overline{\gamma} are the hyper-parameters in the training.$

3.3 Variation Ratio of the Loss

In Line 1 of Algorithm 1, the *if* condition includes a term $\gamma \leq \overline{\gamma}$. This is related to a notion we propose in this work: Variation Ratio of the Loss (VRL). γ is the current value of VRL while $\overline{\gamma}$ is the threshold value of VRL. VRL is calculated by Eq. 3, where $L(n)$ is the loss value at iteration n. The idea of introducing VRL is based on the following and considerations and observations to the training process.

$$VRL_n = |L_{n-1} - L_n| / L_n \tag{3}$$

First, the reason why we want to increase the learning rate is because we want to jump out of the local optimums quickly. But through the observations to the training process, we found that when there is a sufficient change in the loss between two consecutive iterations (i.e., $|L_{n-1} - L_n|$) compared with the current value of the loss, it is very likely that the training can jump out of the local optimums itself without the need of adjusting the learning rate. It is reasonable because when there are sufficient changes in the loss between iterations, it means that the current learning rate is performing well.

Second, Algorithm 1 takes i^T as input. The value of i^T is directly based on the value of the regrouping interval T (we will present how to determine T in next subsection). The computed value of T may not be accurate. The consequence of the inaccurate T is that the next regrouping and the adjustment of the learning rate are not performed around the local optimum regions (we will discuss the impact of inaccurate T in more detail in next subsection). Introducing VRL can serve as a remedy scheme in the case where the value of T is accurate. Namely, when we attempts to adjust the learning rate, the loss curve is not around a local optimum region. Then it is likely that the current training iterations still produce sufficient changes in loss, which can be detected by checking whether $\gamma \leq \overline{\gamma}$.

3.4 The Regrouping Interval

The regrouping interval is the number of iterations performed between two consecutive regrouping instances. We draw an analogy to illustrate the impact of the regrouping interval on the training process. Suppose each generative group is a student who needs to solve a set of mathematical questions and different students are given different sets of questions. Regrouping is to swap the question sets among the students. The regrouping interval T is then the time given to the students to work out their questions before they face new questions. If T is set too big, the generators cannot learn more features from the current samples and waste the training time. If T is too small, the generators do not have enough time to learn the features in the current samples before they are forced to learn other features, which may aggravate the concussion during the training, and even worse the training may never converge. Therefore, if the value of T cannot be accurately predicted anyway, we would rather T to be overestimated than to be underestimated.

Based on the above discussion, we propose the following strategy to determine the regrouping interval T. The initial value of T, denoted by T_0, is set as the length of period A. Given the current T, the next T is calculated by $T = T * \xi$, where ξ is a number greater than one and is a hyper-parameter in the training. This suggests that in our MGGAN the regrouping interval increases as more regrouping instances are performed. The reasons for this are two-fold.

First, as a generator has learned more features (modes), it needs, in theory, to spend longer time in capturing additional new feature. This is because the network needs to adjust more parameters associated with the modes which have been learned to minimize the divergence in the characteristic distributions between the generator and the discriminator. As the training progresses, more regrouping instances are performed and more modes have been learned. Therefore, it should take longer for a generative group to learn even more modes. Hence the regrouping interval should be increased as subsequent regrouping.

Second, as discussed above, that we would rather T to be overestimated than to be underestimated.

We also set an upper limit for T, denoted by $\overline{T}$, which is a hyper-paramter in training. The reason is because when T is too big, the frequency at which the generative groups are fed new samples will be too low. We found this will cause the generative group to regard the new features contained in the new samples as the noise. This claim is also supported by our experimental results (Fig. 4 in the experiment section).

4 Experiments

In this section, we conduct the experiments to evaluate MGGAN and analyze its behaviour. Both synthetic data and real-world datasets are used. The synthetic data are used to visualize the generated samples and evaluate the learning ability of MGGAN. Meanwhile, we use the real-world datasets to demonstrate the reliability and robustness of MGGAN in tackling the mode collapse in a high-dimensional data space. We train our model on a GPU server equipped with two 1080Ti GPUs. This equipment meets the resource demand for training the GAN.

4.1 Experimental Setup

The GAN Structure: In the experiments, we adopt DCGAN[9] as the base GAN model . In addition, we can use different types of generative models as the base GAN model in MGGAN. For example, we can use both DCGAN and WGAN as the base GAN models. The two GAN models can be trained simultaneously and the network shares the parameters between them.

Datasets: We conducted the experiments on both synthetic data and realistic datasets. The synthetic data can evaluate the ability to extract multiple data modes. Though the synthetic data has limited modes and diversity, it can be quickly assessed via visual inspection. The large-scale datasets includes MNIST [17], CIFAR-10 [18] and LFW [19] (LFW is a dataset of face photographs designed for studying the problem of unconstrained face recognition).

Evaluation Metrics: Inception Score (IS) [6] and Frechet Inception Distance (FID) [20] are used as the evaluation metrics. IS is the most widely adopted metric. However, one disadvantage of IS is that it might make the wrong decision if the generators only produce one image per class. FID is sensitive to mode collapse. So FID is a better assessment of evaluative image diversity. A lower value of FID indicates better image quality and variety.

4.2 Experimental Results

4.2.1 Synthetic Data

In these experiments, we use the synthetic data to evaluate the ability of MGGAN to learn and capture multiple data modes and avoid mode collapse. A simple MGGAN architecture is trained on a 2D mixture of 8 Gaussians arranged in a circle. The architecture and the list of hyperparameters are shown in Appendix A. We did not apply our adjustment strategy for the learning rate and the regrouping interval in this training because i) the toy dataset has limited modes and ii) we want to evaluate the impact of different regrouping intervals on the training outcome. Figure.2 shows the output of MGGAN and the original GAN. The generated samples are around the authentic modes of the distribution but usually ignore some specific modes. When we apply MGGAN to the training, the generators converge to the real distribution quickly, and the system learns all the modes with the suitable regrouping interval. More experimental results will be presented in Appendix B. The top

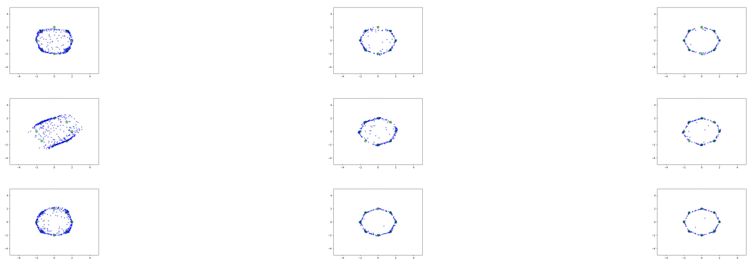

(a) E=600 (b) E=1500 (c) E=2700

Figure 2: The MGGAN training output on a 2D mixture Gaussians dataset. The columns display the data distribution after certain epochs, while the first, second and third row show the training output with the regrouping interval of 0, 25 and 300, respectively. The blue dots are the generated samples, and the green dots are the targeted distribution.

row of Fig.2 shows that the GAN without regrouping ignores a specific mode. The second and third rows employ MGGAN and converges to the targeted distribution quickly. However, we notice that the data distribution in the second row is more confusing. The distribution spreads out with disorder due to the distinct value of regrouping interval T. The learning procedure in the second row has more regrouping instances than that in the third row, which means that the model requires sufficient time to capture explicit features before they learn new samples.

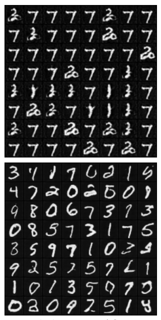
4.2.2 MNIST

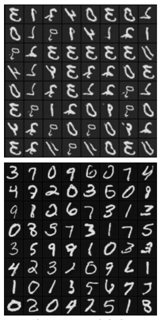
Through this experiment, the generated samples demonstrate that our method produces various modes of data, which can be seen by inspecting Fig.3. The model without regrouping generates the same modes even if it has been trained maturely. The samples in the top row generate same digits, such as 3 and 7, after a long time of training. The second row can produce a variety of modes from the early stage of training. The images in the second row demonstrate that MGGAN increases the quality and diversity of the generated samples.

4.2.3 CIFAR-10

CIFAR-10 is a real-world dataset with high-dimensional images and is much more complicated compared with the MNIST. Therefore, we test MGGAN on a more effective convolution architecture called DCGAN [9]. In this experiment, we employ our learning rate adjustment strategy, but use different regrouping interval to examine the impact on the quality and the time cost of the samples.

We first show the performance in terms of IS on CIFAR-10 collected from the standard GAN and MGGAN with the same network structure. We set the output from the standard GAN as the benchmark to evaluate the performance. The positive values in the figures mean that MGGAN returns a


(a) E=50


(b) E=400

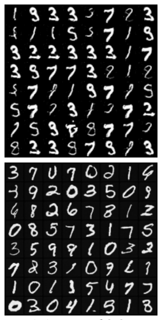
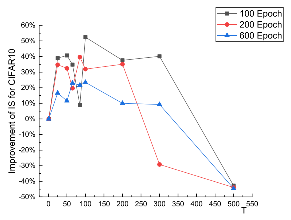
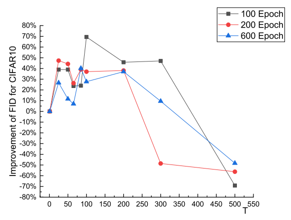

(c) E=600

Figure 3: MGGAN avoids mode collapse for a GAN trained on MNIST. The top row was generated by the standard GAN, while the bottom row was generated by MGGAN. The images are generated by the generators after the specified number of training epoch.

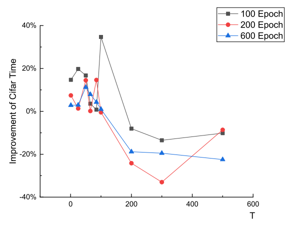
better result. Fig.4(a) and Fig.4(b) shows that MGGAN can improve the capability of the generative model in producing the images with high quality and diversity. Fig.4(c) shows the effectiveness of MGGAN in accelerating the training.



(a)



(b)



(c)

Figure 4: The performance improvement over the epochs using MGGAN on CIFAR-10 with T=0, 25, 50, 65, 85, 100, 200, 300, 500. a) Improvement of IS; b) Improvement of FID; c) Improvement of training time.

Fig.4 shows that when the regrouping interval is too high (more than 200), MGGAN produces worse results than the standard GAN. It suggests the regrouping interval T cannot be too big. The reason is because when T is too big, the generative groups are fed with new samples very infrequently. Then the generative groups are likely to treat the new features in the new samples as noise. This is reason why we set a upper limit for T in the training.

MGGAN can speed up the training and capture the distribution if enough iterations are provided. From Fig.4(c), we can see that our approach can significantly reduce the training cost with suitable regrouping interval.

Finally, we randomly select several samples generated by MGGAN trained on the CIFAR-10 and present them in Fig.5. It shows that MGGAN can produce visually appealing images with convoluted features like cars, trucks, aeroplanes and animals.

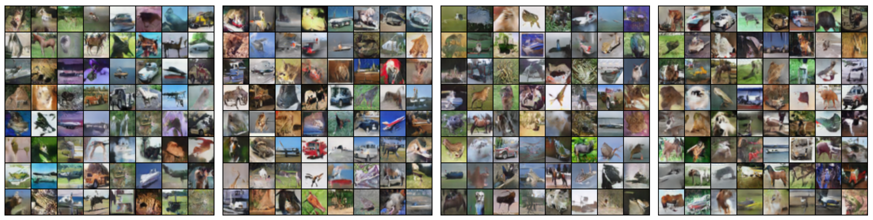
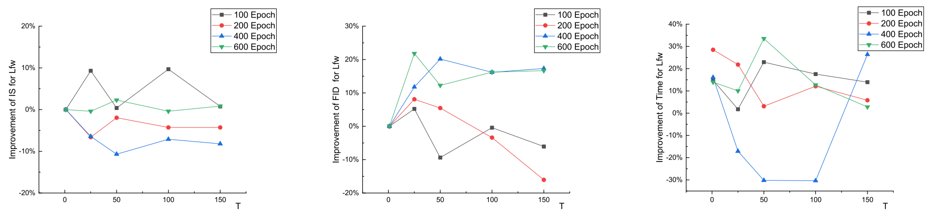


Figure 5: Images generated by MGGAN on the CIFAR-10 dataset.

4.2.4 LFW

LFW is a fine-grained human face dataset with sophisticated features such as hair, clothes and facial expressions. These factors make the training more difficult if we do not improve the capability of the generative model. The standard generators often mislay some particular texture, resulting in mode collapse and low-quality samples. MGGAN can guide the generators to learn the fine-grained features with less training time and increase the learning quality of the base GAN models even for the complex data.



(a) (b) (c)

Figure 6: The performance improvement over epochs by using MGGAN on LFW with T=0, 25, 50, 100, 150. a) Improvement of IS for Lfw; b) Improvement of FID for Lfw; c) Improvement of training time for Lfw

Though the data from LFW have much more fine-grained texture than the images from CIFAR-10, MGGAN can still save a lot of training time while keeping the quality and diversity of generated samples. The figures show that MGGAN has better performance in improving IS. Moreover, training process with more epochs can generate the samples with a higher FID value. Fig.7 shows some samples produced by MGGAN after training on LFW.

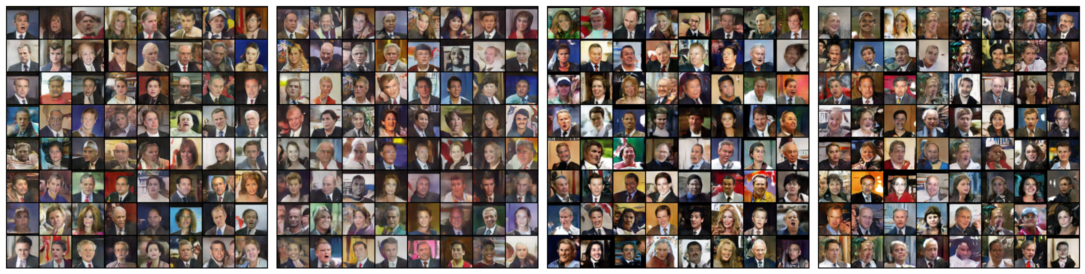


Figure 7: Images generated by MMGAN on the LFW dataset.

5 Conculsion

In this paper, we present a novel GAN framework called MGGAN. In MGGAN, we propose to use a set of generative groups. A learning rate adjustment strategy is proposed to help the model accomplish faster convergence and reduce concussion during the training. The regrouping interval is also craftly determined to ensure the model can capture more modes effectively and efficiently. We have conducted the extensive experiments to evaluate the robustness and scalability of MGGAN by using the Gaussian mixture distribution and the real-world datasets. The experimental results show that MGGAN i) addresses the mode collapse problem well, ii) generates more diverse and higher quality samples with different type of images such as aeroplane and animals, iii) achieves better results when training with fine-grained and complicated datasets such as human faces, and iv) reduces the training cost while maintaining the high quality of the generated samples.

References

[1] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[2] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[3] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks. *arXiv preprint arXiv:1612.02136*, 2016.

[4] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *arXiv preprint arXiv:1606.03657*, 2016.

[5] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

[6] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *arXiv preprint arXiv:1606.03498*, 2016.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[8] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. *arXiv preprint arXiv:1606.07536*, 2016.

[9] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[10] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738, 2015.

[11] Akash Srivastava, Lazar Valkov, Chris Russell, Michael U Gutmann, and Charles Sutton. Veegan: Reducing mode collapse in gans using implicit variational learning. *arXiv preprint arXiv:1705.07761*, 2017.

[12] Mohamed Elfeki, Camille Couprie, and Mohamed Elhoseiny. Learning diverse generations using determinantal point processes. 2018.

[13] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 06–11 Aug 2017.

[14] Dan Li, Dacheng Chen, Baihong Jin, Lei Shi, Jonathan Goh, and See-Kiong Ng. Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks. In *International Conference on Artificial Neural Networks*, pages 703–716. Springer, 2019.

[15] Tu Dinh Nguyen, Trung Le, Hung Vu, and Dinh Phung. Dual discriminator generative adversarial nets. *arXiv preprint arXiv:1709.03831*, 2017.

[16] Daniel Jiwoong Im, He Ma, Chris Dongjoo Kim, and Graham Taylor. Generative adversarial parallelization. *arXiv preprint arXiv:1612.04021*, 2016.

[17] Y. LECUN. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*.

[18] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[19] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. In *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, 2008.

[20] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *arXiv preprint arXiv:1706.08500*, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No] Our discussion does not have any potential negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] The data has been released to public users.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] Our data does not contain personally identifiable information or offensive content.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [Yes]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [No] We do not have any participant risks.
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [No] We do not have any participants payment.