# Analyzing the Mode Collapse Problem in GANS

Nikitha Rao[1], Piyush M. Surana[2] and Nypunya Devraj
[1]01FB15ECS364, [2]01FB15ECS365
Department of Computer Science,
PES University, 100 Ft Ring Road, BSK III Stage, Bengaluru, India

*Abstract*— **It is easier for someone to identify Picasso's painting than drawing one. Generative models, which are known for generating or creating data, are considered much more difficult to build and train when compared to discriminative models, which are known for processing the data. We will understand the working of GANs and discuss the process involved in training. We will also try to understand why training a GAN is hard and how many of the GAN models suffer from major problems such as non convergence, mode collapse, diminished gradient to name a few. In this paper, we will address the problem of mode collapse and look at some of the ways in which we can overcome this problem. We will discuss the working of PacGAN and Unrolled GANS in depth as solutions to the mode collapse problem.**

## I. Introduction to GANs

Traditional neural networks can be fooled easily just by adding a small amount of noise. This usually happens because of overfitting as learning happens from a limited number of images for each class. It could also happen when there is not much non-linearity between the input-output mapping.

One way to solve this would be to train the networks on adversarial samples as well. To generate these adversarial samples, we can use Deep Generative models. There are different generative models like PixelCNN, Variational Auto-encoders and Generative Adversarial Networks (GANs) [1]. Out of these, GANs give the best results.

Generative Adversarial Nets provide a new method to estimate generative models through a process that is adversarial i.e. two models are trained simultaneously- a generative model (G) which is able to learn the distribution of data and a discriminative model that can estimate the probability of a sample coming from training data and not G. The purpose of the generator is to take in noise vectors and produce images that resembles the input data distribution closely and try to fool the discriminator into classifying a fake image as a real image. The function of the discriminator is to classify a generated image as real or fake. Whats going on between the generator and the discriminator here is a 2 player zero sum game. In other words, in every move, the generator is trying to maximize the chance of the discriminator misclassifying the image and the discriminator is in turn trying to maximize its chances of correctly classifying the incoming image.

The basic math in GANs can be described as a minimax game. In the given equation we have two terms. The first term is the expected log likelihood of an image x being real with input samples from real data. We look to maximize the first term because the discriminator needs to maximize its probability of classifying an image correctly as real or fake. Here, the images are sampled from the original data distribution, which is the real data itself. Also, remember that D(x) represents the probability that the input image is real. Hence, the discriminator will have to maximize D(x) (i.e., it has to be close to 1.0) and log(D(x)) . And hence, the first term has to be maximized.

Objective

$$\mathit{min(g)}\ \mathit{max(d)}\ \mathbf{E}_{x\sim P_{data}}[\log(D(x;\theta_d))] + \mathbf{E}_{z\sim P_z}[\log(1 - D(G(z;\theta_g);\theta_d))]$$

$P_z$ -> The data distribution of the noise
$P_{data}$ -> The original data distribution as in the dataset
$x \sim P_{data}$ -> Data sampled from $P_{data}$
$z \sim P_{data}$ -> Data sampled from $P_z$
$\theta_g$ -> The parameters of the generator network
$\theta_d$ -> The parameters of the discriminator network

Fig. 2. Basic Math for GAN

```
for number of training iterations do
  for $k$ steps do
    • Sample minibatch of $m$ noise samples $\{z^{(1)},\ldots,z^{(m)}\}$ from noise prior $p_g(z)$.
    • Sample minibatch of $m$ examples $\{x^{(1)},\ldots,x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
    • Update the discriminator by ascending its stochastic gradient:
        $\nabla_{\theta_d} \frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right].$
  end for
  • Sample minibatch of $m$ noise samples $\{z^{(1)},\ldots,z^{(m)}\}$ from noise prior $p_g(z)$.
  • Update the generator by descending its stochastic gradient:
        $\nabla_{\theta_g} \frac{1}{m}\sum_{i=1}^{m}\log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$
end for
```

Fig. 3. Training Algorithm for GAN

The second term deals with the generated image from the noise sample 'z'. It is the log likelihood of the image generated by Generator being classified as fake by the discriminator.Images from the generators output are passed in to the discriminator. From the Generator's perspective, it has to maximize the chances of the discriminator getting fooled by the generated images. Which means, the generator should want to maximize D(G(z)) . Which means, it should look to minimize 1 D(G(z)) and hence log( 1 D(G(z)).

In a Vanilla GAN, for every 'k' steps of training for discriminator, generator is trained once. We stop the training

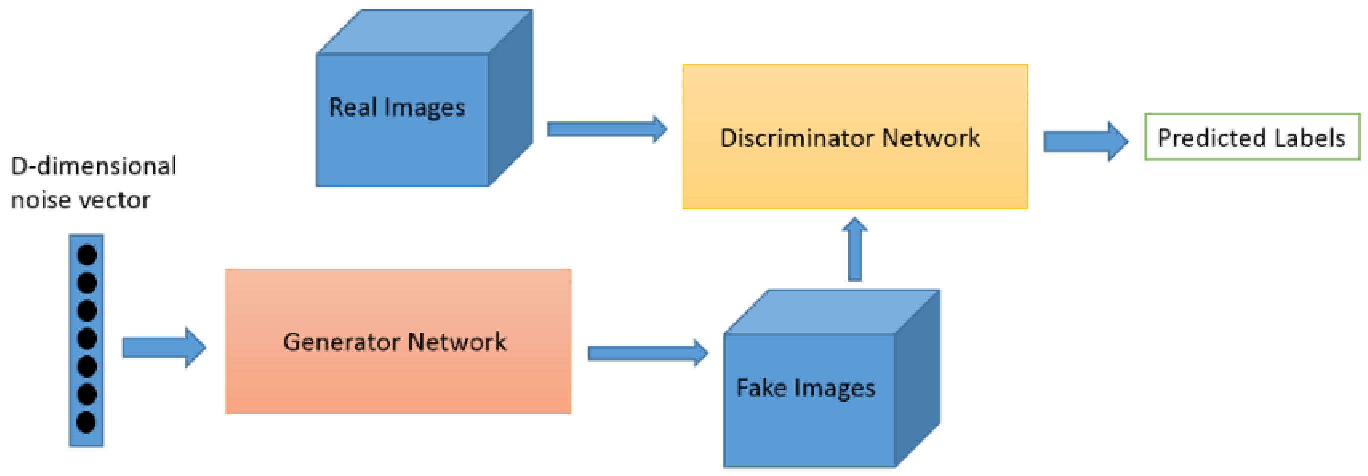


Fig. 1. GAN's Architecture

when the Nash equilibrium is reached i.e. the GAN model converges. Ideally, we would expect the discriminator to always give '0.5' as the output. The generator is the ultimate winner.

GANs are used in various applications today. It is used in text translation to images. It is used when the training data is small. GANs are also used in Drug Discovery, Molecule Development in Oncology and so on.

Many GAN models suffer the following major problems:

- Non-convergence: The model parameters oscillate, destabilize and never converge
- Diminished gradient: The discriminator gets too successful that the generator gradient vanishes and learns nothing
- Mode collapse: The generator collapses which produces limited varieties of samples

## II. Understanding the Mode Collapse Problem

Mode collapse is one of the most important issues to be solved for training GANs. It is a failure case that is encountered very commonly for GANs where the generator is able to learn to only produce samples which have very low variety despite the high degree of variations present in the training set. A lot of interesting real-world data distributions are seen to be multimodal and highly complex. It means that, the data is described by probability distribution which has multiple peaks and different sub-groups of samples are concentrated. When training the GAN a commonly encountered issue is that a mode collapse will occur. This results in the generator outputting samples from a single or limited set of modes. The generator will thus show not very good diversity among the generated samples, which reduces the usefulness of the learnt GAN.

To understand this better, we will take a simple example of a dataset which contains a mixture of readings during the

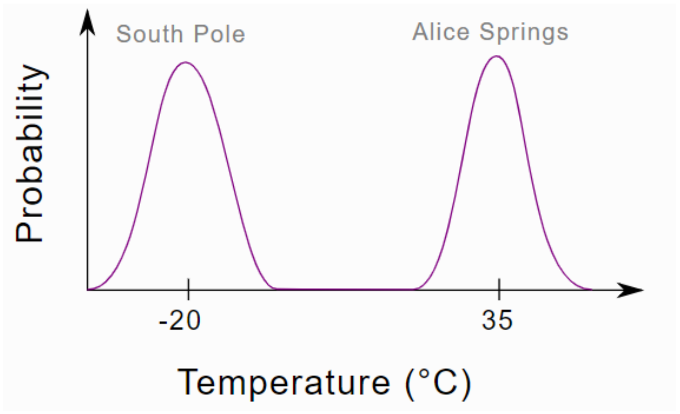


Fig. 4. Probability distribution of temperatures in South pole and Australian Alice springs

day in Summer in Alice Springs in central Australia (usually very hot) and the South Pole of Antarctica (usually very cold). This distribution is clearly bimodal - i.e. there exist peaks at the mean temperatures of the places with a gap in between them. The graph shown in the Figure 4 shows that more clearly.

To train a GAN that can actually produce temperature values that are plausible, is not that simple.Based on our intuition, the generator would be expected to learn to be able to produce cold and hot temperatures with their probabilities being roughly equal. But,the issue of mode collapse can occur, which can make the generator output only samples from one mode (example only hot temperatures). This happens mainly because :

.The generator believes that the discriminator can be fooled into thinking that the generator is outputting real temperatures by outputting only values close to the

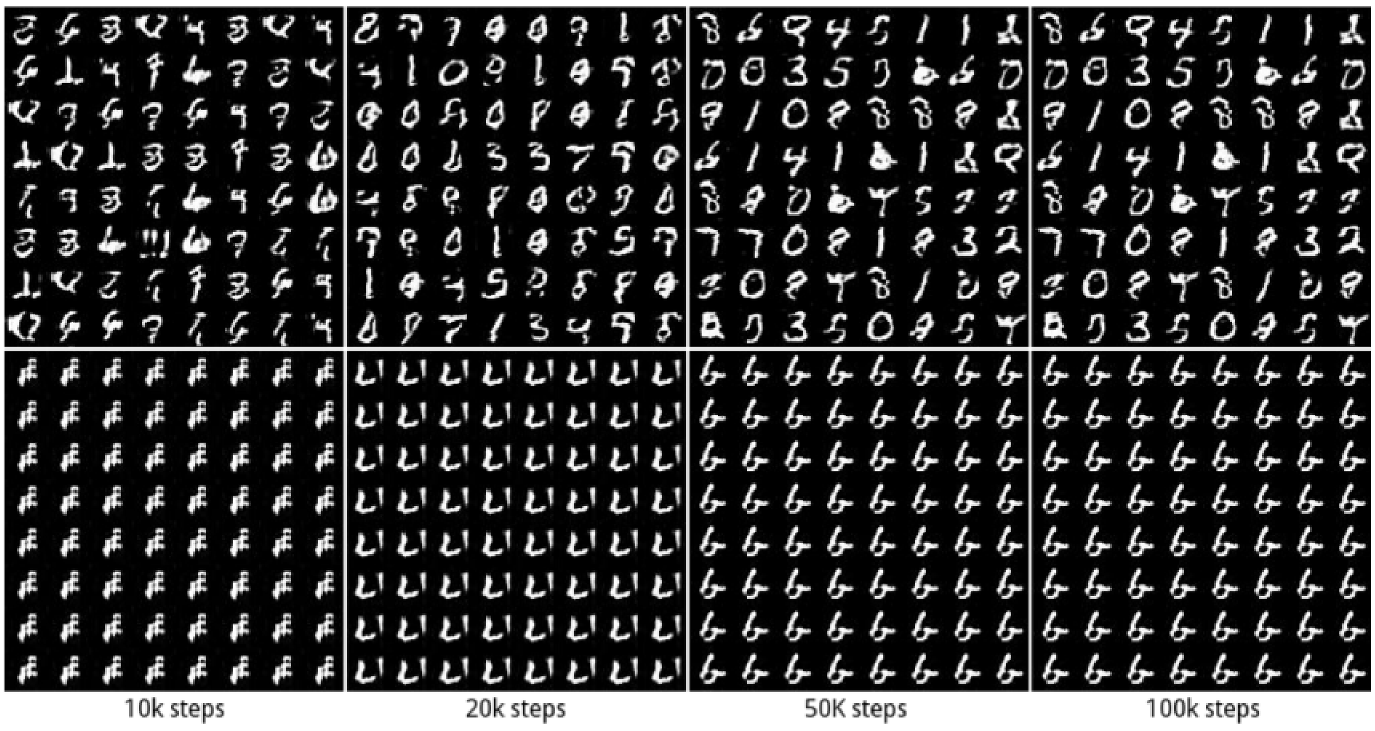


Fig. 5. Mode collapse in generating digits from MNIST

Antarctic temperatures The discriminator can counter that by instead learning that the real temperatures are all hot Australian temperatures (not generated by generator), and it basically guesses cold only Antarctic temperatures since they still are indistinguishable. The generator exploits the discriminator by switching modes to produce values close to Australian temperatures instead, abandoning the Antarctic mode. It is assumed by the discriminator that all the hot temperatures from Australia are fake and the ones from cold Antarctic area are real. The first case is repeated

This game is called the game of cat-and-mouse and it repeats with the generator never being truly incentivised to cover all the modes. The generator collapses and produces only limited varieties of samples.

Another example for this is using the MNIST dataset where there are 10 major modes from digit '0' to digit '9'. The samples in Figure 5 are generated by two different GANs. The top row produces all 10 modes while the second row creates a single mode only (the digit '6').

In reality, mode collapse has a varied level of severity varying from total collapse (all samples generated are very identical) to partial collapse (most sampled have very similar features). Mode collapse can, unfortunately, occur randomly, making playing around with GAN architectures very difficult. Mode collapse is a very well-recognized problem, and a few attempts have been made by some researchers to address it.

## III. Solutions to the Mode Collapse Problem

### A. *Encourage diversity*

It is nearly impossible for us to determine the output diversity when we consider the individual samples in pure isolation. The next logical step is to use batches of samples instead to directly encourage diversity. The two techniques to achieve this are feature matching and mini-batch discrimination.

In feature matching, we modify the cost function of the generator to factor in the diversity present in the generated batches. This is done by matching the statistics of the discriminator's features for the fake batches and that of real batches.

In mini-batch discrimination, we give the discriminator power to compare samples across a batch instead of just an individual sample to determine whether the whole batch is fake or real.

### B. *Using multiple GANs*

Instead of fighting the mode collapse, we could accept that a given GAN will be able to cover only a small subset of all the modes that are present in the given dataset. We can then use multiple GANs and train them for generating data with different modes. When we combine all the GANs, the set will cover all the modes that were in the dataset. The major disadvantage using this approach is that training of multiple GANs is very time consuming and computationally expensive.

### C. *Anticipate counterplay*

One of the ways to prevent the oscillation between modes is to take a peek into the future, allowing us to anticipate counterplay during updation of parameters. The approach here is very similar to minimax in game theory. Intuitively, this idea prevents the players from making a move that can easily be countered in the GAN game.

### D. *Using experience replay*

We can minimize oscillating between modes by showing the discriminator some old samples that are fake from time to time. This is done to prevent the discriminator network from becoming exploitatory, especially for the modes that have not previously been explored by the generator. We can also achieve a similar effect by occasionally substituting the generator or discriminator with an older version for a few iterations.

## IV. Comparison of solutions implemented in Papers

### A. *PacGAN: The power of two samples in generative adversarial networks*

In this project [2], they proposed a principled approach to understand the connection between mode collapse and packing, showing multiple samples simulataneously at the discriminator input. This intuition that packing should help mitigate mode collapse has been around since the original minibatch discriminator.They make this this intuition precise and make the following contributions.

- Conceptual: They propose a mathematical definition of mode collapse, that allows them to measure how severe mode collapse is for a given pair of target distribution P and the generator distribution Q. This allows one to formally compare two generators, in terms of how strong mode collapses they exhibit.
- Analytical: They borrow proof techniques from Blackwell's seminal result in binary hypothesis testing in "Comparison of experiments" in 1951 and data processing inequalities from "The composition theorem for differential privacy" by Kairouz, Oh, and Viswanath, originally introduced for analyzing composition of differentially private mechanisms. This allows them to prove a fundamental connection between mode collapse and packing. In a nutshell, packed GAN naturally penalizes those generators that exhibit strong mode collapse, thus encouraging PacGAN to learn a non-mode-collapsing generators.
- Experimental: They propose a simple architectural change that can be applied to any standard GANs. The simplicity of the proposed architecture allows one to focus on the gains that are resulting from the idea of packing. They measure the gain of packing in benchmark datasets where quantifiable empirical measure of mode collapse has been introduced in the GAN literature, including synthetic mixture of Gaussians, stacked MNIST, and CelebA. They show how packing improves mode collapse in all such benchmark datasets, using the experimental settings, hyper parameters, and the codes from existing literature, for fair comparisons.

*1) Main Idea behind PacGAN:* One of the main reasons GANs suffer from mode collapse is because most of the popular GANs unnecessarily restrict the discriminator to be a function of a single input. In other words, diversity (or lack of diversity) in the generated samples is easier to detect if the discriminator is allowed to make the decision based on multiple samples jointly. Our main contribution is in making this informal intuition formal, by introducing a mathematical definition of mode collapse, and proving that packing naturally penalizes severe mode collapse. When implementing this idea of packing in all our experiments, we make minimal changes to a given mother architecture as shown in Figure 10.

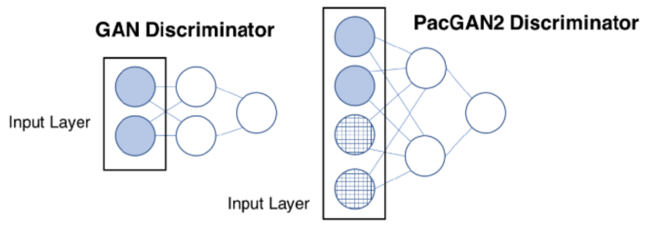


Fig. 6. Difference in GAN Discriminator and PacGAN2 Discriminator

When packing a given mother architecture for empirical comparisons, they were careful in trying to match all hyperparameters used in training. However, there is one hyperparameter that is tricky, which is the minibatch size in the stochastic gradient update. Initially, they matched the minibatch size (for example averaging over 64 sampled gradients for both GAN and PacGAN2) as shown in the figure 7. The reasoning for this choice was that they thought it is fair to have the same number of gradient computation per minibatch for both GAN and PacGAN. However, a question was raised that PacGAN now sees twice as more samples to perform one gradient update, and the gain might be coming from this increased effective minibatch size, which was not their intention.

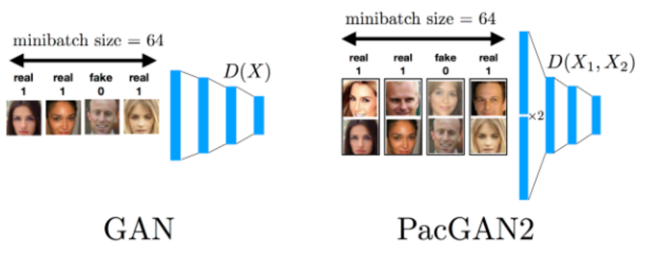


Fig. 7. Difference in GAN Discriminator and PacGAN2 Discriminators Architecture using batch size 64

To address this concern, they redid some of the experiments with the following minibatch update as shown in

Figure 8. They make the minibatch size smaller (by a factor of two) for PacGAN2, such that the number of samples seen by the discriminator per gradient update is the same. Notice that now this is in some sense unfair to the PacGAN2 as they are allowing half as much computation (to compute the gradient). Even then, they observed for the stacked MNIST experiment as per VEEGAN setting that this choice of minibatch size allows PacGAN2 to recover all 1000 modes always.

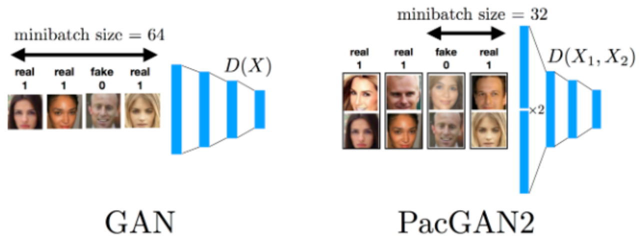


Fig. 8. Difference in GAN Discriminator and PacGAN2 Discriminators Architecture using batch size 32

PacGANs are able to overcome the shortcomings of a GAN, and achieve the remaining performance gain that cannot be gained by simply increasing the discriminator sizes. This suggests that packing gain is fundamental in providing the extra gain not attainable by larger discriminators. This idea is illustrated in the graph in Figure 9

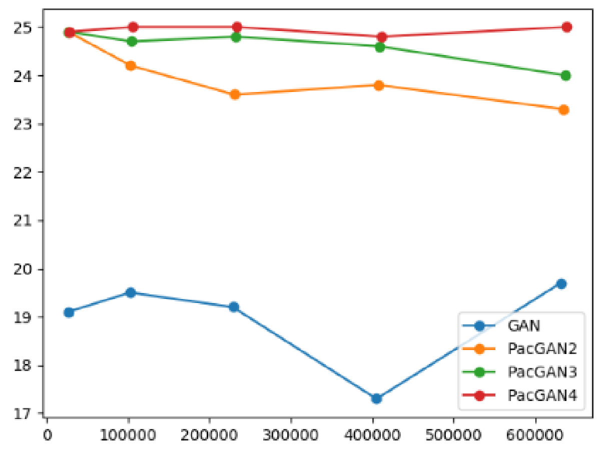
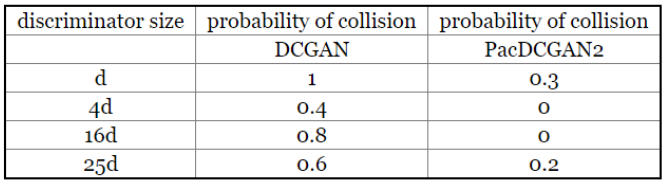


Fig. 9. Performance comparison upon increasing discriminator size

*2) Conclusion:* We train DCGAN and PacDCGAN2 on CelebA dataset, and evaluate the probability of collision for various sizes of the discriminator. The table in Figure ?? shows that for various discriminator sizes, PacDCGAN2 has a smaller collision probability, implying that the PacGAN2 generated more diverse samples when compared to a DC-GAN. Therefore, we can say that PacGAN has successfully overcome the mode collapse problem and provides much more variations in its generated output.



| discriminator size | probability of collision | probability of collision |
|---|---|---|
| | DCGAN | PacDCGAN2 |
| d | 1 | 0.3 |
| 4d | 0.4 | 0 |
| 16d | 0.8 | 0 |
| 25d | 0.6 | 0.2 |

Fig. 10. Results comparison DCGAN PacDCGAN2

## *B. Unrolled generative adversarial networks*

Unrolled GANs [3] allow the generator to ”unroll” the updates of the given discriminator in a completely differentiable manner. So the generator learns to not just fool the present discriminator network, but learns to fool the discriminator maximally after it has responded, thus taking into account counterplay. Some downsides to this approach include increased time in training (as every generator update needs multiple discriminator updates to be simulated) and a much more complicated calculation to get the gradient (back-propogation using an optimiser’s update step is difficult).

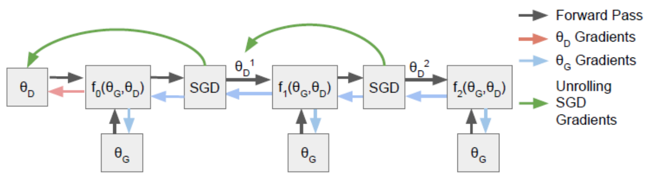


Fig. 11. Unrolled GAN architecture

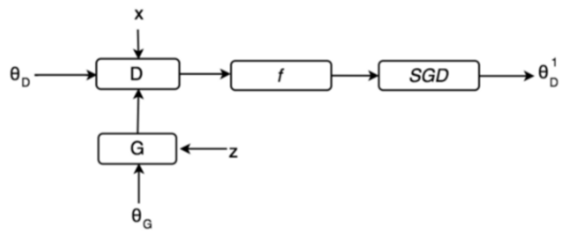


Fig. 12. Discriminator architecture

*1) Main Idea behind Unrolled GAN:* In Unrolled GAN, we give an opportunity for the generator to unroll k steps on how the discriminator may optimize itself. Then we update the generator using backpropagation with the cost calculated in the final k step. The lookahead discourages the generator to exploit local optimal that easily counteract by the discriminator. Otherwise, the model will oscillate and even become unstable. Unrolled GAN lowers the chance that the generator is overfitted for a specific discriminator. This lessens mode collapse and improves stability.

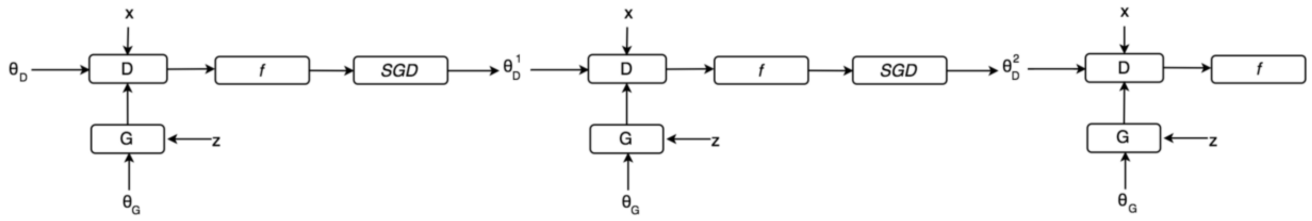


Fig. 13. Generator unrolled for 3 steps

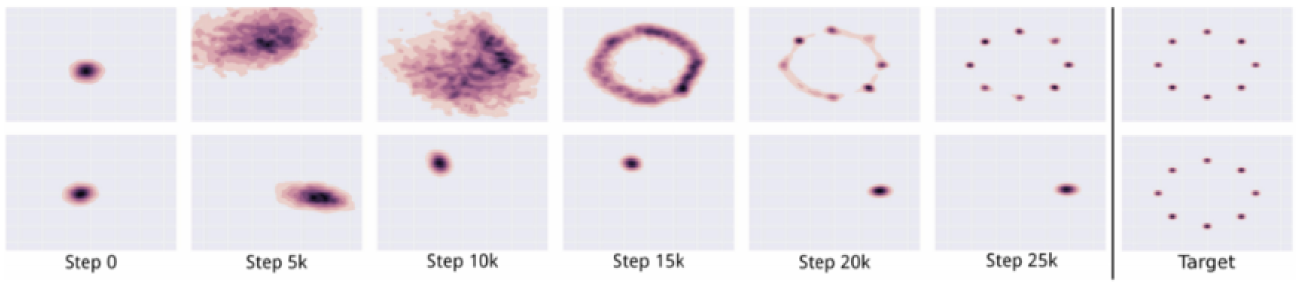


Fig. 14. Results on toy 2D mixture of Gaussians dataset

| Discriminator Size | Unrolling steps | 0 | 1 | 5 | 10 |
|---|---|---|---|---|---|
| 1/4 size of D compared to G | Modes generated | $30.6 \pm 20.73$ | $65.4 \pm 34.75$ | $236.4 \pm 63.30$ | **$327.2 \pm 74.67$** |
| | KL(model \|\|data) | $5.99 \pm 0.42$ | $5.911 \pm 0.14$ | $4.67 \pm 0.43$ | **$4.66 \pm 0.46$** |
| 1/2 size of D compared to G | Modes generated | $628.0 \pm 140.9$ | $523.6 \pm 55.768$ | $732.0 \pm 44.98$ | **$817.4 \pm 37.91$** |
| | KL(model \|\|data) | $2.58 \pm 0.751$ | $2.44 \pm 0.26$ | $1.66 \pm 0.090$ | **$1.43 \pm 0.12$** |

Fig. 15. Comparison of number of modes discovered by a GAN with different number of unrolled steps

*2) Discriminator Training:* In Unrolled GAN, we train the discriminator exactly the same way as GAN.We compute the cost function and use backpropagation to fit the model parameters of the discriminator D and the generator G.

*3) Generator Training:* Unrolled GAN plays k steps to learn how the discriminator may optimize itself for the specific generator. In general, we use 5 to 10 unrolled steps which demonstrates pretty good model performance. The cost function is based on the latest discriminators model parameters while the generators model parameters remain the same. At each step, we apply the gradient descent to optimize a new model for the discriminator. We only use the first step to update the discriminator. The unrolling is used by the generator to predict moves but not used in the discriminator optimization. Otherwise, we may overfit the discriminator for a specific generator.

For the generator, we backpropagate the gradient throughout all k steps. This is very similar to how LSTM is unrolled and how gradients are backpropagated. Since we have k unrolled steps, the generator also accumulates the parameter changes k times.
To summarize, the Unrolled GAN uses the cost function calculated in the last step to perform the backpropagation for the generator while the discriminator uses the first step only.

*4) Conclusion:* In the Figure 14, Unrolling the discriminator will stabilize the GAN training for a toy 2D mixture of Gaussians dataset. The columns represent a heatmap for the generator's distribution upon increasing the number of steps in training. The last column represents the distribution of data. The first row displays training of a GAN with ten unrolling steps. The generator is quick to spread out and converge to the desired target distribution. The second row displays the training of a standard GAN. The generator rotates through the modes of the data distribution and fails to converge to a distribution representative of the target as it only assigns a significant probability mass to a single data mode at once.

Provided with a less complex generator, the GAN in the second row manages to generate good data quality but fail to achieve diversity. The mode collapses. Applying Unrolled GAN, it discovers all 8 modes with high quality (the first row).

## References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozairy, Aaron Courville, Yoshua Bengioz. Generative Adversarial Nets.ArXiv e-prints. 2014

[2] Zinan Lin, Ashish Khetan, Giulia C. Fanti and Sewoong Oh. PacGAN: The power of two samples in generative adversarial networks. CoRR. 2017

[3] Luke Metz, Ben Poole, David Pfau, Jascha Sohl-Dickstein. Unrolled generative adversarial networks. ICLR 2017