Evaluation of Mode Collapse in Generative Adversarial Networks

Sayeri Lala
Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, USA
ls2121@mit.edu

Maha Shady
Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, USA
mashady@mit.edu

Anastasiya Belyaeva
Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, USA
belyaeva@mit.edu

Molei Liu
Department of Biostatistics
Harvard University
Cambridge, USA
molei_liu@g.harvard.edu

***Abstract*—Though widely used in imaging tasks, General Adversarial Networks suffer from mode collapse. Various novel Generative Adversarial Network architectures have been proposed to address the issue. However, there exists little work systematically comparing their performance. This work compares state of the art Generative Adversarial Network architectures, in particular AdaGAN, VEEGAN, Wasserstein GAN, and Unrolled GAN, on datasets with different distributions, include synthetic and real datasets and with respect to several commonly used metrics for quantifying mode collapse. Our findings suggest that AdaGAN performs consistently better among the GANs on nearly all datasets and Wassertstein GAN performs poorly on the datasets. Our results also suggest that one metric is not sufficient to quantify mode collapse for GANs as the metrics do not give consistent results.**

***Index Terms*—Generative Adversarial Networks, mode collapse**

I. Introduction

Generative adversarial networks (GANs) [1] have been demonstrated to be a powerful tool for various generative tasks such as image generation and video prediction [2]–[4]. The scheme of a GAN model is to simultaneously train a generator that maps random noise to data samples, and a discriminator that attempts to distinguish the real samples from the generated samples by assigning high probabilities to the former while giving low probabilities to the latter.

Let $\{x_i : i = 1, 2, \cdots, n\}$ be the training data, drawn from an unknown distribution $p_X(x)$, the generator be G_γ, parameterized by γ, and the discriminator be D_ω, parameterized by ω. G_γ maps input random noise $z \in \mathbf{R}^K$ to the data items $x \in \mathbf{R}^D$, while D_ω maps the real samples and generated samples to distinguished values. Specifically, the model solves the following minimax problem:

$$\max_{\omega}\min_{\gamma} Q(\omega,\gamma) := \mathrm{E}_z[\log\sigma(D_\omega(G_\gamma(z)))]+ \mathrm{E}_x[\log(1-\sigma(D_\omega(x)))], \tag{1}$$

where $\sigma(\cdot)$ represents the sigmoid function.

Despite its success in many fields, GANs has been frequently criticized as fickle to train due to their training sensitivity and instability. GANs have also been reported as suffering from mode collapse, or the missing mode problem [5], [6], where the generator fails to capture some modes present in the training samples. Thus, data generated from GANs can have less variability than data sampled from the original true distribution. From the perspective of min-max theory, mode collapse is caused by the improper convergence of $Q(\omega, \gamma)$ to the optimum, due to the non-convexity and non-concavity of the objective function [7].

Various novel GAN methods have been proposed to handle mode collapse. Many of these methods claim to solve the mode collapse problem and demonstrated the stability and robustness of their method for a particular architecture and on particular datasets. However, since most of these methods are newly proposed, there exists little work that compares their performance systematically. In this project, we study several novel GAN methods and compare their performance via numerical experiments, in terms of generated samples and vulnerability to mode collapse. We select several commonly used metrics, as well as different synthetic datasets and real datasets in our comparative study. To ensure a fair comparison between the GAN models, we control for architecture and training parameters used across GAN variants for each dataset.

This report is organized as follows: section II summarizes the novel GAN methods and metrics studied in our project; section III introduces the datasets, settings, and results of our studies; section IV is our discussion.

II. Methods

A. GAN models

Standard GAN training involves training the generator and discriminator iteratively. The discriminator is updated via stochastic gradient ascent such that

$$\log D_\omega(x) + \log(1 - D_\omega(G_\gamma(z))) \tag{2}$$

is maximized. The generator is trained via stochastic gradient descent to minimize

$$\log(1 - D_\omega(G_\gamma(z))). \tag{3}$$

This is equivalent to minimizing the Jensen-Shannon distance between the models distribution and the data generating process. Several GANs have been proposed to improve the samples generated by the GAN and some methods have specifically focused on the issue of mode collapse in GANs. Here we review the most recent, widely used GANs such as Wasserstein GAN (WGAN) [8], VEEGAN [9], Unrolled GAN [7], and AdaGAN [10].

WGAN: WGAN is motivated by the observation that the Jensen-Shannon distance is not a sensible cost function when learning distributions supported by low dimensional manifolds. The authors propose a more natural measure between two probability distributions, the Wasserstein distance, i.e.

$$W(P_r, P_\theta) = \sup_{||p(x)||_L \leq 1} E_{x\sim P_r}[p(x)] - E_{x\sim P_\theta}[p(x)], \tag{4}$$

where the supremum is over all 1-Lipschitz functions. Intuitively, this distance measures the amount of effort needed to move "a pile of dirt" from one distribution to another. Although it has been known that the Wasserstein distance is a sensible metric for comparing two probability distributions, it is hard to compute in practice. Thus, the authors propose to calculate an approximation to the Wasserstein distance by clipping the weights to ensure that the learned function is indeed k-Lipschitz. Although this GAN was not designed to specifically address the problem of mode collapse, the authors claim that the new distance stabilizes the training process and alleviates mode collapse.

VEEGAN: The main idea of VEEGAN is to introduce a reconstruction network F_θ, which inverts the generator G_γ by mapping the data item or generated sample x to the random noise z. Intuitively, if we find some $F_\theta(\cdot) \approx G_\gamma^{-1}(\cdot)$, and want the generator to recover all modes of the training data x, we should expect that $F_\theta(x)$ is quite close to the distribution of the input random noise, e.g. standard normal distribution. To achieve this goal, $Q_{en}(\theta, \gamma)$ is introduced to penalize the difference between $F_\theta(\cdot)$ and $G_\gamma^{-1}(\cdot)$, and the cross entropy $H(Z, F_\theta(X))$:

$$Q_{en}(\theta, \gamma) := \mathrm{E}[\|z - F_\theta(G_\gamma(z))\|^2] + H(Z, F_\theta(X)). \tag{5}$$

However, such regularization term is not easy to calculate for the samples, so instead Jensen's inequality is used to find an upper bound for $H(Z, F_\theta(X))$:

$$H(Z, F_\theta(X)) \leq \mathrm{KL}[q_\gamma(x|z)p_Z(z)\|p_\theta(z|x)p_X(x)] - \mathrm{E}[p_Z(z)], \tag{6}$$

where $q_\gamma(x|z)$ and $p_\theta(z|x)$ are conditional densities induced by the generator G_γ and reconstructor F_θ respectively. Replacing $H(Z, F_\theta(X))$ with its upper bound in equation (5), the regularization term $Q_{up}(\theta, \gamma)$ is obtained and the objective function becomes:

$$\max_{\omega} \min_{\gamma,\theta} Q(\omega, \gamma) + Q_{up}(\theta, \gamma). \tag{7}$$

Then the training process is formulated as follows:

- Sample $z^i \sim p_Z(z)$, $x^i \sim p_X(x)$, $z_g^i \sim p_\theta(z|x^i)$, and $x_g^i \sim q_\gamma(x|z^i)$.
- With the samples, perform SGD to update γ with respect to $Q(\omega, \gamma) + Q_{up}(\theta, \gamma)$, update ω with $Q(\omega, \gamma)$, and update θ with $Q_{up}(\theta, \gamma)$.

The authors recommend pre-training the reconstructor before the training process to achieve better performance.

AdaGAN: AdaGAN learns a mixture of GANs, inspired by boosting approaches in machine learning. To do this, it first learns weights over the training set. It then trains a component GAN G_c^t at step t on the reweighted version of the training set. G_c^t is added to the current mixture model G_t according to the scheme

$$G_t = (1 - \beta_t)G_{t-1} + \beta_t G_t^c \tag{8}$$

where B_t is some mixture weight computed according to some heuristic.

They prove that their algorithm decreases the f-divergence between the distribution P_g represented by their mixture model G_t and the true data distribution P_d given that the component GAN G_c approximates the optimal component GAN G_c^*. The optimal component GAN at some step t is characterized by the following distribution (see [10, p.9] for derivation):

$$dG_c^*(x) = \frac{dP_d}{\beta}\big(\lambda^* - (1-\beta)\frac{dP_g}{dP_d}\big) \tag{9}$$

This distribution is a reweighted version of P_d, where the reweighting is a function of the mixture weight β and the density ratio $\frac{dP_g}{dP_d}$. The density ratio can be computed with a function of a "mixture" discriminator, which is trained to separate points from the mixture model and the training set. λ^* is a normalization factor.

Thus the component GAN G_c is learned by training on the corresponding reweighted training set where the weights for each sample x_i are given by

$$w_i = \frac{p_i}{\beta}(\lambda^* - (1-\beta)h(d_i)) \tag{10}$$

where $p_i = \frac{1}{N}$, the empirical distribution on the training set of size N and $h(d_i)$ computes the density ratio using the discriminator.

Our studies implement the mixture weight scheme proposed and used in their experiments which is $\beta_t = \frac{1}{t}$ i.e., all component generators are weighted equally in the mixture model.

UnrolledGAN: The unrolled GAN [7] uses unrolled optimization of the discriminator network to create a surrogate objective for the generator update.

$$Q_k(\gamma, \omega) = Q(\gamma, \omega^K(\gamma, \omega)) \tag{11}$$

where K is the number of unrolling steps. The unrolled discriminator update function is

$$\omega^k = \omega^{k-1} + \eta^k \frac{dQ(\gamma, \omega^{k-1})}{d\omega^k} \quad (12)$$

Using this surrogate objective, the generator and discriminator parameter updates are

$$\gamma \leftarrow \gamma - \eta \frac{dQ_K(\gamma, \omega)}{d\gamma} \quad (13)$$

$$\omega \leftarrow \omega + \eta \frac{dQ(\gamma, \omega)}{d\omega} \quad (14)$$

the gradient of the surrogate loss in the generator update is

$$\begin{aligned} \frac{dQ_K(\gamma, \omega)}{d\gamma} &= \frac{\partial Q(\gamma, \omega^K(\gamma, \omega))}{\partial \gamma} \\ &+ \frac{\partial Q(\gamma, \omega^K(\gamma, \omega))}{\partial \omega^K(\gamma, \omega)} \frac{d\omega^K(\gamma, \omega)}{d\gamma} \end{aligned} \quad (15)$$

The first term in this gradient corresponds to the standard GAN update. The second term takes into account how the discriminator would respond to a generator update. The authors hypothesize that this will make the generator favor spreading its mass as it tries to take actions that are hard for the discriminator to respond to, thus preventing mode collapse.

B. GAN evaluation metrics

Several metrics have been proposed for evaluation of GANs for both synthetic and real datasets [10]–[12]. We highlight several metrics that our studies implement.

Wasserstein distance: The Wasserstein distance has been proposed to measure how similar the real and generated distributions are. It is computed by training a Wasserstein critic on training data and evaluating the similarity between generated and test samples. The Wasserstein distance is lower when the two distributions are similar. This metric quantifies overfitting as well as mode collapse.

Coverage: The coverage metric C used in [10] computes the probability mass of real data that is covered by the generated samples as a measure of the quality of the generator. Specifically, let P_{model} and P_{data} be the distribution of the real data and the generated data respectively. For synthetic datasets, kernel density estimation is first used to estimate P_{model}, and its 0.95 quantile t is computed such that $\mathrm{P}_{model}(d\mathrm{P}_{model} > t) = 0.95$. The coverage metric is then computed with $C := \mathrm{P}_{data}(d\mathrm{P}_{model} > t)$. The coverage metric is believed to be more interpretable than likelihood, making it more useful in comparing model performances.

For the MNIST dataset, P_{model} is estimated differently. Assuming that the data has 10 modes (i.e., digits 1-10) [10], a pretrained MNIST classifier is used to select samples with high confidence classifications (i.e., score $>$ 0.999), and the distribution of modes present within these samples is measured.

Number of captured modes and High quality samples: For synthetic datasets, the number of modes, as well as the distributions around these modes are known. We can directly evaluate the performance of the GAN models by counting the number of modes captured and the proportion of the samples around these modes, which are seen as high quality samples. For the low dimensional dataset (described in section III-A1), generated samples within three standard deviations to their nearest mode are counted as high quality samples, while for the high dimensional dataset, we use five standard deviations as our criterion. We specify the number of captured modes in the same way: a mode is counted as captured if we find high quality samples around it. This metric follows evaluations designed in [9].

For the MNIST dataset, assuming it has 10 modes, we compute these quantities by estimating the distance between samples and modes using a pretrained MNIST classifier. The number of captured modes is equal to the number of digits represented by the samples that the classifier is highly confident on. High quality samples are those samples that the classifier is highly confident on i.e., 99.9% confidence score.

Birthday Paradox: The Birthday paradox test in [13] uses the birthday paradox principle to measure the support size of the distribution learned by a GAN. The Birthday paradox states that for a distribution with support size N, a sample of size $\sqrt{N}$ would have a high probability of containing a duplicate.

The proposed test involves obtaining a sample of size s from the generator. The top 20 most similar pairs in the generated sample are then identified automatically based on some similarity metric. Finally, a human then examines the identified sets of duplicates and decides whether a real duplicate exists in the set.

The test is repeated several times to determine the probability of seeing at least 1 duplicate pair in the generated samples. If there a good probability (i.e., $\geq 50\%$) of seeing duplicates, then the support size is at most s^2 since sampling more would only increase the probability of seeing a duplicate.

We propose a modified test that would allow us to estimate the actual support size s_{true}^2 instead of an upperbound. We perform binary search over s to approximate s_{true}. We begin with some initial value for s and run the test described in [13] several times. In our studies, we ran the test 10 times for each s we tried on real datasets and 50 times on synthetic datasets. If we observe collisions in $\geq 50\%$ of the runs, we set this s as an upperbound of s_{true}, and run experiments with a smaller s. If we observe collisions $< 50\%$ of the time with a particular s, we set this s to be a lowerbound of s_{true}. We continue until we reach the minimum value for s that is is sufficient to get collisions $\geq 50\%$ of the time. This becomes our approximation for s_{true}, call it s', and we say that the approximation of the support size for the distribution is s'^2.

For the synthetic and real datasets, the similarity metric for identifying the top 20 pairs is Euclidean distance. For synthetic datasets, we automated the identification of real collisions among the closest pairs, removing the need for manual inspection, as follows. In the case of low dimensional synthetic data sets, we considered a pair of points a collision if their distance was < 0.05 from each other on all dimensions. For high dimensional synthetic data, we defined collision in terms of euclidean distance, and considered a distance smaller than 2.25 to be a collision. This cutoff was determined to be appropriate based on multiple samplings of 1000 data points from the test set of the true distribution. The same cutoffs were used on samples from all distributions, generated and true. For real data sets, collision was determined as described in [13], by a human observing the 20 closest images and deciding whether a pair looks like a collision.

III. Experiments

In this study, we compare the performance of several prominent GANs, specifically WGAN, AdaGAN, VEEGAN and Unrolled GAN on both synthetic and real datasets with respect to metrics that measure mode collapse. For fair comparisons, we use the same architectures and training procedure across the different GAN algorithms as much as possible with minor changes as necessary for each GAN.

Architectures and training parameters were adjusted for each dataset to account for the dataset complexity.

A. Synthetic Data

1) Datasets: In order to compare the number of modes to the ground truth, we generated three synthetic datasets. The first synthetic dataset is a two-dimensional mixture of 8 Gaussians arranged in a ring. The second synthetic dataset is a two-dimensional mixture of 25 Gaussians arranged in a grid. While both of the mentioned datasets are low dimensional, real datasets have a large number of features. Therefore, we also created a high-dimensional synthetic dataset with 800 features, embedding a mixture of nine 500 dimensional Gaussian data. The number of training examples was 500,000 and the number of test examples was 100,000 for each synthetic dataset.

2) Architecture and Training Parameters: We implemented the architecture in [9].

Generator: The generator was a multi-layer perceptron (MLP) with two hidden layers, each having 128 hidden units. The generator took in random normal noise of latent dimension 2 for the 2D ring and 2D grid datasets. For the high-dimensional synthetic dataset, the latent dimension was set to 100, imitating the size of latent dimension used in real datasets.

Discriminator: The discriminator was an MLP with one hidden layer and 128 hidden units. ReLU activation was used.

Training Parameters: Both the generator and discriminator were trained for a total of 100 epochs with batch size of 500. Adam optimizer was used with learning rate of 0.001 and $\beta_1 = 0.5$. Although occasionally it is advised to train the generator and discriminator for different number of steps at each iteration to reach optimality, here since the synthetic datasets are relatively simple we trained the generator and discriminator for the same number of steps.

Algorithm Specifics: For AdaGAN, the number of steps tried is 10. The mixture discriminator implements the same discriminator architecture described above and is trained for 5 epochs. For WGAN, since the orginal GAN paper advised against Adam optimizer, RMSProp was used instead with learning rate of 0.001. For UnrolledGAN, 5 unrolling steps where used.

3) Results: We compared WGAN, VEEGAN, Unrolled GAN and AdaGAN across five different metrics on the 2D synthetic datasets (2D ring and 2D grid). Our results are shown in Tables I and II. The kernel density plots corresponding to generated samples for all GANs are shown in Figure 1 and Figure 2.

Since these 2D distributions are relatively easy to learn, all GANs generated samples that were close in Wasserstein distance (distance $= 0$) to the test samples.

In terms of coverage, which also measures how close the generated and real sample distributions are, VEEGAN had the highest coverage on 2D ring dataset while WGAN and AdaGAN had the highest coverage on the 2D grid. Although some GANs capture the whole distribution of the data, as measured by coverage and number of modes captured, the quality of samples that they generate may not be good. For example, for WGAN, even though it seems have covered the whole space of the 2d grid distribution, the quality of generated samples is very low. For both datasets, AdaGAN generated the highest quality samples.

For synthetic datasets, we can compare the number of true modes to the number of captured modes for each GAN since we have access to the true number of modes. For the 2D ring, the number of true modes is 8 and all GANs except WGAN were able to capture all 8 modes. For the 2D grid dataset, the number of true modes is 25, and only unrolled GAN could not capture all 25 modes.

When computing the support size, on both the 2D ring and the 2D grid, the approximated support size for AdaGAN matched that of the true distribution. UnrolledGAN had a lower support size, which is expected in 2D grid dataset since not all modes were captured. In the case of the 2D ring, where all modes were captured by the Unrolled GAN, the small support size might suggest that the generator assigned higher probability to some modes over others, whereas the true distribution was uniform across all modes. WGAN and VEEGAN had a higher support size than the true distribution with both datasets, suggesting that the generated data might be noisy and includes samples that do not fall into any of the true modes. Overall for the 2D synthetic datasets, AdaGAN seems to consistently perform better across the metrics compared to the other methods.

For the high dimensional synthetic dataset, we compared GANs across 4 different metrics, with results shown in Table III. We did not compute the coverage metric here since the kernel density estimation does not give good estimates for high dimensional datasets and therefore the coverage was always 0 for all datasets. This is a more challenging synthetic dataset since the different modes are embedded in a high-dimensional space, with added noise. Only AdaGAN was able to capture all 9 modes. VEEGAN generated data that is closest to the real data in terms of Wassestein distance and has the highest quality samples but it only captures 6 out 9 modes. Unrolled GAN failed to capture any of the modes, suggesting that the generated samples were mostly noise and it had not actually learned the input data distribution with training paramteres we used. When computing the euclidean distance for birthday paradox metric on the high-dimensional dataset, we only considered the first 500 dimensions as the remaining 300 dimensions are noise. We were unable to approximate a support size for high dimensional data produced from either the WGAN or the Unrolled GAN, as we did not observe any collisions when sampling up to 1000 points. We simply established a lower bound of 1000000 for the support size, which again suggests that the generated data is quite noisy. Our support size approximation for VEEGAN is quite low, which is expected based on the number of modes covered, and may further suggest that it favors some modes over others leading to an underestimation of the support size. AdaGAN had a higher support size than expected.

B. MNIST

1) Dataset: We also compared the performance of GANs on a real dataset, in particular MNIST, which is a collection of hand-written digits. The MNIST dataset contained 60,000 training examples with 10,000 test examples.

2) Architecture and Training parameters: We implemented the architecture and training parameters used in [10], which made use of convolutional layers.

Generator: The generator consists of a fully-connected layer which maps a 100x1 noise input to 7x7x16 output. It is then followed by 3 deconvolution layers, where each deconvolution consists of 5x5 filters and consists of 8, 4, 1 activation maps respectively. Every layer is followed by a leaky ReLu with 0.3 leak and batch normalization. The noise has latent dimension 100 and was initialized with a uniform distribution.

Discriminator: The discriminator consists of 2 convolutional layers and a fully connected layer. Each convolutional layer used 5x5 filters and was followed by leaky Relu with 0.3 leak and batch normalization. The number of activation maps in each convolutional layer is 16 and 32 respectively.

Training parameters: Both the generator and discriminator were trained for 200 epochs with batch size of 128. For each batch, the generator was trained with 2 steps, the discriminator with 1 step. Adam optimizer was used with $\beta_1 = 0.5$ and learning rate for the generator set to 0.005, learning rate for the discriminator set to 0.001.

Algorithm specifics: For AdaGAN, the number of steps tried is 10. The mixture discriminator implements the same discriminator architecture described above and is trained for 5 epochs. For WGAN, once again RMSProp instead of Adam optimizer was used and in addition no batch normalization was applied in concordance with the author's findings that mentioned stability of the WGAN was improved with no batch normalization [8]. It is important that the discriminator trains to optimality for the WGAN in order to obtain a good estimate of the Wasserstein distance, thus the discriminator was trained for 5 steps while generator was trained for 1 step. The Unrolled GAN used 10 unrolling steps.

3) Results: WGAN, VEEGAN, Unrolled GAN, and AdaGAN are evaluated on 5 different metrics: Wasserstein distance, Coverage, % high quality samples, Adjusted Coverage, number of modes captured. Our results are shown in Table IV and random generated samples are shown in Figure 3.

AdaGAN achieved the lowest Wasserstein distance. All methods but VEEGAN achieved scores of 0.9 for coverage and captured all 10 modes. However, coverage and number of captured modes are based only on those samples that the MNIST classifier is confident on. Thus, it does not account for the discrepancy in % high quality samples among the methods. In particular, Unrolled GAN and AdaGAN produced significantly more higher quality samples than WGAN and VEEGAN.

Based on this observation, we computed an adjusted coverage:

Adjusted Coverage $= P$(classifier is confident on sample)P(coverage among confident samples) $+ P$(classifier is not confident on sample)P(coverage among not confident samples)

P(coverage among not confident samples) $= 0$ since samples that the classifier cannot classify are low quality and thus would not cover the true distribution.

Unrolled GAN and AdaGAN scored significantly higher than WGAN, VEEGAN on Adjusted Coverage, which is consistent with expectations, based on the generated samples in Figure 3. While Unrolled GAN scored higher than AdaGAN, this difference may be due to using a less robust classifier in determining these metrics. For example, Unrolled GAN had higher % high quality samples. Yet, comparing the generated samples displayed in Figure 3, results from AdaGAN are much clearer than results from Unrolled GAN. The MNIST classifier may need more training to improve its accuracy.

We were unable to compute the birthday paradox for this dataset on WGAN, VEEGAN and Unrolled GAN due to the visually low image quality of produced samples. The low quality made it difficult for a human observer to visually asses duplicates. We were able to compute it for AdaGAN, and found a support size of 225, which is exactly the same as what we computed for the true distribution.

Based on these metrics, Unrolled GAN and AdaGAN are the top performing methods on the MNIST dataset.

IV. Discussion

In this work we conducted experiments comparing the performance of several state-of-art GANs, in terms of mode collapse. We experimented with datasets with different distributions, both real and synthetic. We evaluated the GANs via several commonly used metrics. Wasserstein distance and coverage metrics measure the deviation of the generated samples from the real samples. Number of captured modes, % high quality samples, and birthday paradox are concerned with how many modes are captured and how well the GANs capture and recover the modes. To conduct fair comparison on the GAN methods, we implemented the same GAN architecture and training settings across the methods, even though these might not be the optimal settings for one specific method.

GAN Methods: Our results show that AdaGAN performs consistently well on nearly all datasets and under several different metrics. This may be because AdaGAN learns a mixture of GANs, where each GAN is trained to improve the accuracy of the current mixture as demonstrated in Figure 4.

For synthetic datasets, VEEGAN had better performance than Unrolled GAN and WGAN but performed poorly on MNIST. A possible explanation could be that the reconstructor network used, which was set as fully connected layers, does not work well on more complicated datasets like MNIST. A more carefully constructed and pretrained reconstructor might help to improve the performance of VEEGAN. WGAN performed consistently poorly across the datasets. Unrolled GAN performed well on the 2D ring dataset and MNIST, but did not perform well on the 2D grid dataset, and performed quite poorly on the high dimensional dataset.

For all the GAN variants we used, poor performance could be attributed to training parameters and network architectures that are suboptimal for a particular GAN variant on a particular type of data. While the experiments described above shed some light on the strengths and weaknesses of these GAN variants, experimenting with different training parameters and network architectures could further clarify the robustness of these GANs to mode collapse.

Evaluation Methods: Our results demonstrate that one metric is not sufficient to measure mode collapse for GANs. We observe that the metrics do not necessarily give consistent results. For example, on MNIST the number of detected modes and coverage scores for WGAN, Unrolled GAN, and AdaGAN were high, indicating good performance. However, the proportion of high quality samples was low, which motivated computing the Adjusted Coverage metric. Therefore, it is important to use several different metrics to evaluate the extent of mode collapse in generated samples.

Acknowledgment

This project was conducted as the final project for the MIT class Science of Deep Learning: Bridging Theory and Practice (6.883) taught in Spring 2018. The authors thank Professor Aleksander Madry and Professor Constantinos Daskalakis for their instruction and support throughout the course. The authors also thank graduate students Dimitris Tsipras and Emmanouil Zampetakis the course teaching assistants for their guidance.

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

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

[3] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016.

[4] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015.

[5] Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*, 2017.

[6] T Salimans, I Goodfellow, W Zaremba, V Cheung, A Radford, and X Chen. Improved techniques for training gans. nips, 2016. *Yan, Y. Ding, P. Li, Q. Wang, Y. Xu, W. Zuo, Mind the Class Weight Bias: Weighted Maximum Mean Discrepancy for Unsupervised Domain Adaptation, CVPR*, 2017.

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

[8] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[9] Akash Srivastava, Lazar Valkoz, Chris Russell, Michael U Gutmann, and Charles Sutton. Veegan: Reducing mode collapse in gans using implicit variational learning. In *Advances in Neural Information Processing Systems*, pages 3310–3320, 2017.

[10] Ilya O Tolstikhin, Sylvain Gelly, Olivier Bousquet, Carl-Johann Simon-Gabriel, and Bernhard Schölkopf. Adagan: Boosting generative models. In *Advances in Neural Information Processing Systems*, pages 5430–5439, 2017.

[11] G Huang, Y Yuan, Q Xu, C Guo, Y Sun, F Wu, and K Weinberger. An empirical study on evaluation metrics of generative adversarial networks. In *International Conference on Learning Representations.*, rejected.

[12] Ali Borji. Pros and cons of gan evaluation measures. *arXiv preprint arXiv:1802.03446*, 2018.

[13] Sanjeev Arora and Yi Zhang. Do gans actually learn the distribution? an empirical study. *arXiv preprint arXiv:1706.08224*, 2017.

GAN Model Reference Code

1) Wasserstein GAN: https://github.com/cameronfabbri/Wasserstein-GAN-Tensorflow

2) VEEGAN: https://github.com/akashgit/VEEGAN

3) AdaGAN: https://github.com/tolstikhin/adagan

4) Unrolled GAN: https://github.com/poolio/Unrolled_GAN

V. Appendix

A. Tables and Figures

TABLE I: Dataset: 2D Ring

GAN Model	**Metrics**				
	Wasserstein Distance	***Coverage***	***% High quality samples***	***# Modes***	***Birthday Paradox***
WGAN	0.0	0.55	18.9	7	196
VEEGAN	0.0	**1.00**	88.3	**8**	64
Unrolled GAN	0.0	0.91	94.5	**8**	36
AdaGAN	0.0	0.88	**96.9**	**8**	**49**
True Distribution	NA	NA	NA	8	49

TABLE II: Dataset: 2D Grid

GAN Model	**Metrics**				
	Wasserstein Distance	***Coverage***	***% High quality samples***	***# Modes***	***Birthday Paradox***
WGAN	0.0	**1.00**	3.3	**25**	11881
VEEGAN	0.0	0.96	65.5	**25**	289
Unrolled GAN	0.0	0.55	87.5	14	81
AdaGAN	0.0	**1.00**	**89.2**	**25**	**144**
True Distribution	NA	NA	NA	25	144

TABLE III: Dataset: High Dimensional Mixture of Gaussians

GAN Model	**Metrics**			
	Wasserstein Distance	***% High quality samples***	***# Modes***	***Birthday Paradox***
WGAN	1.43	14.6	1	$>$1000000
VEEGAN	**1.15**	**65.5**	6	25
Unrolled GAN	3.23	0	0	$>$1000000
AdaGAN	1.97	39.4	**9**	**441**
True Distribution	NA	NA	9	289

TABLE IV: Dataset: MNIST

GAN Model	**Metrics**				
	Wasserstein Distance	***Coverage***	***Adjusted Coverage***	***% High quality samples***	***# Modes***
WGAN	2.75×10^{-5}	**0.9**	0.12	13	10
VEEGAN	0.0003	0.6	0.09	14	6
Unrolled GAN	0.0007	**0.9**	0.47	**53**	10
AdaGAN	$\mathbf{7.84 \times 10^{-6}}$	**0.9**	0.4	45	10

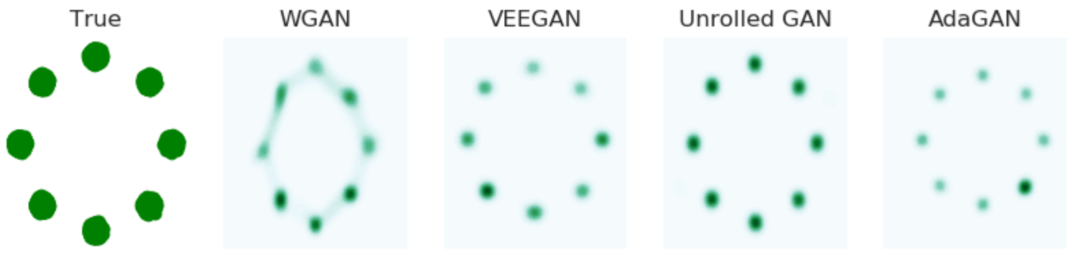


Fig. 1: True distribution and corresponding kernel density estimate of the learned distribution based on the generated samples of WGAN, VEEGAN, Unrolled GAN and AdaGAN from training on 2D ring dataset.

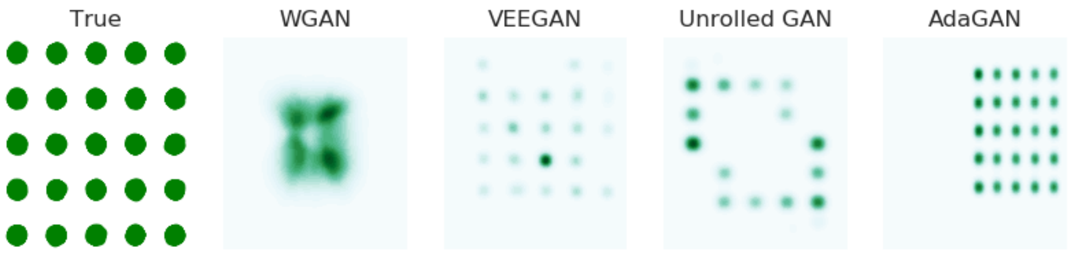
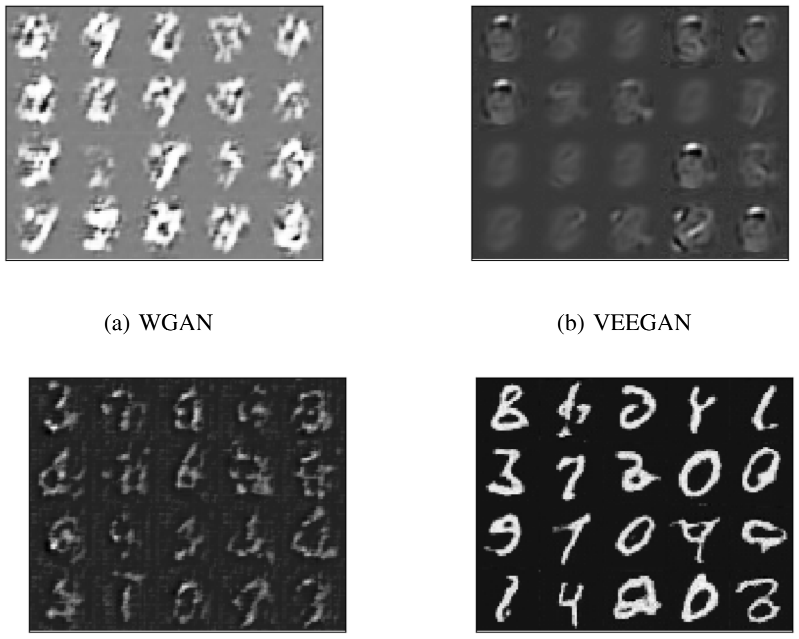


Fig. 2: True distribution and corresponding kernel density estimate of the learned distribution based on the generated samples of WGAN, VEEGAN, Unrolled GAN and AdaGAN from training on 2D grid dataset.



(a) WGAN (b) VEEGAN

(c) Unrolled GAN (d) AdaGAN

Fig. 3: Generated MNIST samples (randomly picked) from WGAN, VEEGAN, Unrolled GAN and AdaGAN.

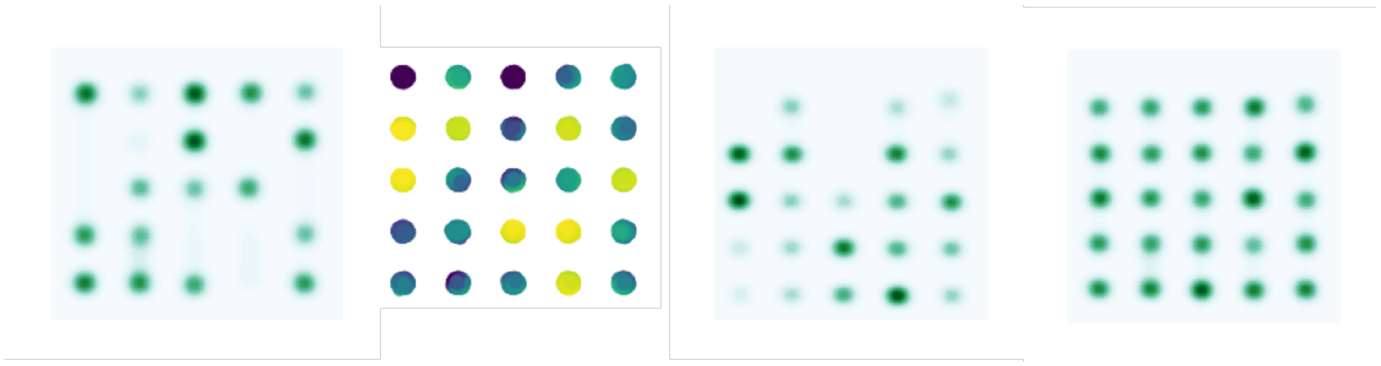


Fig. 4: Example of a step in the AdaGAN algorithm. Left is the samples generated from the mixture at step t-1. In the second figure, the training set is reweighted to capture modes that the mixture model at step t-1 fails to capture. Yellow points are highly weighted, purple points are lowly weighted. In the third figure, the component GAN is trained on this reweighted dataset and generates samples according to this weighted distribution. In the fourth figure, samples from the new mixture model are shown. It captures the 2d grid distribution more closely.