Evaluating GANs using Efficient Real-World Estimate (Project Report - CSC2515)

Vaibhav Saxena
MScAC, DCS
Student No. - 1004824639
University of Toronto
vaibhav@cs.toronto.edu

Pulkit Mathur
MScAC, DCS
Student No. - 1005483692
University of Toronto
pulkit@cs.toronto.edu

Sumeet Ranka
MScAC, DCS
Student No. - 1004945152
University of Toronto
ranka47@cs.toronto.edu

Abstract

Evaluation of generative models is a crucial task, and many metrics today ignore the reverse KL divergence component due to an existing difficulty in obtaining the real-world density estimate p_r from the limited number of samples available. In this work, we propose a MCMC sampling method using Metropolis-Hastings algorithm for obtaining samples from p_r as learned by the discriminator D of a GAN, using samples from the generator G. We improve upon an existing rejection sampling approach with respect to a higher acceptance percentage and less samples rejected around the modes. We show how this can be used to evaluate a metric containing a reverse KLD component, rather than adhering to a log-likelihood metric, which seems unfair for models like GANs which intend to minimize the Jenson-Shannon divergence in their loss function. This method can also be used for monitoring how well the generator has been trained with respect to the real-world estimate of the discriminator.

1 Introduction

Learning a generative model involves learning a close approximation of the real-world distribution p_r. The approximated distribution p_g aims to be close to p_r with respect to a distance metric which basically governs the loss function of the generative model. Generative Adversarial Networks (GANs) [1] try to minimize the Jensen-Shannon Divergence (JSD) between the true data density and model density. Also, evaluation metrics such as log-likelihood estimate are essentially a minimization of the forward KL Divergence (KLD) $KL(p_r||p_g)$. Given the different motivations for a generative model's loss function, it seems unfair to evaluate it using a purely forward KLD based metric.

In this work, we propose a method for obtaining an approximation for the real-world distribution p_r, which can then be used for evaluating the reverse KLD for a GAN. The discriminator of a GAN gives the reinforcement signal to the generator while learning, which tells us that it is possible to gain access to the approximation of p_r which the discriminator has stored. Moreover, it has been shown that a generator with infinite capacity can store the distribution p_r with good accuracy [2]. We use a rejection sampling based method, proposed by Azadi et al. [3], to obtain samples of p_r, using which we estimate the density and use it to evaluate the reverse KLD for the generator. We propose using MCMC sampling (Metropolis-Hastings algorithm) using samples of the generator as a proposal to obtain samples from p_r. In our analysis, MCMC sampling gave a higher acceptance percentage as compared to rejection sampling, and rejected less samples close to the modes of the distribution. Based on the results, it can be inferred that this method can be practically used for monitoring how well the generator has been trained with respect to the real-world estimate of the discriminator.

The rest of the report is organized as follows. In Section 2, we talk about the background of different concepts that were used for this work. In Section 3, we talk about some related works, papers we build upon, and some foundational works which were the basis of the works we relate to. In Section 4, we give a brief description of our interpretation of evaluation in general. Section 5 gives the mathematical

formulation for retrieving the p_r samples using our proposed MCMC technique. Section 6 talks about the limitations of our methodology. Section 7 talks about the experiments we conducted and the results we obtained.

2 Background

2.1 Generative Adversarial Networks

A generative adversarial network (GAN) (Goodfellow et al., 2014) is a framework for learning generative models via an adversarial process. The generator G tries to win an "imitation game" against the discriminator D. D aims to maximize the loss function represented by Eqn. (1), while G tries to minimize the same equation.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z))] \tag{1}$$

Both the players are modeled with neural networks as $G(z, \theta_g)$ and $D(x, \theta_d)$ where θ_g and θ_d represent the weights or parameters of neural nets. This makes the whole framework end-to-end differentiable.

2.2 Mode collapse (A challenge in training GANs):

Arjovsky et al. (2017) [2] proved the following result:

$$\mathbb{E}_{z\sim p(z)}[-\nabla_\theta \log D^*(G_\theta(z))|_{\theta=\theta_0}] = \nabla_\theta[KL(p_{g_\theta}||p_r) - 2JSD(p_{g_\theta}||p_r)]|_{\theta=\theta_0} \tag{2}$$

From (2), we observe that the loss function the generator intends to minimize contains a reverse KLD term, $KL(p_{g_\theta}||p_r)$. This means that the generator has an intrinsic tendency to be conservative in its learning procedure, thus resulting in missing modes in p_r.

2.3 Metropolis-Hastings Algorithm

The Metropolis-Hastings algorithm is a Markov Chain Monte Carlo (MCMC) method for sampling from a high-dimensional distribution $P(x)$. The idea is to define a Markov chain over the possible values of x, in such a way that the stationary distribution of the Markov chain is in fact $P(x)$. In other words, generate a sequence of values x, denoted $(x_0, x_1, x_2, ...)$, such that $x_n \sim P(x)$ as $n \to \infty$.

It is a two stage process for generating x_{n+1} from current state of Markov chain x_n. The first stage is to generate a candidate x^* from the proposal distribution, denoted by $Q(x^*|x_n)$, which depends on the current state of the Markov chain, x_n. The second stage is the accept-reject step where we calculate the acceptance probability $A(x_n \to\ x^*)$, which is given by

$$A(x_n \to\ x^*) = \min\left(1, \frac{P(x^*)}{P(x_n)} * \frac{Q(x_n|x^*)}{Q(x^*|x_n)}\right) \tag{3}$$

Having proposed the candidate x^* and calculated the acceptance probability, $A(x_n \to\ x^*)$, we now decide whether to "accept" the candidate, i.e. $x_{n+1} = x^*$ or "reject" the candidate, i.e. $x_{n+1} = x_n$. For this, we generate a random number, u, between 0 and 1 and then take the decision using the piece-wise function represented by Eqn. (4).

$$x_{n+1} = \begin{cases} x^*, & \text{if } u \le A(x_n \to\ x^*) \\ x_n, & \text{if } u > A(x_n \to\ x^*) \end{cases} \tag{4}$$

2.4 Kernel Density Estimation

Kernel Density Estimation is a technique to estimate the unknown probability distribution of a random variable with the help of kernel, based on a sample of points taken from that distribution. It does so by smoothing out the contribution of each observed data point over a local neighbourhood of that data point. The contribution of data point $x(i)$ to the estimate at some point x^* depends on how far

apart $x(i)$ and x^* are. The extent of this contribution is dependent upon the shape of the adopted kernel function and the width (bandwidth) accorded to it. If we denote the kernel function as K and its bandwidth by h, the estimated density at any point x is given by:

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{X_i - x}{h}\right) \tag{5}$$

3 Related Works

A lot of work has been done in the genre of evaluation of GANs and estimating the data distribution. A recent paper on a rejection sampling mechanism by Azadi et al. [3] talks about recovering the data distribution by using the discriminator of GAN to approximately correct errors in the GAN generator distribution. The proposal was designed keeping in mind the assumptions for GANs that do not hold in practice. The corner cases of these assumptions were examined and they were able to recover the data distribution for a mixture of Gaussians and for the SAGAN model, start-of-the-art in the image generation task.

Another paper by Arjovsky et al. discussed about the steps that need to be taken towards fully understanding the training dynamics of generative adversarial networks [2]. It mainly examined a practical and theoretically grounded direction towards solving instability and saturation that arises during the training of GANs. Based on the analysis by Narayanan et al. [4], they proved that p_g is also extremely concentrated on a low-dimensional manifold. They continued to argue on the basis that if the supports of p_r and p_g are disjoint or lie in low-dimensional manifolds then it can lead to an unreliable training of the generator.

A survey paper by Borji [5] compares various GAN evaluation measures. In addition, it provides a set of 7 desiderata and evaluates whether the discussed measures are compatible with them. It mentions that a good evaluation measure should have the following properties: discriminability, ability to detect overfitting, disentangled latent spaces, well-defined bounds, perceptual judgements, sensitive to distortions, and low sample and computational complexity. The main finding was that it is unlikely that a single score can cover all the aspects. However, measures like FID (Frechet Inception Distance) score seemed more plausible than the others.

Annealed Importance Sampling (AIS) has also been proposed to evaluate log-likelihood for decoder-based models [6]. The idea was to analyze the performance of these models, the effectiveness of existing log-likelihood estimators, the degree of overfitting, and the extent to which these models miss the important modes of the distribution.

The related papers on the evaluation of GANs and the estimation of the true data distribution have been based on the strong foundations of the previous research work. The papers [2, 5] based their arguments against the traditional maximum likelihood approach from the findings of the Huszar, 2015 [7]. It mentions that the maximum likelihood approach takes us away from the objective of generating natural-looking samples. Apart from that, the work by Narayanan et al. [4], cited by Arjovsky et al. in [2], has been used for our results as well.

AIS [8] was the key in the findings of the paper Wu et al. [6]. They showed that AIS was able to perform better than Kernel Density Estimation (KDE) [9], a widely used estimator of log-likelihood for GANs. However, due to the inability of the KDE to estimate likelihood in high dimensions [10], Wu et al. proposed for AIS instead of KDE. The models that were considered in the paper showed that AIS was more accurate than KDE by an order of magnitude of two. They also showed that the accuracy was enough to perform fine-grained comparisons between generative models.

4 The Student-Teacher Testing framework

Briefly analyzing the forward and reverse KLDs qualitatively - forward KLD penalizes the learned probability distribution if it learns to give a zero probability density to a point where the real distribution has a positive density, whereas, reverse KLD penalizes the learned probability distribution if it learns to give a positive probability density to a region where the real distribution has zero density.

We now talk about a general framework for evaluation of any generative model. Drawing analogy from the prevailing education systems, we observe that evaluation is usually done in one of the two

ways: the teacher creates a question paper comprising of test cases from multiple domains and the student is either expected to know the answer to all of the questions, or the student is asked to present a topic he knows well to the teacher. In the second scenario, the teacher will judge the student on the basis of his knowledge in that particular domain. The first scenario is similar to minimizing the forward KLD (a.k.a. likelihood) between the student's and the teacher's knowledge distributions, while the second scenario is same as minimizing the reverse KLD between the same two distributions.

5 Obtaining the real-world (p_r) estimate

Azadi et al. (2018) proposed a method for obtaining the ratio of densities $p_r(x)$ and $p_g(x)$ for a given sample x, from the optimal discriminator D^*. Assume that the output of the discriminator is a sigmoid over the logits $\tilde{D}$, as given by

$$D(x) = \frac{1}{1 + e^{-\tilde{D}(x)}} \tag{6}$$

Also, the optimal discriminator D^*, which minimizes the loss for a particular distribution p_g of the generator G, takes the form as

$$D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)} \tag{7}$$

Using (6) and (7), we have the following derivation:

$$\begin{aligned} D^*(x) = \frac{1}{1 + e^{-\tilde{D}^*(x)}} &= \frac{p_r(x)}{p_r(x) + p_g(x)} \\ \implies 1 + e^{-\tilde{D}^*(x)} &= \frac{p_r(x) + p_g(x)}{p_r(x)} \\ \implies p_r(x) e^{-\tilde{D}^*(x)} &= p_g(x) \\ \implies \frac{p_r(x)}{p_g(x)} &= e^{\tilde{D}^*(x)} \end{aligned} \tag{8}$$

Therefore, for a fixed generator distribution p_g, if we have the optimal discriminator D^* then we can obtain the ratio of p_r and p_g using (8).

Proposed method for obtaining p_r samples:

We use the result obtained in (8) inside the Metropolis-Hastings algorithm (a MCMC approach) to obtain samples from p_r using samples from p_g, i.e. samples from the generator, as the proposals. The full procedure is given in Algorithm 1.

We compare the samples obtained by using Metropolis-Hastings algorithm, and rejection sampling, as proposed by Azadi et al. We observed that Metropolis-Hastings gave around 10% higher acceptance percentage than rejection sampling, and was able to reject samples from p_g which were actually not present in the real data distribution.

After obtaining samples from p_r and p_g, we estimate their densities using the Kernel Density Estimator (explained in Section 2.4).

6 Limitations

The formulation given in Section 5 is based upon various assumptions and corresponding limitations.

1. **Inability to find the optimal discriminator D^*:** We stated a method to retrieve the samples of p_r using samples from G and the logits of the optimal discriminator D^*. However, in practice D^* cannot be computed. Therefore our acceptance probability of a sample from G will not be accurate. However, in our experiments we make sure that the discriminator and generator are trained till convergence, with discriminator outputting 0.5 for samples from p_g at convergence. This keeps the error in acceptance probability within bounds.

Algorithm 1 MCMC-based p_r retrieval

Input: Trained generator G, discriminator D
max_trials $\leftarrow 5000$
count_rejections $\leftarrow 0$ ▷ to count number of samples rejected by this algorithm
samples $\leftarrow [\,]$ ▷ array for storing p_r samples
Sample $\mathbf{x}$ from generator G
prev_ratio $\leftarrow e^{\tilde{D}(\mathbf{x})}$ ▷ $\tilde{D}(\mathbf{x})$ are logits of D at $\mathbf{x}$, as in (8)
for i = 1 $\rightarrow$ max_trials **do**
    Sample $\mathbf{x}$ from generator G
    curr_ratio $\leftarrow e^{\tilde{D}(\mathbf{x})}$ ▷ as in (8)
    acceptance_prob $\leftarrow \min\left(1, \frac{\text{curr\_ratio}}{\text{prev\_ratio}}\right)$
    Sample $u \sim$ Unif(0,1)
    if $u <$ acceptance_prob **then**
        Append $\mathbf{x}$ to samples
    else
        count_rejections $\leftarrow$ count_rejections $+1$
percent_acceptance $\leftarrow (1 - \frac{\text{count\_rejections}}{\text{max\_trials}}) * 100$
return samples, percent_acceptance

Algorithm 2 Generate samples from mixture of two Gaussians

Initialize: $\mu_1, \Sigma_1, \mu_2, \Sigma_2$ ▷ mean vectors and covariance matrices of Gaussians
$\pi_1 \leftarrow 0.5, \pi_2 \leftarrow 1 - \pi_1$ ▷ probability of sampling from each Gaussian
num_samples $\leftarrow 4000$
samples $\leftarrow [\,]$
for i = 1 $\rightarrow$ num_samples **do**
    Sample $u \sim$ Unif(0,1)
    if $u < \pi_1$ **then**
        Sample $s \sim$ Norm(μ_1, Σ_1)
    else
        Sample $s \sim$ Norm(μ_2, Σ_2)
    Append s to samples
return samples

2. **Finite capacity of function approximator for D:** In our experiments, we used a neural network to learn the approximation for D. As shown in [4] and backed by [3], a neural network approximation of a high-dimensional function lies on a low-dimensional manifold of the entire space. We observe in our experiments that the discriminator and generator learn a low-dimensional manifold of p_r, which urges the need of a smoothing density estimator for both p_r and p_g if we want to use them for any kind of evaluation.

Azadi et al. [3] stated a limitation that it is intractable to compute the maximum ratio (M) among all p_r/p_g for normalization of the acceptance probability. However, that limitation is eliminated when using MCMC sampling.

7 Experiments and Results

For our experiments, we generated a bivariate mixture of Gaussians dataset with two modes, as illustrated in Fig. 1. The entire dataset was normalized between 0 and 1, and consisted of 4000 samples. We used the pseudo-code in Algorithm 2 to generate data from a mixture of Gaussians.

Both the discriminator and generator of our GAN contained a single hidden layer with 30 units and ReLU activation. The discriminator had a single output unit with a sigmoid activation, while the generator had 2 output units with identity activation. We illustrate our model in Fig. 2. Formally, our model can be written as Eqn. (9).

$$\begin{aligned} D(\mathbf{x}) &= \sigma(\mathbf{W_2^D} f(\mathbf{W_1^D x})) \\ G(\mathbf{z}) &= \mathbf{W_2^G} f(\mathbf{W_1^G z}) \end{aligned} \tag{9}$$

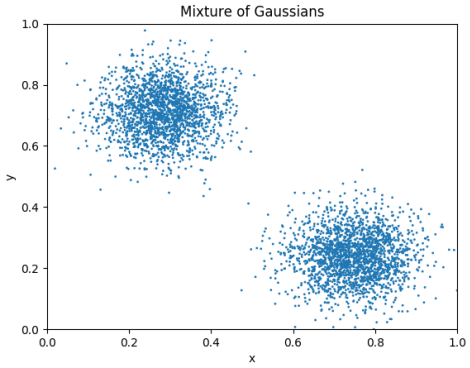


Figure 1: Dataset of a bivariate mixture of Gaussians.

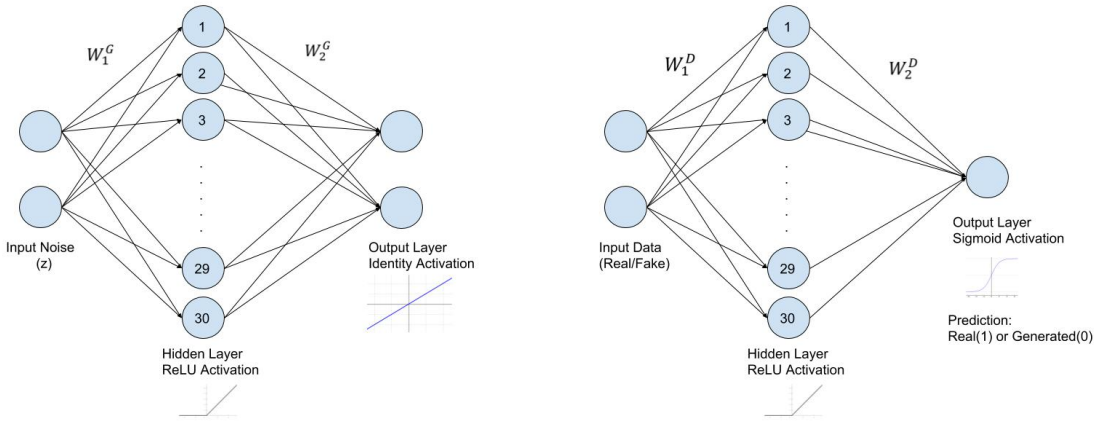


Figure 2: (Left) Generator Architecture and (Right) Discriminator Architecture

where f is the rectified linear activation function, and σ is the sigmoid activation function. The prior inputs $\mathbf{z}$ to the generator were samples from a bivariate normal distribution with mean $[0.5, 0.5]$ and covariance matrix $0.01\mathbf{I}_2$. The loss functions we used for our GAN implementation are as in (10).

$$\begin{aligned} \min_D L(D) &= -\frac{1}{N}\sum_{i=1}^{N}\left[\log D(\mathbf{x}_i)|_{\mathbf{x}_i\sim p_r(\mathbf{x})} - \log(1 - D(G(\mathbf{z}_i))|_{\mathbf{z}_i\sim p_{\mathbf{z}}(\mathbf{z})}\right] \\ \min_G L(G) &= -\frac{1}{N}\sum_{i=1}^{N}\left[\log D(G(\mathbf{z}_i)|_{\mathbf{z}_i\sim p_{\mathbf{z}}(\mathbf{z})}\right] \end{aligned} \tag{10}$$

We tried multiple initialization schemes for the weights of the networks, including normal random, uniform random, Xavier initialization, and truncated normal. **Truncated normal initialization for weights** with mean 0 and standard deviation 2.0 worked best for our setup. We also **pre-trained the discriminator** using just the data distribution with a squared-loss objective, as given in (11). The probability $P_r(\mathbf{x}) \in [0, 1]$ was estimated for a set of discrete points on the domain by fitting a histogram over the available data.

$$\min_D L_{\text{pretrain}}(D) = \sum_{\mathbf{x}\in\mathbb{R}^2} ||D(\mathbf{x}) - P_r(\mathbf{x})||^2 \tag{11}$$

We used the Adam optimization algorithm with learning rate 0.001 for pre-training as well as adversarial learning.

7.1 Results

After training our GAN, we obtained samples from the generator G which are illustrated in Fig. 3. Observing the samples we realize that the generator was indeed able to capture the modes in the data distribution to a certain extent. The modes represented by straight lines in the samples is an indication that the generator learned the data distribution on a low-dimensional manifold rather than the original space. This is mainly because it has a very limited capacity as a neural network. There is empirical evidence that the approximation of p_r lies on a low-dimensional manifold (Narayanan & Mitter, 2010) [4]. Arjovsky et al. [2] have proven the same for the approximation of p_g using a neural network as well.

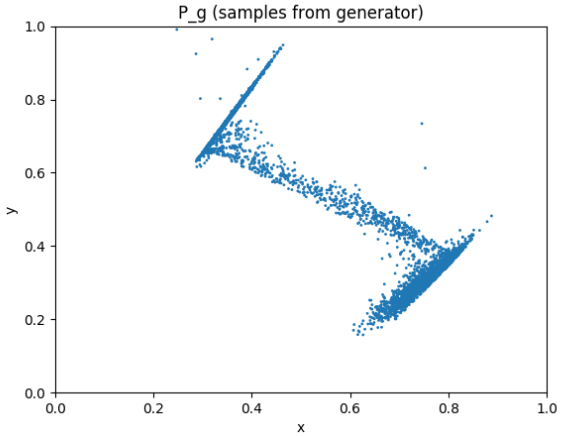


Figure 3: Samples from the trained generator G.

p_r retrieval using Rejection Sampling

We also implemented rejection sampling, as described in [3], to obtain p_r samples from samples of p_g. The samples retrieved and the samples rejected have been illustrated in Fig. 4. It is interesting to see how samples from p_g that were not near the modes in the real data distribution were rejected in this sampling.

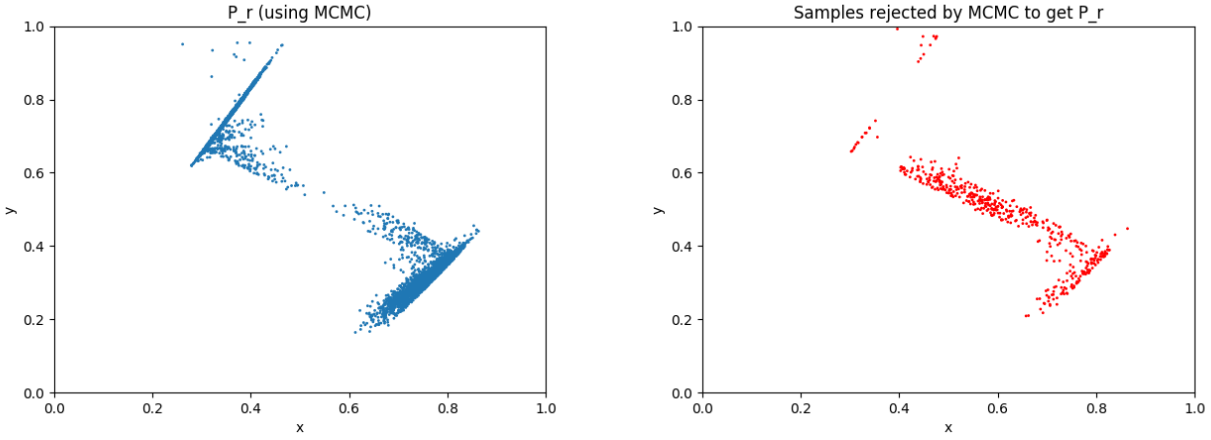
(a) Accepted samples for p_r.

(b) Rejected samples for p_r.

Figure 4: p_r retrieval (from p_g) using Rejection Sampling.

p_r retrieval using MCMC Sampling (Metropolis-Hastings algorithm)

We retrieved p_r samples from the p_g samples using Metropolis-Hastings algorithm, as described in Algorithm 1. We illustrate the samples retrieved as well as the samples rejected in Fig. 5.



(a) Accepted samples for p_r.

(b) Rejected samples for p_r.

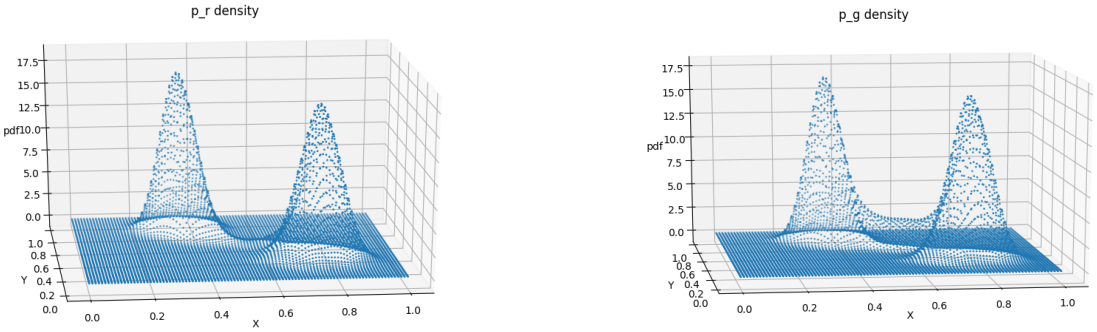
Figure 5: p_r retrieval (from p_g) using MCMC Sampling (Metropolis-Hastings).

Comparing p_r retrieval methods: MCMC (Metropolis-Hastings) vs Rejection Sampling

MCMC sampling rejected 501 out of the 5000 samples generated from p_g, while rejection sampling rejected 947 samples. So MCMC achieved an acceptance percentage of around 90%, while rejection sampling had an acceptance percentage of around 80%. To understand that MCMC did not actually reject samples around the modes, while rejection sampling did, we compare plots 5b and 4b. We observe that the number of samples near the modes that were rejected by rejection sampling is much higher than that by MCMC. Hence, MCMC gives better samples of p_r learned by the discriminator, and can be used further for density estimation of p_r.

Density Estimation

We used Kernel Density Estimator with a Gaussian kernel to obtain the probability density function of p_g and p_r, retrieved using MCMC sampling. The probability density estimates are illustrated in Fig 6. The smoothing effects of the Gaussian kernel are important, because the samples we obtained for p_g and p_r lied on a low-dimensional manifold of the 2-D space, and the Gaussian kernel eliminated that limitation to some extent.



(a) Probability density p_r.

(b) Probability density p_g.

Figure 6: p_r and p_g density functions obtained using KDE with Gaussian kernel.

Evaluation of metric involving reverse KL component

Now that we have the probability density of p_r that the discriminator D has learnt, we can obtain the probability given to a sample generated by generator G by the real data distribution, in the perspective of the discriminator. Thus, any metric containing the expectation $\mathbb{E}_{x \sim p_g} \log(p_r(x))$ term can now be evaluated. This gives a more accurate judgment of the samples generated by G in comparison to the real data distribution learnt by D.

8 Conclusion

In this work, we proposed a MCMC sampling method using Metropolis-Hastings algorithm for obtaining samples from p_r using samples from the generator G. We improved upon the rejection sampling approach given in [3] with respect to a higher acceptance percentage and less samples rejected around the modes. Using MCMC removed the need to find a normalizing factor for the acceptance probabilities. We also looked at some of the limitations and assumptions used in our approach. Finally, we estimated densities p_r and p_g using KDE with a Gaussian kernel stating how it can be used to evaluate a metric containing a reverse KLD component, rather than adhering to a log-likelihood metric, which seems unfair for models like GANs which intend to minimize the Jenson-Shannon divergence. In practice, this method can be used for monitoring how well the generator has been trained with respect to the real-world estimate of the discriminator.

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

[3] Samaneh Azadi, Catherine Olsson, Trevor Darrell, Ian Goodfellow, and Augustus Odena. Discriminator rejection sampling. *arXiv preprint arXiv:1810.06758*, 2018.

[4] Hariharan Narayanan and Sanjoy Mitter. Sample complexity of testing the manifold hypothesis. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1786–1794. Curran Associates, Inc., 2010.

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

[6] Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger Grosse. On the quantitative analysis of decoder-based generative models. *arXiv preprint arXiv:1611.04273*, 2016.

[7] Ferenc Huszár. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv preprint arXiv:1511.05101*, 2015.

[8] Radford M Neal. Annealed importance sampling. *Statistics and computing*, 11(2):125–139, 2001.

[9] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[10] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.