PSL UNIVERSITÉ PARIS

Assignment 2
GAN improvements by rejection sampling methods

Team : PixelGEN

November 12, 2024

Authors :
Mohamed Amine Belkasmi, Emmarius DELAR, Rithy Sochet

1 Introduction

A natural intuition to improve the performance of GAN's generator sampling is to only keep its better produce. This intuition lead us to choose rejection sampling methods to improve our Vanilla GAN. There exist different criterion to decide if a draw obtained by a pre-trained generator is relevant or not, and generally this criterion is based on the discriminator associated to the generator. For this task, we retain two methods, Discriminator Rejection Sampling (DRS) [1] and Metropolis-Hastings GAN (MH-GAN) [2]. DRS and MH-GAN improve the quality of the final generated sample by selecting only the better draws X' from G.
However, we make the assumption that this kind of methods samples only what G know draws better. For example, if our pre-trained generator G is an expert at drawing 0, 1 and 9, then our sample will have a bigger proportion of 0, 1 and 9 than others number. To anticipate this, we also are interested in rejection sampling method integrated during the training of the generator and discriminator. We focused on a single method called Optimal Budgeted Rejection Sampling (OBRS) [3], which would allow to draw various number with a better precision than the classical rejection sampling methods applied at the generation step.

2 What we implemented

First with the files `train.py` and `utils.py` we trained the Vanilla GAN.

We then implemented the Metropolis-Hastings GAN (MH-GAN) (Files: `train_MHGAN.py`, `utils_MHGAN.py`).

We finally implemented the Optimal Budgeted Rejection Sampling (OBRS). (Files: `train_with_OBRS.py` and `utils_OBRS.py`)

We used the pytorch_fid package to measure the results of our experiments and we also used precision and recall package (File: `improved_precision_recall.py`). The `convert.py` file was used to get the MNIST data in `png` format.

3 Rejection Sampling

We first trained a discriminator and a generator of a Vanilla GAN. To go through this study, we must suppose that the discriminator is optimal, so that:

$$D(x) = \frac{p_d(x)}{p_d(x) + p_g(x)}$$

Where $p_d(x)$ is the target distribution and $p_g(x)$ is the generator's distribution.

We can then deduce the acceptation rate as follows:

$$\frac{p_d(x)}{p_g(x)} = \frac{D(x)}{1 - D(x)}$$

In practice, as seen in the paper, we use a function F that verifies:

$$\frac{1}{1+e^{-F(x)}} = e^{D_e^*(x)-D_{e}^*{}_M}$$

Where $M = e^{D_M^*}$ and $D^*(x)$ is the last layer of the discriminator.

We solve the equation to find :

$$F(x) = D_e^*(x) - D_e^*{}_M - \log\left(1 - e^{D_e^*(x)-D_e^*{}_M}\right)$$

We add an hyperparameter γ and a factor ϵ in practice so that:

$$\hat{F}(x) = D_e^*(x) - D_e^*{}_M - \log\left(1 - e^{D_e^*(x)-D_e^*{}_M-\epsilon}\right) - \gamma$$

All in all, during the generation we only accept samples x such that $F(x) \geq \psi$ given the threshold ψ.

With this new function defined, we can train more refinely the GAN with respect to the threshold inequality.

4 Metropolis-Hastings GAN

Similarly to the DRS, with the MH-GAN we used a pre-trained generator G and discriminator $\tilde{D}$. But we are also adding a calibrator C which is a classifier whose role is to "warp" the probability of $\tilde{D}$ such that the final discriminator for our model is: $D(x) = C(\tilde{D}(x))$. The method is using Markov Chain Monte Carlo (MCMC) to obtain the samples to generate, during the algorithm we will sample K images from the generator and we will accept the new sample x_k with the acceptance probability:

$$\alpha(x_k, x) = \min(1, \frac{D(x_k)^{-1} - 1}{D(x)^{-1} - 1})$$

The principle for the algorithm to generate one sample is as follow:

- We start with $x = x_0$ a real image from the real data.
- We generate $x_k = G(z)$.
- We get U from a Uniform(0,1).
- If $U \leq \alpha(x_k, x)$ then $x = x_k$.
- After K samples if x is still a real image then we restart a chain with a generated sample for x_0.
- We return x.

This methods needs a lot of computation power and time, it is not adapted if the goal is to generate a large number of sample. However if the goal is to generate "good" images then MH-GAN is adapted.

Contrary to the DRS method, we do not need to assume that the pre-trained discriminator $\tilde{D}$ is optimal, one close to the optimal is enough. This is due to the calibrator C whose role is to "warp" the probability which is used for $\alpha(x_k, x)$. It is important for the MCMC procedure as an uncalibrated discriminator may output wrong density ratios.

Among the possible experiments to improve the results we have:

- The calibrator C: we used a linear classifier using the logistic regression. The paper also proposes to use the beta or isotonic regression.
- The budget value K: a large number on K would allow us to sample more element from G and avoid restarting a new chain. However a large number of K will also increase the computation time which may not be desirable.

We tested two values of K with $K = 10$ and $K = 100$, we did not observe a major difference in result except for the computation time. We also tried not using the calibrator. The results can be seen in Table 1.

5 Optimal Budgeted Rejection Sampling

Unlike DRS or MH-GAN, OBRS is applied during the training phase of the GAN. Starting with the initial Vanilla GAN architecture, we use optimal budgeted rejection sampling to train the model. Only the training process is modified; however, to ensure comparability, we train it with the same hyperparameters (100 epochs, learning rate 2.10^{-4}, batch size of 64, and 10,000 samples).

5.1 Principle of OBRS

Similarly to DRS and MH GAN, this method also based on the discriminato, D, output. The objectif of OBRS is to train the generator G to minimize, in our case, the Kullback-Leibler divergence (but it can be applied to any divergence function) between p (the target distribution) and p_G the mapping density from G. To minimize that quantity, we find an optimal acceptance function $a_O(x)$ (for a given budget K) defined as :

$$a_O(x) = \min\left(\frac{p(x)}{\hat{p}(x)}\frac{c_K}{M}, 1\right)$$

which is computed using the function **optimal_a_function** in the file **utils_OBRS.py**

We also have:

- c_K is computed using the function **compute_ck** in the file **utils_OBRS.py**, which implements the dichotomy algorithm proposed by [3].
- $M = \sup_{x \in X} \frac{p(x)}{\hat{p}(x)}$ is computed using the function **GAN_opt_likelihood** in the file **utils_OBRS.py**.

First, we update the gradient of D as in a classical GAN by maximizing the expression

$$E_{x \sim P}[D(x)] - E_{x \sim P_G}\left[f^*(T(x))\right],$$

where $f^*(u) = \exp(u-1)$, the conjugate of the Kullback-Leibler (KL) divergence function. Then, updating c_K allows us to calculate a_O, which in turn enables updating G by minimizing

$$E_{x\sim P_G}\left[Ka_O(x)f\left(\frac{r(x)}{Ka_O(x)}\right)\right].$$

where

$$r^{opt}(x) = \nabla f^*\left(D_{\text{opt}}(x)\right) = \frac{p(x)}{\hat{p}(x)}.$$

and repeat this mecanism for each epoch.

5.2 Arbitrary Choices During the Implementation of OBRS

Training a GAN with OBRS depends on a few hyperparameters, such as K, the budget, which we set to 2.6. To learn c_k, we use a threshold of $1e^{-4}$ with an initial value $c_0 = \frac{c_{\min}+c_{\max}}{2}$, where $c_{\min} = 1e^{-10}$ and $c_{\max} = 1e^{10}$.

6 Results of experiments and on the test platform

To ensure a coherent comparison between the Vanilla GAN, the Vanilla GAN with DRS, the Vanilla GAN with MH sampling, and the Vanilla GAN trained with OBRS, we use a single Vanilla GAN architecture, trained on the same training and test datasets, with the same batch size, and number of epochs.

To evaluate our methods, we first compute the FID[1], Precision, and Recall[2] locally. The results are presented in Table 1.

Model	FID	Precision	Recall	Time
Baseline: Vanilla GAN	27.62	0.328	0.176	33 s
Vanilla GAN + MH ($K = 10$)	25.42	0.324	0.177	942 s
Vanilla GAN + MH No calibration ($K = 10$)	23.83	0.330	0.185	536 s
Vanilla GAN + MH ($K = 100$)	25.43	0.322	0.196	6201 s
Vanilla GAN + MH ($K = 100$)	**24.77**	0.332	0.190	8243 s
Vanilla GAN Tw/ OBRS	27.4	**0.352**	**0.21**	**33.52 s**

Table 1: Results with our own testing

According to these results, we can see that the Vanilla GAN with T/w OBRS provides the best improvement in both precision and recall. Additionally, incorporating rejection sampling allows for better performance than simply generating samples with standard rejection sampling methods. Furthermore, generating samples from the Vanilla GAN with T/w OBRS is significantly faster than using the Vanilla GAN with MH. Moreover, the Vanilla GAN with MH achieves very similar, and sometimes even better, results with a smaller budget of K = 10 compared to K = 100 which also greatly increases the computation time.

Another way to evaluate our model is through the testing platform[3], with a summary of the results presented in Table 2. In contrast to Table 1, these results show that the Vanilla GAN with

[1]`https://github.com/mseitzer/pytorch-fid`

[2]`https://github.com/youngjung/improved-precision-and-recall-metric-pytorch`

[3]`https://www.lamsade.dauphine.fr/~testplatform/prds-a2/results/last.html`

Model	FID	Precision	Recall	Time
Baseline: Vanilla GAN	26.86	**0.53**	0.24	90
DRS	30.04	0.52	**0.26**	106
Vanilla + MH-GAN (K=10)	**25.36**	**0.53**	0.22	1044
Vanilla GAN Tw/ OBRS	27.69	0.51	0.2	101

Table 2: Results on testing platform

T/w OBRS underperforms and yields the worst results. None of our methods achieved the best performance across all metrics; however, the Vanilla GAN with MH sampling at a budget K=10 achieved the highest Precision (equal to the Vanilla GAN) and the best FID score. On the other hand, the Vanilla GAN with DRS showed significantly better Recall.

References

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

[2] Ryan Turner, Jane Hung, Eric Frank, Yunus Saatchi, and Jason Yosinski. Metropolis-hastings generative adversarial networks. pages 6345–6353, 2019.

[3] Alexandre Verine, Muni Sreenivas Pydi, Benjamin Negrevergne, and Yann Chevaleyre. Optimal budgeted rejection sampling for generative models. pages 3367–3375, 2024.