f-Divergences and latent re-weighting on GAN

Rafael BENATTI, Yassine KALAA

nGANnou

Abstract

This report outlines our extensive exploration of integrating f-divergences and latent reweighting within a Generative Adversarial Network (GAN) [1]. Our main goal is to examine how experimental adjustments affect image generation. Emphasizing empirical analysis and the pursuit of enhanced model performance, our focus centers on methodological insights, the particular modifications implemented, and the resulting outcomes detailed throughout this report.

I Introduction

Initially, our exploration involved implementing a basic GAN architecture, serving as a foundational benchmark for subsequent model comparisons[1]. The primary objective of deploying GANs revolves around their ability to generate synthetic data by learning the underlying data distribution.

The fundamental principle of GANs involves two neural networks, a generator (G) and a discriminator (D), engaged in a minimax game. The generator aims to produce data samples that resemble the real data distribution, while the discriminator strives to differentiate between real and generated samples [2].

Mathematically, the GAN training process can be described as optimizing the following objective function [2]:

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}(\boldsymbol{x})}} [\log D(\boldsymbol{x})] + \mathbb{E}\boldsymbol{z} \sim p\boldsymbol{z}(\boldsymbol{z}) [\log(1 - D(G(\boldsymbol{z})))] \tag{1}$$

Our initial GAN implementation focuses on minimizing this adversarial loss by updating the generator and discriminator networks iteratively.

The evaluation of the GAN model's performance is demonstrated in Figure 1:

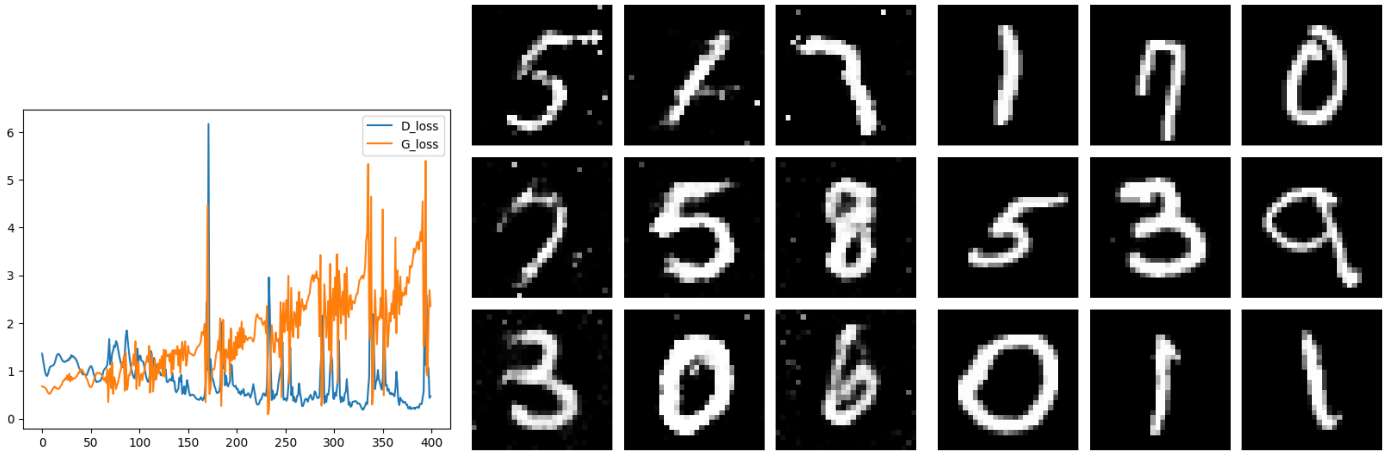


Figure 1: GAN Training Metrics

In Figure 1, we showcase the progress of the GAN training process. The left plot depicts the adversarial loss convergence over epochs, demonstrating the learning dynamics of the GAN. The middle and right plots display the generated samples and real data samples, respectively, revealing the model's ability to capture the underlying data distribution.

Our initial evaluation indicates that the GAN model has started to capture certain characteristics of the data distribution, as evidenced by the visual comparisons between generated and real samples. However, further optimization and experimentation are required to enhance the model's performance, particularly in generating high-quality samples across various data categories.

[1]The code for this approach can be found in the "GAN.py" file

I.1 Model Evaluation Metrics

The adversarial loss convergence and visual inspection of generated samples against real samples serve as primary indicators of the model's progress and fidelity in replicating the data distribution. Also, we continuosly calculated the FID (Frechet Inception Distance) of out generated images. The FID measures the similarity between the statistics of generated images and real images by comparing feature representations extracted from a pre-trained deep neural network, it has a good relation with human perception. The lower the FID the best. We also had available a platform to calculate the precision and recall of generated images (which was more complex to develop the code) but since we would have a limitation of evaluating one model per day at maximum, we decided to
Moreover, to mitigate overfitting, we adopt a data splitting strategy, dividing the dataset into train and test sets during each training session.

II f-GAN

II.1 Introduction to f-Divergences for f-GAN Development

f-divergences represent a versatile class of measures crucial in quantifying dissimilarities between probability distributions. These divergences, characterized by their convex function f, serve as powerful tools for assessing the discrepancy between two distributions. The inherent flexibility of f-divergences allows for comparisons between distributions across diverse domains, making them instrumental in capturing the nuanced differences in probability distributions.
Mathematically, f-divergences possess key properties that define their behavior. They are non-negative for all pairs of probability distributions and become zero only when the distributions are identical. However, they lack symmetry in general. Additionally, when f-divergences tend to zero, it implies that the two distributions are converging.
The f-GAN idea is to lower bound the distance between the true and generated distribution by using:

$$D_f(P\|Q) \geq \sup_{T\in\mathcal{T}} (\mathbb{E}_{x\sim P}[T(x)] - \mathbb{E}_{x\sim Q}[f^*(T(x))])$$

To use the variational objective for various f-divergences, we must adhere to the domain of the conjugate functions f^*. To achieve this, we assume that the variational function T_ω takes the form $T_\omega(x) = g_f(V_\omega(x))$ and modify the saddle objective accordingly [1]:

$$F(\theta,\omega) = \mathbb{E}_{x\sim P}[g_f(V_\omega(x))] + \mathbb{E}_{x\sim Q_\theta}[-f^*(g_f(V_\omega(x)))]$$

.
The idea of it is actually quite simple. We'll lower bound the distance between distributions (calculated with a f-divergence) with empirical means. Further we will ensure our outputs are in the domain we need.

II.2 Implementation

To implement the concept of f-divergences in GAN we will take our discriminator as variational function V_ω and use g_f as activation function. Further, we will also use the Frechet conjugate in the loss function. The f-divergences and their respectives activation functions an Frechet conjugates can be seen in the table below.

Table 1: Name, Output activation g_f, , Conjugate $f^*(t)$, $f'(1)$

Name	Output activation g_f	Conjugate $f^*(t)$	$f'(1)$
Kullback-Leibler (KL)	v	$\exp(t-1)$	1
Reverse KL	$-\exp(-v)$	$-1-\log(-t)$	-1
Pearson χ^2	v	$\frac{1}{4}t^2+t$	0
Squared Hellinger	$1-\exp(-v)$	$\frac{t}{1-t}$	0
Jensen-Shannon	$\log(2)-\log(1+\exp(-v))$	$-\log(2-\exp(t))$	0
GAN	$-\log(1+\exp(-v))$	$-\log(1-\exp(t))$	$-\log(2)$

$f'(1)$ can be interpreted as a threshold for the discriminator to tell whether a image is true or not.

The f-GAN paper provides a plot of each activation function g_f To ensure our implementation was correct we plotted our implemented functions as function of v.

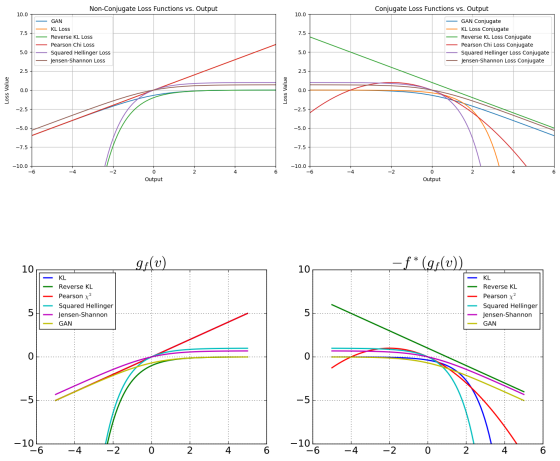
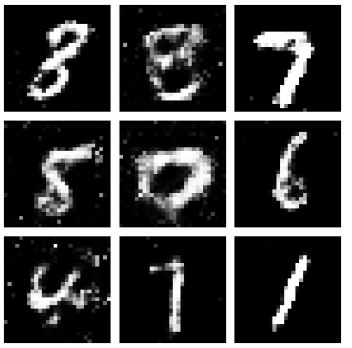
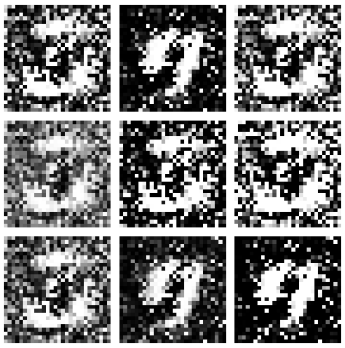


Figure 2: Images generated usign the GAN divergence.

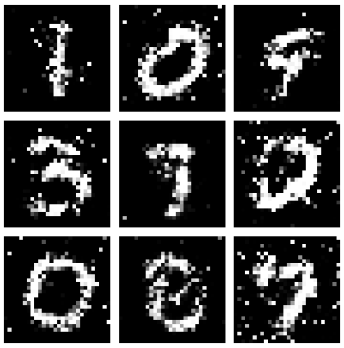
Still, to ensure one more time our code wasn't making any mistake in the method, we started by the implementation of the GAN divergence, which is equivalent to the original GAN loss function, which is equivalent to binary cross entropy. So we should have the same results. In the image below we can see digits generated using the GAN divergence and see that we had the expected result, the images are quite similar and the FID values are close. They may differ because of randomness in weight initialization and data sampling from the MNIST dataset. A self critic here is that we didn't seeded these variables in order to compare properly the results.



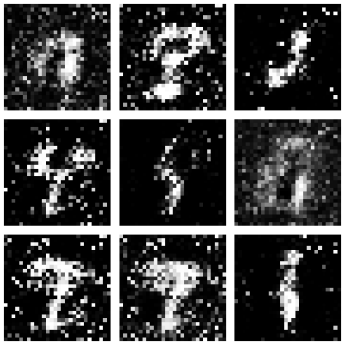
(a) Jensen Shannon Digits



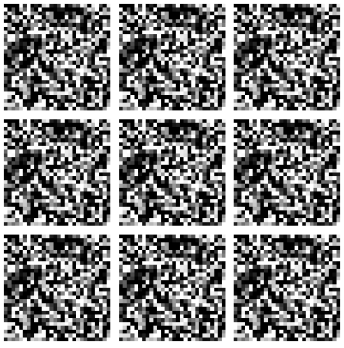
(b) Hellinger Digits



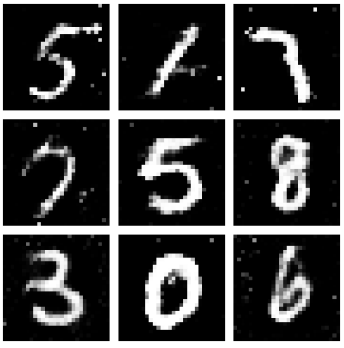
(c) Pearson Digits



(d) Reverse KL Digits



(e) KL Digits



(f) BCE Digits

Figure 3: Generated samples with different divergences

In the Figure 3 we can see some generated samples with different divergences. These samples doesn't reflect totally the potential of the loss functions of these objective functions since with different initializations they yielded different and sometimes better results.

One important empirical observation was that the model with the BCE objective function was stable and gave similar results each time we trained it while the other objective functions were really unstable. The Jensen Shannon divergence was more stable since as we can see in Figure 2 it's really similar to the GAN function, which is equivalent to BCE.
In the table 2 we can see the FID for models trained with different f-divergences.

Model	**FID Score**
f-divergence GAN	49
f-divergence KL	441
f-divergence Reverse KL	230
f-divergence Pearson X^2	196
f-divergence Squared Hellinger	319
f-divergence Jensen-Shannon	79

Table 2: FID Scores for Different Divergences

III Latent Space Re-weighting

III.1 Principle

Latent space re-weighting aims at improving the quality of the images generated by the GAN after the training. To do so, we associate to each latent space variable a weight that indicates its importance, those weights are learned during training by adversarial learning. These weights help in avoiding low-quality images and concentrate on producing high-quality ones, thanks to a policy applied before generating fake samples.

The approach over-parameterizes the generative distributions by defining a parametric class of importance weight functions, each of which maps from a multi-dimensional space to positive real numbers. The goal is to minimize the Wasserstein distance between the true distribution and a newly modeled class of generative distributions. This is achieved by finding the optimal importance weights. The importance weight function, implemented using a feed-forward neural network, re-distributes the mass of the original distribution so that the new distribution is closer to the target distribution in terms of Wasserstein distance. Thus, the discriminator approximates the Wasserstein distance and is trained adversarially with the importance weight function, while the weights of the generator remain unchanged. Having D_α and G_θ respectively the discriminator and the generator associated to the this sub-GAN problem and w^ϕ being the weights of the latent space variables, the training process looks for optimizing the following problem :

$$\min_{\phi\in\Phi}\max_{\alpha\in A}\mathbb{E}_{x\sim_{data}(x)}[D_\alpha(x)] - \mathbb{E}_{z\sim Z}w^\phi(z)\times D_\alpha(G_\theta(z))$$

The algorithm used to learn the weights is on the right side. Note that the gradient penalty (GP) method is used in order to bring more stability to the to training and the hyper-parameters used are taken from the paper.

Algorithm 1 Adversarial learning of w^ϕ

Require: Data μ_n, Prior Z, Gen. G_θ, Disc. D_α, number of D_α updates n_d, soft-clipping param. m, regularization weights λ_1 and λ_2, batch size b;
while ϕ has not converged **do**
  for $i = 0, \dots, n_d$ **do**
    Sample real data $\{x_i\}_{i=1}^b \sim \mu_n$;
    Sample latent vectors $\{z_i\}_{i=1}^b \sim Z$;
    $\text{EMD} \leftarrow \frac{1}{b}\sum_{i=1}^b D_\alpha(x_i) - w^\phi(z_i)D_\alpha(G_\theta(z_i))$;
    $\text{GP} \leftarrow \text{Gradient-Penalty}(D_\alpha, x, G_\theta(z))$;
    $\text{grad}_\alpha \leftarrow \nabla_\alpha(-\text{EMD} + \text{GP})$;
    Update α with grad_α;
  end for
  Sample $\{z_i\}_{i=1}^b \sim Z$;
  $\Delta \leftarrow \min_i |D_\alpha(G_\theta(z_i))|$;
  $\text{EMD} \leftarrow \frac{1}{b}\sum_{i=1}^b w^\phi(z_i)D_\alpha(G_\theta(z_i)) - \Delta$;
  $R_{\text{norm}} \leftarrow (\frac{1}{b}\sum_{i=1}^b w^\phi(z_i) - 1)^2$;
  $R_{\text{clip}} \leftarrow \frac{1}{b}\sum_{i=1}^b \max(0, w^\phi(z_i) - m)^2$;
  $\text{grad}_\phi \leftarrow \nabla_\phi(\text{EMD} + \lambda_1 R_{\text{norm}} + \lambda_2 R_{\text{clip}})$;
  Update ϕ with grad_ϕ;
end while

Once the weights are trained, one need to apply a policy to determine if we sample from a variable z from the latent space or not. A sample z from the latent space is accepeted with a probability given by $P_a(z) = \frac{w^\phi(z)}{m}$. This gives us the following algorithm of latent rejection sampling :

Algorithm 2 LatentRS

Require: Prior Z, Gen. G_θ, Importance weight network w^ϕ, maximum importance weight m;
  while True **do**
    Sample $z \sim Z$;
    Sample $\alpha \sim$ Uniform$[0, 1]$;
    if $\frac{w^\phi(z)}{m} \geq \alpha$ **then**
      break;
    end if
  end while
  $x \leftarrow G_\theta(z)$;
  Result: Selected point x

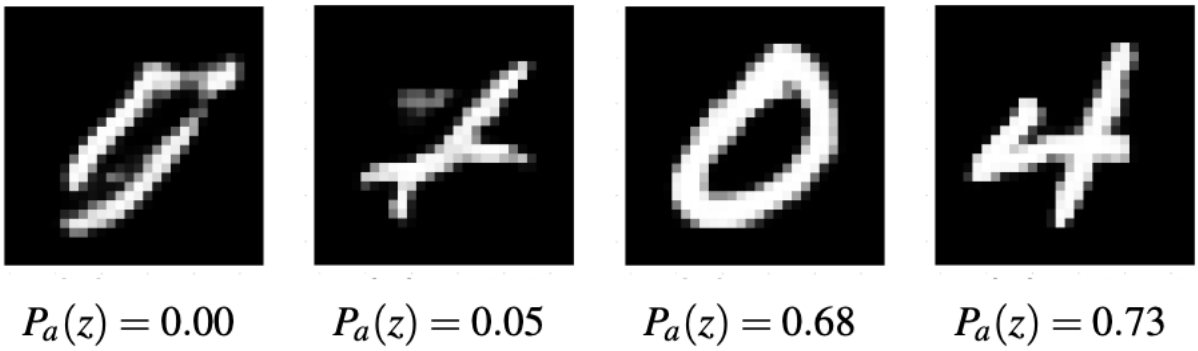


Figure 4: Probability of acceptance and associated image

As expected, the quality of images correlates with higher acceptance rates, illustrated by higher probability $P_a(z)$.

III.2 Results

Figure 5: Vanilla-GAN

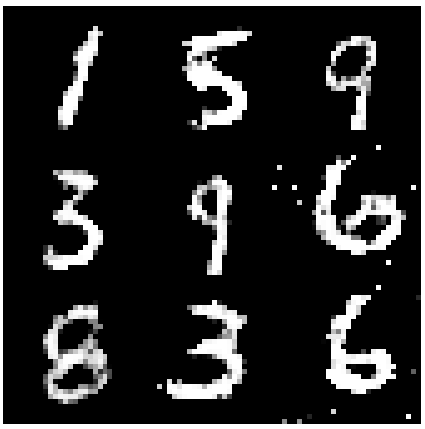


Figure 6: Vanilla-GAN with latentRS

As expected, we have better results as the images generated are more often looking like the real data : less weird digits are generated. In terms of FID, it does not improve a lot as we go from 49 to 46, however, the precision gets better as we obtain an improvement of 15 points to reach 0.38.

III.3 Efficiency

Besides, working on the latent space is quite interesting as its size (100 in our case) is much smaller than the feature space (28x28). So, it makes the rejection sampling more efficient. However, the training becomes heavier as we have to train another adversarial network to learn the importance weights. Thus in comparison to the Discriminator Rejection Sampling, once the two models are trained, the latent RS approach may be more efficient but should take more time to train. These two approaches are very different : the first one takes part in training whereas the second one comes after it to accept reject some sample using the latent space.

IV Conclusion

In this project, we looked at two different ways to improve a GAN. First of all, we have to mention that we had issues because of the hardness to make a GAN converge. That is what research have been looking to it very carefully to make it converge. The f-GAN models use different divergence to compute the dissimilarities between two distributions, while the second approach looks into the latent space and try to generate better images based on a latent policy.

References

[1] Nowozin, S., Cseke, B., & Tomioka, R. (2016). "f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization." Advances in Neural Information Processing Systems (NIPS), 2016, 271-279.

[2] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). "Generative Adversarial Networks." Advances in Neural Information Processing Systems (NIPS), 2672-2680.

[3] Latent reweighting, an almost free improvement for GANs, Thibaut Issenhuth1,2, Ugo Tanielian1, David Picard 2, JÃ©rÃ©mie Mary, Criteo AI Lab, Paris, France.