Université Paris Sciences & Lettres

Data Science Lab A2 Report

Learning Latent Space Representations: Application to Image Generation

Students:

Xichen ZHANG, Mariem AALABOU, Hangyue ZHAO

Team: Yesterday

November 12, 2024

Learning Latent Space Representations

Xichen ZHANG, Mariem AALABOU, Hangyue ZHAO

November 12, 2024

Yesterday All my troubles seemed so far away - The Beatles

1 Introduction

In this short journey, we briefly go through several classic GANs, including Wasserstein GANs (Sec.3), Gaussian Mixture GANs(Sec.4), and Post-training Methods for Refining Latent Space (Sec.5). Each above section, we try to give some analysis regarding the imporvement. At the end, in Sec.(6), we show and summary the results

For the implementation GM-GANs, DRS and some plots, please refer to the kaggle notebook GM-GANs & DRS (GPU P100). See all the algorithms results Table.(1)

2 Background of GANs

Generative Adversarial Networks (GANs), introduced by Ian Goodfellow and his colleagues in 2014, represent a powerful framework for generative modeling, where two neural networks, a generator and a discriminator, are set up in a competitive setting.[1] The generator's goal is to produce realistic images from a random vector sampled from a latent space, essentially learning how to create data that resembles the real dataset. The discriminator, on the other hand, acts as a binary classifier, attempting to distinguish between real images (from the training set) and fake images created by the generator(see Fig. 1).

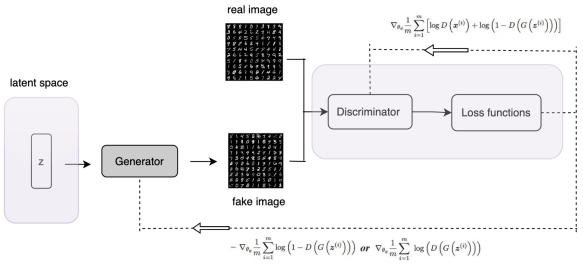


Figure 1: *GANs Playgrounds*

In GANs, the generator and discriminator play a minimax game. The generator tries to minimize the probability that its fake images are recognized as fake, while the discriminator tries to maximize its accuracy in telling real images from fake ones. This competition is captured by the objective function:

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

2.1 The Balance of Players

For GANs to work well, there needs to be a balance: if the discriminator gets too strong, the generator may struggle to improve, and vice versa. Through backpropagation, the gradients from the discriminator help the generator learn to make better images. However, challenges like mode collapse (where the generator produces limited types of images) and training instability can occur, which often require further improvements in the discriminator architecture or adjustments to the loss functions (discussed in the next Sec.3).

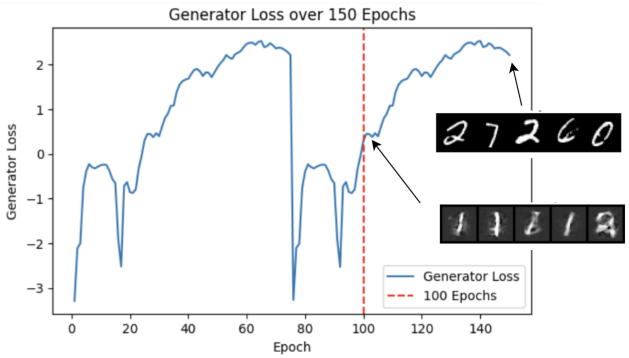


Figure 2: *Impact of a Strong Discriminator on Generator Performance*

In this Fig.2, we observe the generator's loss over 150 epochs when trained against a discriminator equipped with convolutional layers, which make the discriminator more powerful and accurate. This increased strength of the discriminator can cause two main issues for the generator.

- First, mode collapse, where the generator starts producing only a limited variety of outputs (shown by repetitive digits in the figure), occurs because the generator struggles to explore diverse outputs.
- Second, training instability arises, as shown by fluctuations in the loss curve, indicating that the generator has difficulty learning effectively under the pressure of a strong discriminator.

This illustrates the challenge of balancing generator and discriminator capacities for stable GAN training.

3 Wasserstein GANs

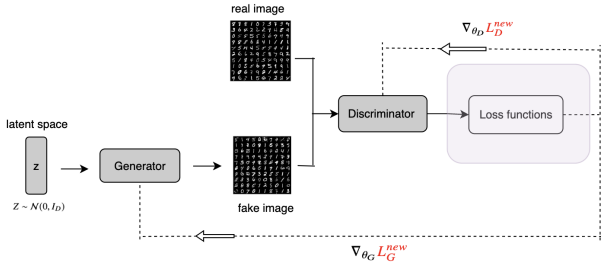


Figure 3: *W-GAN Structure*

The Wasserstein Generative Adversarial Network (W-GAN) improves on traditional GANs by addressing issues like training instability and mode collapse, using the Wasserstein distance as a new metric to measure the difference between real and generated data distributions. This metric offers a smoother, more stable convergence compared to the Jensen-Shannon (JS) divergence used in standard GANs, especially when the distributions of real and generated samples do not overlap[1].[2]

3.1 Kantorovich-Rubinstein Duality

In the continuous probability domain, the Wasserstein distance $W(p_r, p_g)$ between the real data distribution p_r and generated data distribution p_g is defined as:

$$W(p_r, p_g) = \inf_{\gamma \sim \Pi(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \tag{2}$$

where $\Pi(p_r, p_g)$ is the set of all possible joint distributions γ with marginals p_r and p_g.

Each $\gamma(x, y)$ represents a plan for moving mass from point x to point y , minimizing the average "cost" $\|x-y\|$ of transporting the "mass" to transform p_r into p_g.

Directly computing $W(p_r, p_g)$ is intractable, so W-GAN reformulates it using the Kantorovich-Rubinstein duality:

$$W(p_r, p_g) = \frac{1}{K} \sup_{\boxed{\|f\|_L \leq K}} \mathbb{E}_{x\sim p_{data}}[f(x)] - \mathbb{E}_{x\sim p_g}[f(x)]$$

Here, f is constrained to be K-Lipschitz continuous, meaning that $|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$ for any x_1, x_2 . This requirement on f ensures stable optimization by limiting how quickly f can change. This Lipschitz constraint on the discrimintator is the key for stability.

3.2 W-GAN Loss Function and Training Process

In WGAN, the discriminator is replaced by a critic that learns the optimal K -Lipschitz function f_w , parameterized by w , to estimate the Wasserstein distance. The W-GAN loss function is:

$$\begin{aligned} L(p_r, p_g) &= W(p_r, p_g) \\ &= \max_{w\in W} \mathbb{E}_{x\sim p_r}[f_w(x)] - \mathbb{E}_{z\sim p_z}[f_w(G_\theta(z))] \end{aligned}$$

where $G_\theta(z)$ represents the generator's output from a random noise z. As the loss decreases, the Wasserstein distance between p_r and p_g decreases, meaning the generated data distribution gets closer to the real data distribution.

3.3 Enforcing Lipschitz Continuity with Weight Clipping and Alternatives

To enforce the Lipschitz constraint, W-GAN initially used weight clipping: after each gradient update, the weights of f_w are clamped to a small fixed range, typically [-c, c] . This approach helps keep f_w K-Lipschitz but can lead to problems like slow convergence and vanishing gradients.

An alternative approach, gradient penalty, was introduced to replace weight clipping. It penalizes the norm of the gradient of f with respect to its input, encouraging f to remain Lipschitz without strictly limiting weight values. This is more stable than weight clipping, as it allows the model to maintain the Lipschitz constraint smoothly.

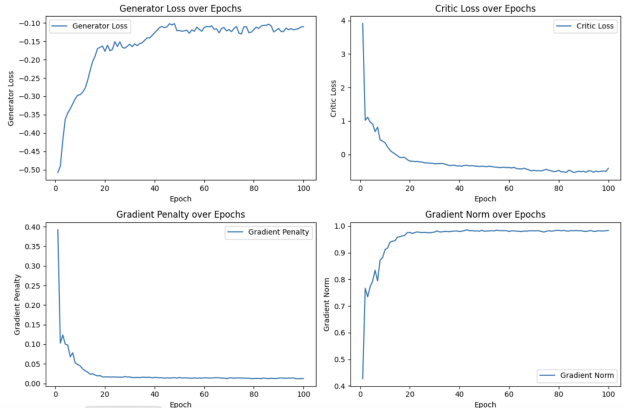


Figure 4: *Training Progress of WGAN-GP: Generator and Critic Loss, Gradient Penalty, and Gradient Norm over 100 Epochs Showing Stable Convergence*

This Fig.4 shows the stable training process of W-GAN-GP. The generator loss gradually rises and stabilizes, indicating effective learning. The critic loss initially drops quickly, then flattens, showing the critic effectively distinguishes between real and fake data. The gradient penalty and gradient norm both stabilize early, maintaining the Lipschitz constraint required for the Wasserstein distance. Overall, these plots confirm that W-GAN-GP achieves smooth training, with each component working as intended.

3.4 Advantages of W-GAN

WGAN offers several advantages over traditional GANs:

- Smoother and More Meaningful Loss: The Wasserstein distance provides a continuous and differentiable measure, even when the distributions do not overlap, which helps with stable training.
- Better Convergence: Using Wasserstein distance allows for more gradual improvements in the generator, reducing the likelihood of mode collapse.

[1] This section uses Lil'Log, one of the best blogs out there, as a reference

It is worth to note that W-GAN does not guarantee performance improvements.

4 Gaussian Mixture GANs

In this section, we introduce Gaussian Mixture GANs (Dynamic GM-GAN), an algorithm that dynamically adjusts parameters in the latent space during training.[3]

Gaussian Mixture GANs (GM-GANs) improve GANs by using a multi-modal Gaussian distribution for the latent space, aligning better with complex, diverse datasets. Dynamic GM-GAN goes further by updating the Gaussian parameters (means and covariances) during training, allowing the latent space to adapt continuously for a better match with the data distribution. This dynamic adjustment leads to improved performance and convergence by refining the latent representation throughout the training process.

This approach refines the latent representation throughout the training process. In contrast to traditional generative models, where the generator G is optimized without any prior knowledge of the latent space and refinement occurs only after training, GM-GANs dynamically refine the latent space during training. This approach has shown significant improvements, achieving a distribution that closely matches the optimal target distribution.

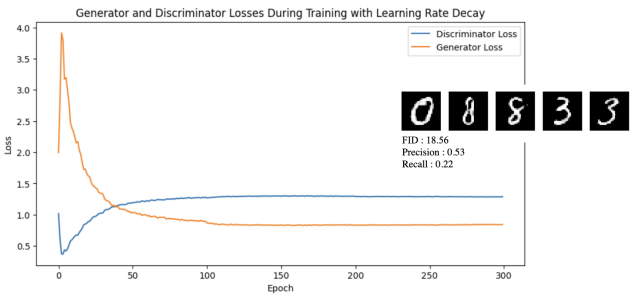


Figure 5: *Training Dynamics of GM-GAN with Learning Rate Decay: Generator and Discriminator Losses over 300 Epochs*

The Fig.5 shows the training process of a Gaussian Mixture GAN (GM-GAN) with dynamic adjustments to the latent space, optimized for the MNIST dataset, which has 10 distinct classes. The loss curves demonstrate balanced adversarial training, with the generator and discriminator losses stabilizing as training progresses. By structuring the latent space into clusters (matching MNIST's 10 classes), the GM-GAN can better represent the unique features of each digit, allowing the generator to produce distinct digit-like samples. This dynamic approach of training and continuously refining the latent space during the learning process enhances the generator's capacity to reach an optimal distribution. The evaluation metrics—FID of 18.56, Precision of 0.53, and Recall of 0.22—indicate good quality and diversity of the generated samples, with the generated images (shown on nearby) closely resembling real MNIST digits.

4.1 Why Dynamic GM-GAN works?

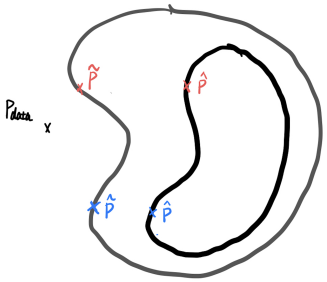


Figure 6: *Illustration of training and refining GM-GAN at same time.*

This Fig.6 illustrates how Dynamic GM-GAN refines its latent space during training to achieve a closer match to the target data distribution, P_{data} . The black boundary represents the initial, fixed latent space where the distribution of GM-GAN begins. As training progresses, the model dynamically refines this latent space, as shown by the grey boundary, allowing it to adjust and better fit the data distribution.[2]

Argument for Simultaneous Training and Refinement: The inner regions marked with $\hat{P}$ and $\tilde{P}$ represent distributions learned at various stages, where the red points indicate simultaneous approximations(training + refining) and the blue points(pretrained + refining) show fixed latent positions. By simultaneously training the generator and refining the latent space, Dynamic GM-GAN incrementally adjusts its representation, which helps it move closer to the optimal distribution. This iterative refinement reduces the gap between the learnt distribution of the model and P_{data}, effectively decreasing divergence and improving the quality of generated samples.

This adaptive approach helps avoid the fixed latent space, which can restrict the ability of the model to represent complex data distributions. Consequently, this process brings Dynamic GM-GAN closer to an optimal solution, as it learns not only from the target data but also fine-tunes the latent space to more accurately represent it.

5 Post-training Methods for Refining Latent Space

To improve the quality and diversity of samples in generative models, post-training refinement of the latent space is essential. By optimizing the structure and sampling within the latent space, we can guide the generator to produce more realistic and varied outputs. Refining the latent space also helps capture finer details of the data distribution, leading to generated samples that more closely match real data. This section introduces several latent space optimization methods, which used to enhance the performance of generative models.

5.1 Discriminator Rejection Sampling (DRS)

Discriminator Rejection Sampling (DRS) is a method designed to improve the quality of samples generated by a GAN.

[2] This drawing and argument are inspired by the insightful Section 6.3 from Verine's Thesis as a reference.

The basic idea is to use the GAN discriminator not only as a tool for training but also as a filter to select the best samples from those generated by the model. DRS works by checking each sample's "realness" score from the discriminator and keeping only those that are most likely to resemble real data.[4][5]

The discriminator is trained to distinguish between real and fake data. When it has reached a good level of accuracy (at convergence), we can use its output to estimate a **density ratio** $r(x)$, which tells us how likely it is that a given sample x resembles real data. Mathematically, the density ratio $r(x)$ is given by:

$$r(x) = \nabla f^*(D(x)), \tag{3}$$

where $D(x)$ is the discriminator's output for the sample x, and f^* is a function that helps interpret $D(x)$ in terms of how closely x matches the real data distribution $p(x)$.

Using this density ratio, DRS applies an **acceptance function** $a(x)$, which determines the probability of accepting each generated sample:

$$a(x) = \frac{r(x)}{M}, \tag{4}$$

where M is the maximum density ratio observed across a large set of generated samples (estimated during a "burn-in" phase). Samples with higher density ratios are more likely to be accepted, while those with lower ratios are more likely to be rejected. This acceptance function effectively filters out lower-quality samples, improving the overall quality of the generated data.

In some cases, the acceptance rate might be too low (meaning we are too selective). To adjust for this, a scaling factor γ is introduced to make the acceptance function more flexible:

$$a_{\text{DRS}}(x) = \frac{r(x)}{r(x)(1 - e^{\gamma}) + Me^{\gamma}}. \tag{5}$$

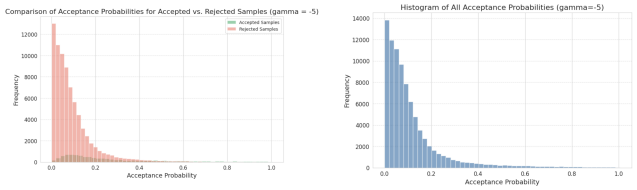
With $\gamma < 0$, we can increase the acceptance rate, allowing more samples to pass through the filter. This helps in cases where a higher acceptance rate is needed without compromising too much on sample quality.

In summary, DRS uses the discriminator's output to keep only the best samples from the generator, improving both quality and diversity by selectively choosing those that closely align with real data.

Fig.7 and 8 display the acceptance probabilities for Discriminator Rejection Sampling (DRS) at $\gamma = -5$ and $\gamma = -8$, respectively. With $\gamma = -5$ (Fig.7), DRS is highly selective, resulting in an 11.43% acceptance rate, meaning that most accepted samples have low probabilities. This strict filtering favors high-quality samples but also significantly increases sampling time, demanding more computational resources. In contrast, with $\gamma = -8$ (Fig.8), the acceptance rate rises to 57.62%, allowing a broader range of samples and enhancing diversity. This relaxed threshold reduces sampling time and computational load but may admit some lower-quality samples.

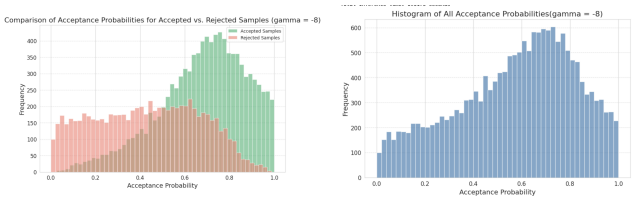
Here is Algorithm(1) and (2) for DRS with Summary of Variables:

- G: Generator
- D: Discriminator
- n: Number of samples processed per batch.
- B: Number of batches used to estimate M.
- M: Estimated maximum density ratio used for normalization in DRS.
- N: Target number of samples to generate using DRS.
- γ: Scaling factor adjusting the acceptance probability in DRS.



(a) **(b)**

Figure 7: *DRS Acceptance Probability Histograms ($\gamma = -5$): Comparison of Accepted vs. Rejected Samples and Overall Distribution with 11.43% Acceptance Rate.*



(a) **(b)**

Figure 8: *DRS Acceptance Probability Histograms ($\gamma = -8$): Comparison of Accepted vs. Rejected Samples and Overall Distribution with 57.62% Acceptance Rate.*

Algorithm 1 Discriminator Rejection Sampling (DRS)

Require: G, D, M, N, n, γ
Ensure: Generated samples $\{x_j\}_{j=1}^{N}$

Initialize $\mathcal{S} \leftarrow \emptyset$
while $|\mathcal{S}| < N$ **do**
  Sample $\{z_i\}_{i=1}^{n}$ from latent space P_z
  Generate $\{x_i\}_{i=1}^{n}$, where $x_i \leftarrow G(z_i)$
  Compute $p_i \leftarrow D(x_i)$
  Compute $r_i \leftarrow \dfrac{p_i}{1 - p_i}$
  Compute acceptance probabilities:

$$a_i \leftarrow \frac{r_i}{r_i(1 - e^{\gamma}) + Me^{\gamma}}$$

  for $i = 1$ to n **do**
    Sample $u_i \sim \text{Uniform}(0, 1)$
    if $u_i \leq a_i$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{x_i\}$
    end if
  end for
end while
return $\{x_j\}_{j=1}^{N}$ from $\mathcal{S}$

Algorithm 2 Estimate Maximum density ratio M

Require: G, D, n, B
Ensure: Maximum density ratio M
$M \leftarrow 0$
for $k = 1$ to B **do**
  Sample $\{z_i\}_{i=1}^n \sim P_z$
  Generate $\{x_i\}_{i=1}^n$ where $x_i \leftarrow G(z_i)$
  Compute $p_i \leftarrow D(x_i)$ for $i = 1$ to n
  Compute $r_i \leftarrow \dfrac{p_i}{1-p_i}$ for $i = 1$ to n
  $M \leftarrow \max(M, \max\{r_i\})$
end for
return M

5.2 Discriminator Gradient flow

The DG-FLOW method refines samples from a generative model by leveraging gradient flow to move generated samples closer to the target data distribution. This process is based on a Fokker-Planck equation with a stochastic differential equation (SDE) counterpart:

$$d\mathbf{x}_t = -\nabla_{\mathbf{x}} f\left(\frac{\rho_t}{\mu}\right)(\mathbf{x}_t)\, dt + \sqrt{2\gamma}\, d\mathbf{w}_t$$

where $\mathbf{x}_t$ represents the evolution of a sample under the influence of drift and diffusion. To push the generated samples (initially drawn from ρ_0) closer to the target density μ, the SDE is simulated using methods like Euler-Maruyama.

The density ratio $\frac{\rho_t(\mathbf{x})}{\mu(\mathbf{x})}$ is approximated with a discriminator D trained to distinguish between samples from ρ_0 and μ, given by:

$$\frac{\rho_0(\mathbf{x})}{\mu(\mathbf{x})} = \frac{1 - D(y=1|\mathbf{x})}{D(y=1|\mathbf{x})} = \exp(-d(\mathbf{x}))$$

where $d(\mathbf{x})$ denotes the logit output from the discriminator. This process, called Discriminator Gradient Flow (DG-FLOW), refines samples using the gradient flow of f-divergences based on the discriminator output.

5.3 Discriminator Driven Latent Sampling (DDLS)

DDLS introduces a sampling approach for Wasserstein GANs by incorporating Langevin dynamics, allowing the generative process to be treated as an energy-based model (EBM). In this approach, the Wasserstein GAN loss objectives are defined as follows:

$$L_D = \mathbb{E}_{p_g}[D(x)] - \mathbb{E}_{p_d}[D(x)], \quad L_G = -\mathbb{E}_{p_0}[D(G(z))]$$

where D is constrained to be a K-Lipschitz function.

This method interprets the discriminator as an energy function that defines a density model, where the discriminator phase (D-phase) aims to match the data distribution p_d with an intermediate distribution $p_t(x) \propto p_g(x)e^{D_\phi(x)}$. The generator phase (G-phase) then adjusts the generator to match $p_g(x)$ with $p_t(x)$.

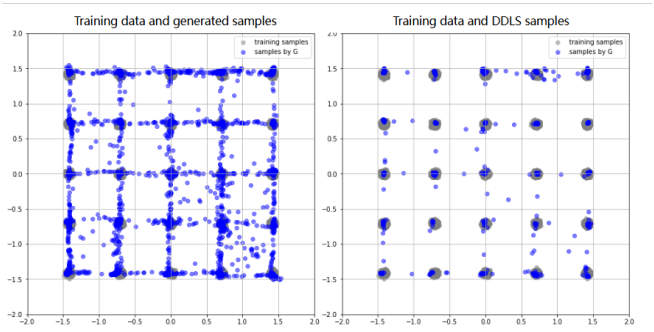


Figure 9: *Comparison between DDLS and generator alone on 2d mixture of Gaussians distribution*

As shown in the Fig.9, compared to the original generator, DDLS can sample better-quality examples in the latent space on the 25-gaussians dataset. DDLS suffers less from mode dropping when modeling the 2D synthetic distribution.

6 Results

We conducted experiments on the MNIST dataset, as shown in Table 1, using FID, precision (P), and recall (R) as evaluation metrics. FID measures the similarity between generated and real data, with lower values indicating better quality. Precision reflects how well generated samples match real data, while recall measures the diversity of the generated samples.

The results show that:
- FID: GM GAN performs best with a score of 18.56.
- P: GM GAN achieves the highest precision of 0.53.
- R: W-GAN shows the highest recall at 0.29.

Overall, GM GAN demonstrates the best performance in FID and precision, while W-GAN excels in recall.

The grey cell indicates uncertainty—either because the theoretical validity is unclear or because the implementation may not be correct.

Table 1: *Comparison of Results for Different GAN Models*

Model	FID	P	R
Baseline VanillaGAN	63.99	0.39	0.21
VanillaGAN with DRS	38.49	0.34	**0.28**
W-GAN	40.78	0.34	**0.29**
W-GAN with DRS	38.02	**0.51**	0.25
W-GAN with Dgflow	40.02	0.47	0.25
GM GAN	**18.56**	0.53	0.22

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[2] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *Advances in neural information processing systems*, vol. 30, 2017.

[3] M. Ben-Yosef and D. Weinshall, "Gaussian mixture generative adversarial networks for diverse datasets, and the unsupervised clustering of images," *arXiv preprint arXiv:1808.10356*, 2018.

[4] S. Azadi, C. Olsson, T. Darrell, I. Goodfellow, and A. Odena, "Discriminator rejection sampling," *arXiv preprint arXiv:1810.06758*, 2018.

[5] A. Verine, "Quality and diversity in generative models through the lens of f-divergences," Ph.D. dissertation, Université Paris sciences et lettres, 2024.