Supervised by Benjamin Negrevergne and Alexandre Vérine

Assignment 2
Learning latent space representation and application to Image Generation

EleGANt

2024-2025

BOUSSOUF Noâm

ELAFANI Maïssa

KANLI Emre Çağan

1 Baseline : Vanilla GAN

GAN models consist of two networks: the Generator and the Discriminator. While the Generator tries to create realistic fake data, the Discriminator tries to distinguish real and generated data. This adversarial environment helps both networks to improve on their tasks. For our model, firstly we tried running the given model with different parameters. In the end, the best FID value we had from Vanilla GAN was **38**. For this project, this was our baseline and we tried to implement different methods to get a better result on FID.

2 Wasserstein GAN [7]

WGAN is a variation of GAN, where the main idea is to use a different loss function called Earth Mover. To have this loss function, some changes are required such as unbounded discriminator output and 1-Lipschitz discriminator function. After trying different methods and structures, we had good results with two seperate models.

$$\mathcal{L}_{\text{Critic}} = E_{\mathbf{x}\sim P_{\text{data}}}\left[D(\mathbf{x})\right] - E_{\mathbf{z}\sim P_{\mathbf{z}}}\left[D(G(\mathbf{z}))\right]$$

$$\mathcal{L}_{\text{Generator}} = -E_{\mathbf{x}\sim P_{\text{z}}}[D(G(\mathbf{x}))]$$

2.1 Normalization for the Generator Layers

Inspired by the implementation [11], we put different normalization steps between the generator's layers. When we added these steps, the model worked well and generated good images. We also tried dropout for the discriminator alongside the normalization. But any dropout rate we tried made the generation worse. Trying both, batch normalization did better than layer normalization. With batch normalization added to WGAN and nothing else, we got a FID value of **27**. However, since this meant changing the generator's architecture, we did not follow on this path to stay true to the aim of this project.

2.2 Gradient Penalty [10]

WGAN-GP is a commonly used modification of WGAN where a gradient penalty term is added to the loss function of the Discriminator to force 1-Lipschitz requirement.

$$\mathcal{L}_{\text{GP}} = \lambda \cdot E_{\hat{\mathbf{x}}\sim P_{\hat{\mathbf{x}}}}\left[\left(\left\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\right\|_2 - 1\right)^2\right]$$

$$\mathcal{L}_{\text{Critic}} = E_{\mathbf{x}\sim P_{\text{data}}}\left[D(\mathbf{x})\right] - E_{\mathbf{z}\sim P_{\mathbf{z}}}\left[D(G(\mathbf{z}))\right] + \mathcal{L}_{\text{GP}}$$

For a WGAN-GP trained on 200 epochs, we saw that the FID was starting to decrease after 150 epochS. To have a better model, we used the checkpointS of the 150th epoch (which gave the best fid) to train the model with a lower learning rate for 30 more epochs(we divided by 2 the original ones from both generator and discriminator). This is equivalent to performing learning rate decay with an "early-stopping" technique rather than with a fixed number of epochs stepsize (i.e. we start decreasing the stepsize when the fid starts decreasing).

2.3 Results

Using WGAN-GP Discriminator Loss, the resulting model gave a FID score of **28** initially. After using learning decay, the FID score we got was 16. The implementation is inspired from [1].

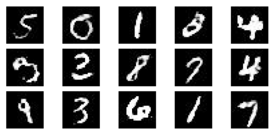

Figure 1: Random samples generated with WGAN-GP

3 Regularization and optimization methods

In this section, we will tackle the methods that we have implemented and tested that aim at improving the training of GANs (which are known to be very unstable) by providing a different optimization technique or aiming at adding a regularization effect.

3.1 Mixture Density GANs [9]

3.1.1 Description of the method

Mixture Density GANs (MD-GANs) [9] are a type of GAN where the discriminator models the embedding space of real data as a mixture of Gaussians that forms clusters in the embedding space (a Simplex Gaussian Mixture Model). For a given embedding e, the likelihood is defined as:

$$lk(e) = \sum_{i=1}^{C} \frac{1}{d+1} \cdot \Phi(e; \mu_i, \Sigma_i)$$

where each Gaussian component $\Phi(e; \mu_i, \Sigma_i)$ has mean μ_i and covariance Σ_i. The discriminator maximizes the likelihood $\log(lk(D(x)))$ for real data encodings while minimizing $\log(\lambda - lk(D(G(z))))$ for fake ones (with λ set to the maximum of $lk(e)$). By doing so, the discriminator aims at creating several clusters in its output embedding space for **real** images. The generator, in turn, tries to draw fake embeddings towards the center of these clusters, i.e. it is rewarded if it generates samples that end up close to any of the clusters in the discriminator's embedding space which encourages the generator to cover multiple modes.
Note that each cluster is represented by a Gaussian kernel, puting these clusters together gives what is called a Gaussian Mixture Model. But here we make a strong assumption that every cluster center is equidistant (hence the appelation **Simplex** Gaussian Mixture Models).

3.1.2 Results

We inspired our implementation from the original one of the paper [2] and another one [3]. We first tried the method in the 2D-case (2D Grid of Gaussians). We show the results below with a 2D Grid of 25 Gaussian components (Figure 2) (as in the original paper) and 9 Gaussian components (Figure 3) :

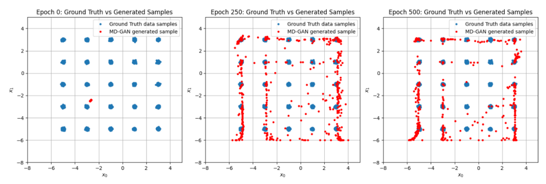

Figure 2: Results of MD-GANs for 25 gaussian components

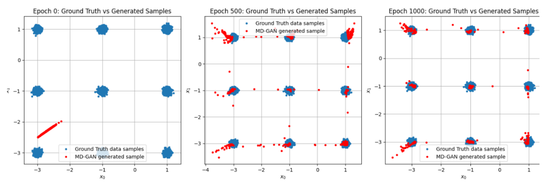

Figure 3: Results of MD-GANs for 9 gaussian components

The results seem very nice in the 2D case with clusters being equi-distant. However we've had poor results on the MNIST dataset, we couldn't find a set of hyperparameters that worked well **with the imposed architecture** (indeed in the original paper [9], the authors get results on MNIST by using DCGAN) and we decided to give up on this method after many hours spent on it to focus on the improvment of WGAN-GP. We didn't even suceed to find a set of parameters that works well on the 2D case with the Vanilla architecture **that was originally on the platform**. Indeed, using it led to instant model collapse in the 2D case and vanishing gradients with MNIST (and tuning hyperparameters didn't improve a lot). Here the results provided in the 2D case use one less fully connected layers than the original ones (of the platform).
NB : More implementation details can be found in the branch "MD-GAN" of our Github. The training for MNIST can be done using the file "train_MD.py" (but it doesn't give good results) and for the results shown before, one can use the script run.sh.

3.2 Unrolled GANs [13]

3.2.1 Description of the method

Unrolled GANs aim at improving the stability of training in traditional GANs by 'unrolling' the discriminator's optimization process. Instead of only using the current feedback from the discriminator, the generator anticipates the discriminator's future responses by simulating several steps of its optimization. This is done by adjusting the generator's objective to include not just the current discriminator's output but also the discriminator's responses **after multiple optimization steps**. Indeed, during training, the discriminator is updated for several steps independently from the generator, while the generator is trained by considering both the current and unrolled discriminator's responses. This unrolling process's goal is to help the generator to anticipate the future improvements in the discriminator, which may lead to more stable and effective training.

3.2.2 Results

We inspired our implementation from the original implementation of the paper [4] and another one [5]. We first tried to implement the method in the 2D-case where we took random samples from a Normal Distribution gathered in a circle (to compare our results with the ones of the original paper). For the results shown below we start plotting at iteration 300 and end it at iteration 1500 with a 300 iterations step. The original samples are in green and the generated ones are in blue.

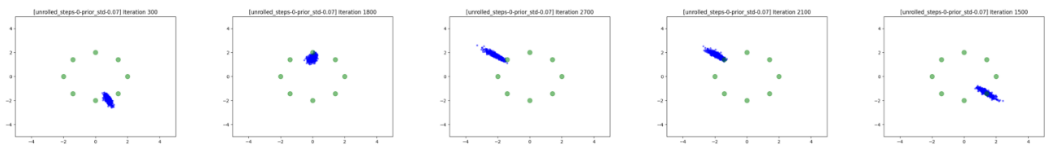

Figure 4: Results of a Vanilla GAN architecture on 2d Gaussian samples gathered in a circle

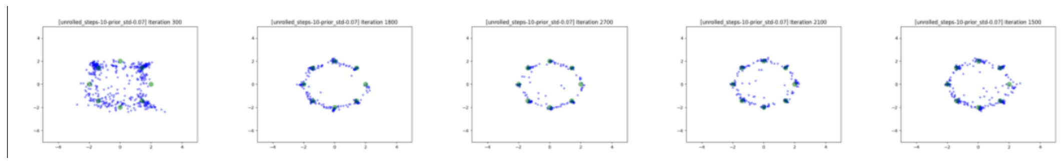

Figure 5: Results of Unrolled-GAN with Vanilla architecture on 2d Gaussian samples gathered in a circle

We can see on figure 5 that the results are very nice with Unrolled GANs on a 2D case. However we've had poor results on the MNIST dataset, we couldn't find a set of hyperparameters that worked well **with the imposed architecture** (indeed in the original paper [9], the authors get great results on MNIST by using other architectures) and we decided to give up on this method after many hours spent on it to focus on the improvment of WGAN-GP. We didn't even suceed to find a set of hyperparameters that work well on the 2D case with the Vanilla architecture **that was originally on the platform**. Indeed, using it led to instant model collapse, and tuning the learning rates of the discriminator and generator didn't improve much. Here the results provided in the 2D case use one less fully connected layers than the original one (of the platform).

3.3 R1-Regularization [12]

We tried the R1 Regularization technique [6] which aims at stabilizing the training of the models and improve their generalization abilities. It consists in adding a regularization term to the discriminator's loss function that penalizes the gradient of the discriminator's output on real data:

$$R_1(\psi) = \frac{\gamma}{2} E_{p_D(x)} \left[\|\nabla D_\psi(x)\|^2 \right]$$

with ψ representing the parameters of the discriminator that we aim to optimize; γ, the regularization hyperparameter that controls the strength of the penalty. The higher the value of γ, the more impact the regularization term will have on the discriminator's learning. And with $E_{p_D(x)} [\|\nabla D_\psi(x)\|^2]$, the expected value of the norm of the gradient of the discriminator calculated on real data which measures how the discriminator changes as real input data varies.

In summary, the regularization term $R_1(\psi)$ penalizes the model when the discriminator's "curvature" is too strong on real data preventing it from forming overly sharp decision boundaries. It

means that the model's boundaries between different classes are so finely tuned to specific data points that they may capture noise or unnecessary details making the model prone to errors when new data doesn't exactly match the training data. This can lead to instability and overfitting which we are trying to avoid.
Using the R1 Regularization, the resulting model gave a FID of **42** which is worse than the baseline. To conclude, this method wasn't efficient in our setting or the hyperparameters were not optimal.

3.4 Discriminator Rejection Sampling [8]

Rejection Sampling aims to improve generated samples by rejecting some of the outputs according to an acceptance probability distribution. For the GAN case, the output of the discriminator for a generated sample, which is transformed to a number between 0 and 1 if it hasn't been already, is used as the probability. Using the Rejection Sampling Latent Space Optimization on the baseline and the best model we had, the FID scores improved from **38 to 22** for Vanilla GAN and took 32,760 attempts to produce 10,000 images. On the other hand, WGAN-GP rejected only 240 images but consequently did not improved its FID score much, only from 16 to 15.

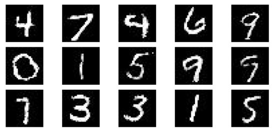
Figure 6: Random samples generated with Vanilla GAN and rejection sampling.

4 Conclusion

In this project, we tried various methods to improve the digits image generation of a GAN model. Compared to the baseline 38 for the baseline Vanilla Model, we created models that gave significantly better results (both visually and theoretically with fid). We chose not to evaluate theoretically our methods using Precision and recall because we had results far from what the platform gave us.

	VGAN	R1-Reg	WGAN w\BN.	WGAN-GP	VGAN w\DRS	WGAN-GP w\DRS	Unrolled	MD
FID	38	42	27	16	22	15	**inf**	inf

Figure 7: Evaluation of our methods using FID (locally) with the pytorch-fid library

Note that these results were given using the evaluation library pytorch-fid. As explained by e-mail, we had some issues with the platform. Moreover, we used generation of JPG files to get these results (compared with JPG images of the original test_dataset). Indeed for basic Vanilla-GAN for example, 37 fid on the platform, 30 on the local but when using PNG files and 38 when using JPG files. So we stuck to JPG files for both the original test dataset and the generated samples evaluated locally. Note also that we didn't push the results of WGAN-GP with rejection sampling in the platform, we only evaluated it locally (because of the technical issues mentionned below, feel free to use the file generate_WGAN.py to try it manually (but change the .jpg to .png in torchvision.save_image !)).

References

[1] https://github.com/wbjang/mnist_wgan_gp.

[2] https://github.com/eghbalz/mdgan.

[3] https://github.com/haihabi/MD-GAN.

[4] https://github.com/poolio/unrolled_gan.

[5] https://github.com/mk-minchul/unroll_gan.

[6] https://github.com/ChristophReich1996/Dirac-GAN/blob/decb8283d919640057c50ff5a1ba01b93ed86332/dirac_gan/loss.py#L292.

[7] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[8] Samaneh Azadi, Catherine Olsson, Trevor Darrell, Ian Goodfellow, and Augustus Odena. Discriminator rejection sampling, 2019.

[9] Hamid Eghbal-zadeh, Werner Zellinger, and Gerhard Widmer. Mixture density generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5820–5829, 2019.

[10] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

[11] Che-Jui Huang. Vanilla generative adversarial networks. Medium, 2022. Accessed: 2024-11-12.

[12] Sebastian Nowozin Lars Mescheder, Andreas Geiger. Which training methods for gans do actually converge? *arXiv preprint arXiv:1801.04406v4*, 2018.

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