PSL
UNIVERSITÉ PARIS

Generative Adversarial Network

Assignment 2 - Report

VeGANs team
Mathilde Kretz, Amélie Leroy, Marius Roger

November 17, 2023

Numbers generated with our latest implementation

Introduction

The goal of a Generative Adversarial Network (GAN) problem is to represent a learning, or latent, space in a certain way so that a generator can produce outputs from it without having access to the real data. The connection between the two is established by a discriminator, which is trained in opposition to the generator by attempting to distinguish real data from generated ones. This scenario can be formalized with the following optimization problem:

$$\min_G \max_D \mathbb{E}_{X_r \sim P}[\log D(x)] + \mathbb{E}_{X_g \sim \hat{P}_G}[\log(1 - D(x))]$$

In the context of the assignment, the structure of the generator is fixed. The overall objective was to generate images from the MNIST dataset.

The method explored in this report is the Gaussian Mixture model in the supervised learning setting.

1 Implementation Details

1.1 Dynamic Gaussian Mixture

The approach investigated in this assignment was based on the Gaussian Mixture Model as presented by [1]. The main objective of this approach is to model the latent space using various Gaussian distributions, from which we will sample the inputs of the generator. These distributions are represented by their mean and standard deviation, with a total of K distributions to choose from.

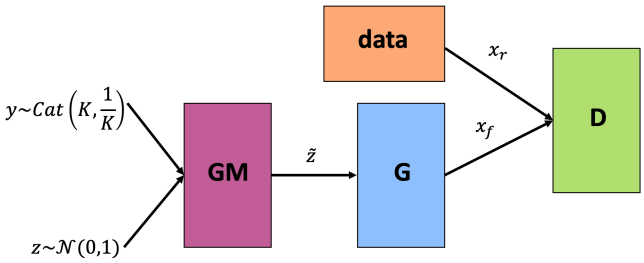

Figure 1: Architecture of the GM-GAN.

The implementation - inspired by the one described by [2] - can be illustrated by Figure 1 to depict the various components of the overall model:

The GM box represents the Gaussian Mixture module, which takes as input a random noise z, and a variable y representing one of the K Gaussian distributions we want to sample. The output follows the k^{th} Gaussian Distribution $\mathcal{N}(\mu_k, \sigma_k)$ of the latent space, defined as $\tilde{z} = \mu(y) + \sigma(y) \cdot z$. The parameters μ (mean) and σ (standard deviation) act as weights for the GM modules through the layers `sigma` and `mu`, and thanks to this configuration, can be learned. The GM can be viewed as an extension of the generator, and this configuration -, also known as the reparametrization trick -, enables backpropagation through the entire branch. Consequently, the mean and standard deviation of the Gaussians can be updated. As suggested by [1], σ_k is considered as a diagonal matrix. Both the `sigma` and `mu` layers are linear and have a size corresponding to the latent space.

1.2 Unsupervised Setting

The first approach implemented as described by [1] was the unsupervised setting. In this configuration, the GAN is trained as the vanilla GAN, where the discriminator aims to predict whether its input is real or fake. The Binary Cross Entropy (BCE) loss was then employed for the training of both the discriminator and the generator.

Subsequently, a decision was made to guide the model in generating diverse data by compelling the Gaussians to represent specific individual classes, transforming the problem into a supervised setting.

1.3 Supervised Setting

The learning approach for the previously described model has been chosen to be supervised, with the number K fixed at 10. This decision was made to ensure that each Gaussian in the latent space represents a distinct class of the dataset, with 10 classes in the case of MNIST.

In the supervised GM-GAN setting, one of the primary objectives is to train the discriminator to distinguish between real and fake samples, as well as to identify the class of real samples. Consequently, the generator is compelled to produce outputs classified into each of the classes.

The output of our discriminator is modified so that, instead of returning a single scalar representing the probability that the input is a real sample, it now returns a vector with 11 components. The first 10 components represent the probability of belonging to one of the classes, and the last one indicates whether it is a fake sample or not. This implementation is done directly in the file `model.py`.

$$\begin{cases} (0, \dots, 1^{\text{ith}}, \dots, 0) & \text{if the input is of class } i \\ (0, \dots, 0, 1) & \text{if the input is a fake sample} \end{cases}$$

However, in the implementation, the values of the discriminator's output can't be interpreted as probabilities. For stability reasons, the softmax activation function is not applied in the last layer of the discriminator but rather within the CrossEntropy Loss.

The CrossEntropy Loss has been used for the training of both the generator and the discriminator. This criterion computes the cross-entropy loss between input logits and targets:

$$\text{CrossEntropy}(x, y) = (l_1, .., l_n)^\top, l_n = -\sum_{c=1}^{C} \log \left(\frac{\exp(x_{n,c})}{\sum_{i=1}^{C} \exp(x_{n,i})} \right) \cdot y_{n,c}$$

This CrossEntropy loss was employed to train both the generator and the discriminator. The generator loss takes as an input $D(G(\widetilde{z}))$, representing the discriminator's prediction for the generated data, and as a target, the label k,corresponding to the original class of y_k from which $\widetilde{z}$ was sampled. The generator is thus forced to produce content that is correctly classified as true and belongs to the class from which it was sampled.

Simultaneously, the discriminator loss is the sum of the losses when having a real sample as input and when having a generated sample. The input $D(x_{real})$ is expected to be classified as y_{real} as per the MNIST dataset labeling. On the contrary, the input $D(G(\widetilde{z}))$ is expected to be classified as fake, and the label is thus set to $y_{generated} = 10$.

The distinction between these two learning processes is fundamental: the discriminator aims to classify generated images as fake, while the generator wants them to be classified as belonging to a specific class. This adversarial learning principle justifies that the learning is optimized when the discriminator is maximally confused and cannot decisively determine whether a generated image is fake or real. At this point, its output reaches a probability of 0.5 for being fake and 0.5 for being true for a specific class, a state known as the Nash Equilibrium.

The generator loss is backpropagated through both the Gaussian Mixture module and the generator module, while for the discriminator loss, it updates only the discriminator weights. All three modules were optimized using the Adam optimizer, employing various learning rates, which will be presented below.

The implementation can be found under the file `utils_supervised.py` on GitHub.

1.4 Alternative setting : objective separation

Based on the main supervised approach, we wanted to try an alternative one that would fully separate the training between a "(number) classification" objective, in which the discriminator must predict a one-hot encoding on the 10 classes for both real and generated data, and a "real/fake discrimination" objective on an eleventh output feature. Only the first 10 features are considered for the softmax and then sent to the CrossEntropy loss, and a sigmoid is applied to the 11^{th} number before having its Binary CrossEntropy loss computed with respect to a binary label representing that the sample is either fake or real.

After some testing, we decided not to compute the classification objective on the generated data when training the discriminator, since it hindered performance : on average, the "full loss" version had a FID of 350 after 5 epochs vs 260 for the chosen version, a gap that was not reducing after 100 epochs. Our interpretation is that the generated data quality was too poor for it to be relevant data for the classification part of the model to train on. However, the class of generated data is still very much used as an objective when training the generator and the Gaussian mixture, as in the main approach.

This approach has been developed under the `SeparateObjective` branch on GitHub.

2 Training and parameters

Once the implementation was up and running, the model had to be parameterized to be trained.

2.1 The unsupervised setting: discovery of the collapse mode

The unsupervised version of the Gaussian Mixture GAN was firstly trained with naive parameters and gave non various results that leads us to consider implementing the supervised version. Mode collapse occurs when the generator produces a limited number of samples, thus losing the diversity of the training dataset. This happens when the generator can only fool the discriminator with a specific set of data, and thus no longer produces diversity. In this specific case, the mode collapse was observed very early and the generator was able to produce only one constant random image.

The supervised version of the training algorithm offers the possibility to explicitly choose the category of generated data. It was anticipated that this approach would artificially aid in creating diversity in the generated outputs.

2.2 Parameter of Gaussian Mixture

In any of the implementations described, the weights of the Gaussian Mixture module represent sensitive parameters of our model. It is important to initialize them to prevent random initialization issues.

Following the proposal of [1], the standard deviation - all weights of the `sigma` layer - are set to a constant value σ_0, while the means - all weights of the `mu` layer - are randomly sampled from the interval $[-c, c]$. This is done in file `train.py`.

The static version of the Gaussian Mixture model, where the mean and standard deviation are not updated, as presented by [1], provides a solid starting point for our initialization. The standard deviation parameter remains particularly crucial, and it seems logical that if it is set to a small value (e.g., 0.1), the Gaussians will be thin, tall, and well-separated from each other. Conversely, if it is set to a larger value (e.g., 1.7), the distributions will be flatter and more mixed. The paper explains that with a small standard deviation, the results tend to be more qualitative but less diverse, while a higher deviation leads to more diverse results but of lower quality, which aligns with intuitive expectations. The goal is to find a suitable trade-off to achieve both qualitative and diverse generated images.

2.3 Learning rate and per-class mode collapse

As mentioned earlier, the learning rates of the Adam optimizers can be set separately for each module to control the rate of gradient descent learning.

The batch size can also be adjusted to either accelerate the training process or provide more accuracy to the gradient computation.

Despite numerous adjustments to these five parameters (three learning rates) and extensive training, the models consistently faced a persistent issue: mode collapse. This type of collapse differed from the one observed in unsupervised learning, as the classes were strictly distinguishable but internally collapsed. This implies that we could distinguish between nine different numbers, but within each class, they all appeared identical.

2.4 Dropout

Dropout is a regularization technique that aims to reduce correlation between latent features at any stage of a machine learning model. It works by "dropping" (setting to 0) individual features of the vector it is applied to, with probability $0 < p < 1$ (which is a hyperparameter of the model) independently for each feature. It is generally deactivated outside the training setting, aiming to enable training of "independent" networks then merged at inference time.

Dropout is embedded in the discriminator model that we use, and we wanted to test if using it on our 100-dimensional latent space obtained after the Gaussian Mixture would help mitigate mode collapse. It was not effective in that regard.

2.5 Weighted Loss

This improvement was specific to the approach presented in 1.4. The objective was to understand if the ratio between class prediction and reality prediction objectives was a key factor in avoiding mode collapse. We saw a very small improvement on results quality when the class prediction loss had a coefficient (1.5) slightly higher than that of the BCE loss. However, this very marginal change in behavior did not help with the mode collapse issue, and almost all other trials of weighting led to the model not even being to form shapes looking like numbers.

3 Results and Analysis

3.1 Mode Collapse

We conducted extensive training and experiments with various parameters to improve our models and address the issue of mode collapse. Unfortunately, none of the parameters setting were satisfactory in preventing it, we however managed to improve our FID to 100 at this stage. Even when the Gaussians parameters were initialized - with a high standard deviation for diversity - and frozen, the mode collapse keep occurring. A short extract is presented on Table 2 which highlights the lower bound our FID were stuck to because of the lack of diversity in the generated images. The model from the second approach suffers from this issue as well, and we can see in Figure 3 that the FID won't go below 100.

σ	c	learning rate	FID
1.7	3	8e-5	100
0.7	3	2e-4	150
1	3	2e-4	134
1	1	2e-4	210

Figure 2: Results after 150 epochs of training on supervised setting.

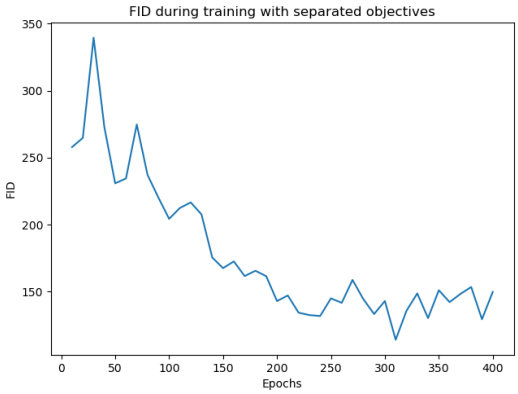

Figure 3: FID for separated losses model

We can analyse the mode collapse of the training by plotting the losses in Figures 4 and 5 of the initial supervised learning to understand this specific behaviour.

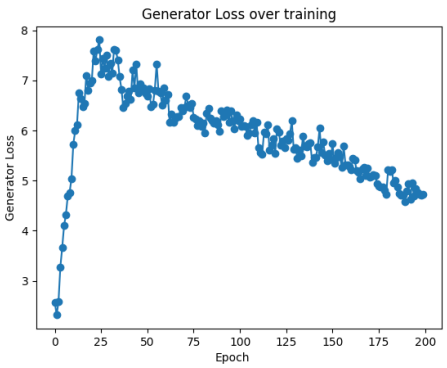

Figure 4: Generator loss depending on epochs.

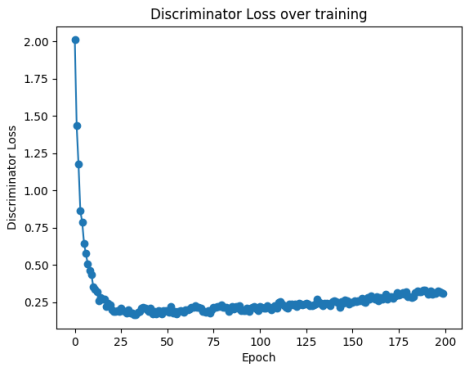

Figure 5: Discriminator loss depending on epochs.

We observe in Figure 5 that the discriminator loss decreases as expected as it is learning and becomes more and more powerful. However the generator loss increases a lot and remains stuck at a pretty high level meaning that the discriminator over trained and the generator becomes unable to counteract it. That is the typical response during the mode collapse observed in any of the training conducted with various learning rates.

3.2 Solution to the mode collapse

Changing learning rates between the modules (generator, discriminator and Gaussian Mixture) has had little effect until the GM learning rate was set to a very low value (`1e-7`) compared to those of generator and discriminator, on first supervised model presented in 1.3. The idea was to reduce the sensitivity of our Gaussian Mixture model to avoid making significant changes to the parameters especially at the beginning when the loss are pretty bad due to the quick start of the discriminator.

The new learning behavior - exit of the mode collapse - can be observed on the Figures 6 and 7, and significant results improvement are presented Table 1 and Figure 8. However, this only helped the main setting and did not solve the issue for the separated objectives model presented in 1.4.

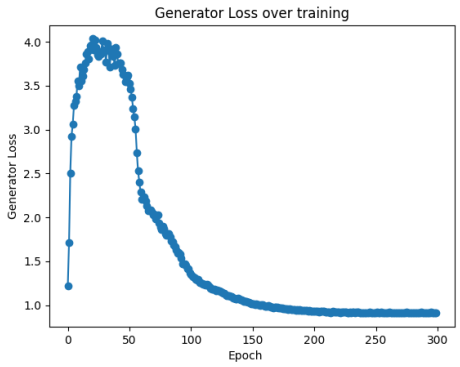

Figure 6: G loss on non collapsing model.

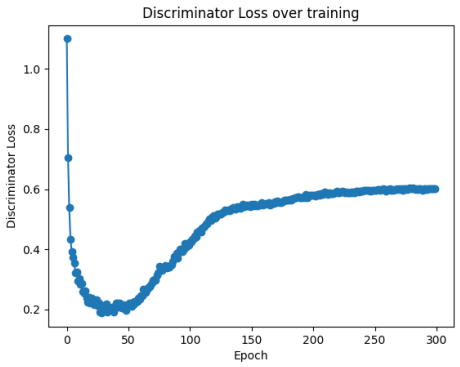

Figure 7: D loss on non collapsing model.

It is interesting to observe how the generator manages to counter the discriminator after around 50 epochs and even to lower its impact on its own loss as the maximum observed loss is of 4.0 against 7.5 when collapsing. The overall behavior shows strong signs of learning as the generator loss correctly decreased and then stabilized, while that of the discriminator increased and stabilized at 0.6 indicating that the discriminator is maximally confused.

σ	c	learning rate	GM lr	local FID	server FID	precision	recall
1.7	3	8e-5	1e-9	8.63	26.06	0.53	0.31
1.4	3	8e-5	1e-7	8.17	17.84	0.56	0.23
1.4	3	8e-5	1e-9	8.18	-	-	-
1.1	3	8e-5	1e-9	11.96	-	-	-

Table 1: Results after 300 epochs of training on supervised setting.

The Table 1 highlights the results dependencies on the initialization of the standard deviation σ. We observed as described in 2.2 that a smaller standard deviation will lead to a better precision, while a wider standard deviation produces a better recall.

4 Conclusion

It was interesting to work on a real neural network, especially the aspect of training it on the LAMSADE server using the SSH protocol and establishing a Git connection. However, adapting to a fixed structure posed some challenges, particularly when exploring alternative approaches to address the inherent instability of GANs. We looked into different training strategies and observed how they influenced the performance of a unique model.

In summary, our implementation of Gaussian Mixture model has been studied with different losses. We successfully addressed the challenge of mode collapse by generating diverse and high-quality digits. Additionally, we found that adjusting the standard deviation provides a means to control the trade-off between precision and recall. An idea for extending our study could involve incorporating Rejection Sampling, particularly effective when the generator consistently produces high-quality images.

References

[1] M. Ben-Yosef and D. Weinshall, "Gaussian mixture generative adversarial networks for diverse datasets, and the unsupervised clustering of images," 2018.

[2] T. Pandeva and M. Schubert, "Mmgan: Generative adversarial networks for multi-modal distributions," 2019.

A Additional material

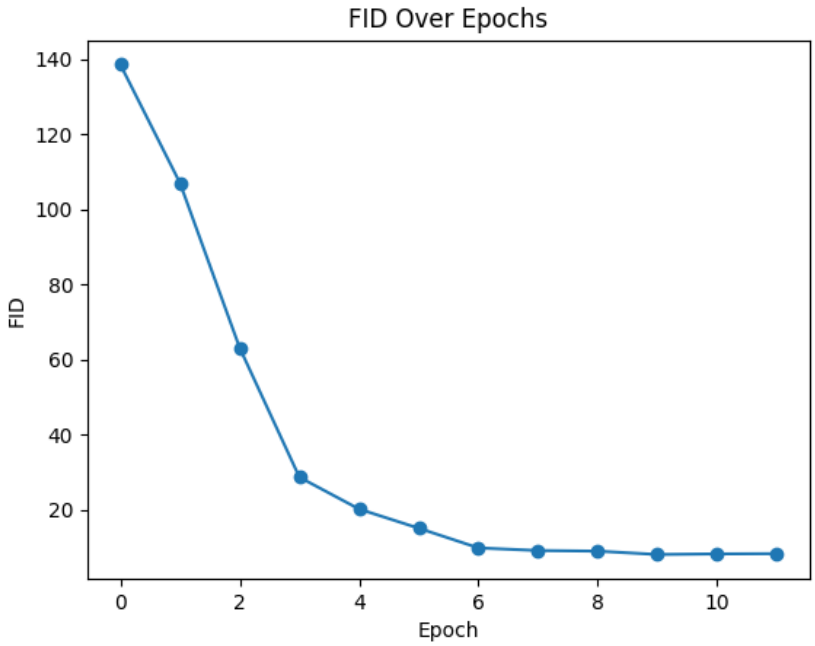

Figure 8: Locally calculated FID over training with $\sigma = 1.4$ and $lr_{GM} = 1e-9$ for the first supervised setting.