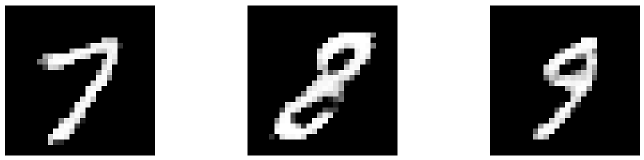
Gaussian Mixtures for Generative Adversarial Networks in Unsupervised Learning Scenarios

GANSTAS: Matteo Sammut, Alexandre Ngau, and Jules Merigot[1]

[1]PSL Research University - Data Science Project

November 2023



1 Introduction

Problem Statement: Enhancing Vanilla GANs for MNIST Dataset Using Gaussian Mixture Models in Unsupervised Learning Scenarios

Our Approach. In this project, we aim to enhance digit generation on the MNIST dataset using Gaussian Mixture GANs (GM-GANs), preceded by a hyperparameter optimization of vanilla GANs. Initially, we refined the vanilla GAN by experimenting with various hyperparameters to establish a robust baseline model. Upon achieving optimal settings, we transitioned to GM-GANs, testing specifically the static mode. This approach ensured we started with well-tuned hyperparameters for the Gaussian Mixture model. The project focuses on thoroughly fine-tuning the GM-GAN hyperparameters to surpass the performance of the vanilla GAN, emphasizing improvements in image quality and diversity in an unsupervised learning context.

2 Generative Adversarial Networks (GAN)

2.1 Vanilla GAN

In our approach to enhancing Generative Adversarial Networks (GAN) [1], we begin by focusing on the Vanilla GAN. This is a fundamental GAN architecture composed of two neural networks: the Generator and the Discriminator. The Generator creates synthetic data resembling the real dataset, while the Discriminator evaluates the authenticity of both real and generated data. This adversarial process drives both networks to improve iteratively until they converge and reach a Nash equilibrium [2].

Our objective is to meticulously fine-tune this vanilla GAN with optimal parameters, ensuring a robust and well-trained baseline model. This finely-tuned model serves as a crucial pretraining step. We will leverage this pretrained Vanilla GAN as the foundational model for our more complex Gaussian Mixture GANs. The Gaussian Mixture GANs are designed to capture more intricate data distributions, particularly useful in scenarios where data demonstrates multi-modal characteristics. By starting with a well-trained vanilla GAN, we ensure that our Gaussian Mixture GANs have a strong initial learning base, enhancing their ability to model complex datasets effectively. This methodical approach of pretraining with a vanilla GAN is expected to significantly improve the performance and stability of our Gaussian Mixture GANs. In the next section, we will briefly outline our hyper-parameter tuning process that allowed us to achieve the best possible vanilla GAN, as well as its results.

2.2 Exploring Hyper-parameters for Vanilla GAN

Before moving on to our main Gaussian Mixture model, we wanted to perfect the Vanilla GAN in order to start with well-tuned hyperparameters. This would allow us to then use a Gaussian Mixture to fine-tune the resulting latent space of our Vanilla GAN. For this, we decided to principally test various batch sizes as we noticed it had the largest impact on the convergence rate and metric results of our Vanilla GAN. We tested models with three different batch sizes and epochs, 32 for 100 epochs, 64 for 200 epochs, and 128 for 250 epochs. Our intuition for this was that a batch size of 64 could be the sweet spot, in which case we wanted to see how the model results would vary when this value is fluctuated. These were tested with a configuration including a learning rate of $\gamma = 2e - 4$.

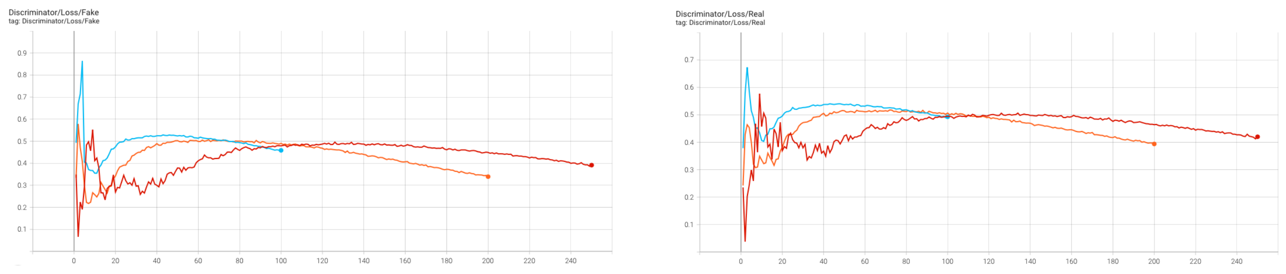


Figure 1: Discriminator Fake loss & Discriminator Real loss

As can be observed in the plots above, the plots of the loss on fake images for the discriminator and the loss on real images for the discriminator. We tested and plotted the results of the 3 batch sizes, with 32 as the blue line, 64 as the orange line, and 128 as the red line. We can see that the 64 batch stagnates for longer around a loss of 0.5, which implies more consistency and a better GAN. This clearly occurs around epoch 50.

Therefore, we decided to select a Vanilla GAN model with a configuration of batch size = 64, epoch = 100, and learning rate $\gamma = 2e - 4$. The table below displays the results of this model as calculated by the provided testing platform. It can be observed that the FID is relatively good, with a low score of 39.45, accompanied by a good precision score of 0.53 and recall of 0.23. Since recall is hard to increase for a Vanilla GAN, this lower number is expected but still considered to be decent in this case.

GAN	FID	Precision	Recall
Platform Vanilla GAN	39.45	0.53	0.23

Table 1: Testing Platform Metric Results of Vanilla GAN

When running this Vanilla GAN, we set checkpoints for every 10 epochs, allowing us to collect information on the model weights and relative latent space at that epoch. Using this, we set the epoch 50 checkpoint as the pre-trained baseline for our Gaussian Mixture GAN. Running a Gaussian Mixture from this baseline is similar to starting the training at epoch 50, and progressing from that point on to produce the best possible results. This will be further discussed in the next section.

3 Gaussian Mixture GAN

In our exploration of Generative Adversarial Networks (GANs), we delved into the realm of Gaussian Mixture GANs (GM-GANs), which stand out for their capability to model complex, multi-modal data distributions. Unlike the standard GAN architecture that use a standard Gaussian distribution for the latent space distribution, GM-GANs employ a mixture of Gaussian models. One way to visualize that, is to associate one modality to each Gaussian distribution. However, the user can adjust the number of Gaussian depending on the inter-variability and the intra-variability of the classes. This approach enables the generation of data that more accurately reflects the diverse and intricate patterns found in real-world datasets. To visualize the Gaussian distributions in a 3-dimensional perspective, we have provided an image to the right that visualizes the distributions relative to each other. We will explore in the next section the relevancy of their means and variances.

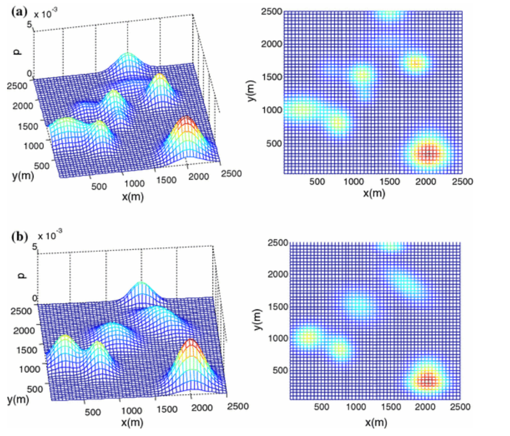


Figure 2: Gaussian Mixtures Visualization [3]

This report aims to detail our experimental findings with GM-GANs when applied to the MNIST dataset. We will detail our numerous experiments during which we meticulously finetuned the relevant hyperparameters in the search for the best possible final performance. As described per the previous section, for this we will be using a baseline of our pretrained Vanilla GAN, which will allow us to use a GM-GAN to start training from epoch 50 of the Vanilla GAN, and therefore finetune its latent space.

3.1 Unsupervised Static Gaussian Mixture GAN Implementation

Gaussian Mixtures can be implemented both as supervised and unsupervised techniques. In the supervised setting, the user associates each Gaussian to a class and the GM-GAN's discriminator is returning $N + 1$ elements in the range $[0, 1]$. Each element can be interpreted as the probability that the given sample is a real sample of class $i \in \{1, \ldots, N\}$ or the probability that it is a fake sample. However, it has been empirically shown (Ben-Yosef and al., 2018) [4] that the unsupervised setting has similar performance, where each Gaussian will naturally be associated to a class. While this is not an obvious occurence, it can be shown in practice due to the similar metric results, such as FID, accuracy, and recall scores. We will explore this later.

Concerning the means and the deviations of the Gaussian distributions, one can specify them like - the *static* mode - or directly learn them - *dynamic* mode. We implemented both methods in `model.py`. However, As our goal was to have a rigorous step-by-step methodology with properly tuned hyperparameters, we decided to focus on the *static* method, and did not have time to compare with the *dynamic* one.

One way to choose the means of the Gaussians, is to sample them uniformally from a compact set $[c, c]$ defined by the hyperparameter c. Concerning the variance, we could define a whole covariance matrix, a diagonal one, or a single scalar used for all Gaussians and all their dimensions. The later has the advantage of reducing the number of parameters used, and thus has an easier interpretation. When choosing these hyperparameters, one should pay attention to the chance of overlapping Gaussian distribution, and the consequences concerning the recall and precision.

4 Exploring Hyperparameters for Gaussian Mixture GAN

4.1 Initial Tests

In order to obtain the best GM-GAN based on the pre-trained Vanilla GAN (chosen in paragraph 2.2), it was crucial to select the best fine-tuning hyperparameters. The methodology we followed for that matter was two-fold, and consisted of two hyperparameters search, the first based on intuition, while the second was more focused on reproducing the results of [4]. As specified in paragraph 2.2, the different experiments were saved code-wise using checkpoints of the fine-tuned models every 10 epochs, and the evolution of our metrics saved in `tensorboard` logs, in order to better analyse and compare the results. The final Vanilla GAN pretraining as well as the Gaussian Mixture GAN finetuning logs can be found in the `runs/` folder in our GitHub repository along with the checkpoints and generated digit samples.

The hyperparameters that were optimized during our experiments are the following, which are also layed out in the Ben-Yosef and al., (2018) article [4], along with many others used in the algorithm. However, we decided to focus on the ones below.

- K - the number of Gaussians in the mixture,
- c - defines the range from which the Gaussians' means are sampled,
- σ - defines the scaling factor for the covariance matrices,
- γ - defines the learning-rate of the algorithm optimizer (in our case we used ADAM).

These were modified in the file `config.json` before each experiment, while the hyperparameter that optimized the pre-trained Vanilla GAN was kept unchanged, that is to say it was kept at a batch size of 64.

For our first set of hyperparameter search experiments, we performed 12 consecutive runs with hyperparameters chosen in specific ranges :

$$K \in \llbracket 10; 15 \rrbracket,\ c \in [1; 5],\ \sigma \in [0.001; 0.05],$$

$$\text{and } \gamma \in [5e-5; 8e-5].$$

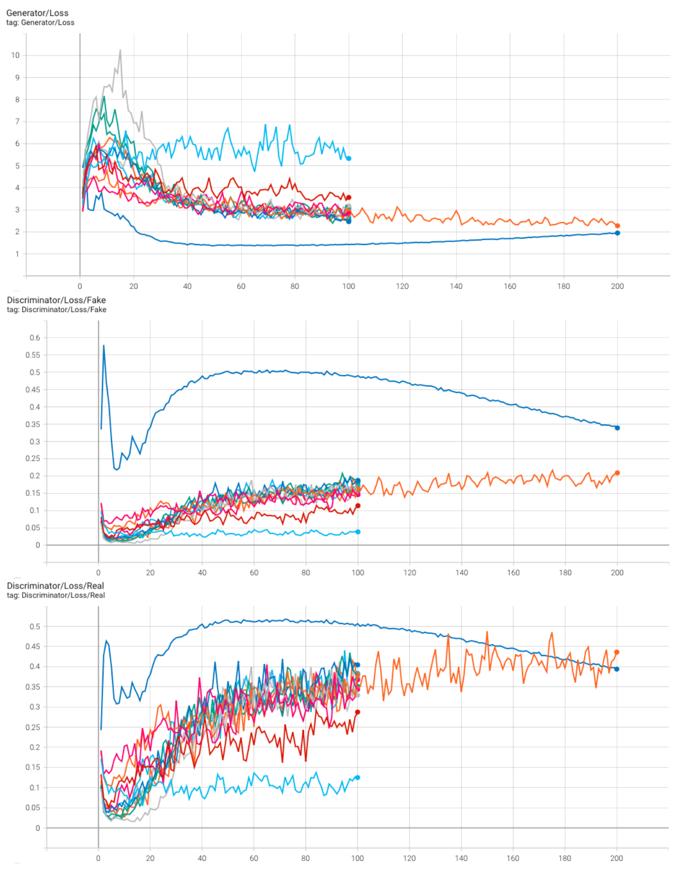
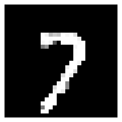
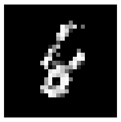
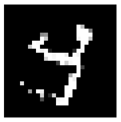


Figure 3: For our first models, the Generator loss, Discriminator loss for fake images, and Discriminator loss for real images (from top to bottom)

As can be seen in Figure 3, where the 12 experiments are plotted together, the general trend of the curves are quite off compared to the dark blue one, which corresponds to the training of the Vanilla GAN selected in paragraph 2.2. While the Vanilla GAN seems to have a stable and steady performance, the rest of the models are nowhere near it. This gave us an early indication that our initial hyperparameter selection may have been incorrect, which led us to finetune further. This will be detailed in the next subsection.

During these training runs, we computed metrics that allowed us to gain a better understanding of the model's performance while it is training. In our `train.py` file, we computed and printed at every epoch: the Generator loss, the Generator accuracy, the Discriminator loss on fake images, the Discriminator loss on real images, and the Discriminator accuracy. Additionally, we also computed the FID every 10 epochs using the implementation of the `pytorch-fid` library found at Pytorch FID GitHub.

The resulting generated sample images that can be seen above were however relatively discernible, but far from perfect. The lack of cohesion and abundance of random noise pushed us to restart our finetuning process with new hyperparameters in order to achieve a much better-performing Gaussian mixture GAN model.

4.2 Improved Hyperparameter Selection

As a consequence, the second set of hyperparameters search experiments revolved around trying to reproduce the paper's [4] results, where the best set of hyperparameters found was $K = 10,\ c = 0.1,\ \sigma = 0.15,$ and $\gamma = 2e - 4$.

Noting this, we set ourselves up to run four different experiments, that are plotted in Figure 4, all with a batch size of 64 and $K = 10$:

- RED: c$= 0.1, \sigma = 0.15,\ \gamma = 2e - 4$
- BLUE: c$= 0.1, \sigma = 0.15,\ \gamma = 8e - 05$
- PINK: c$= 0.467, \sigma = 0.7,\ \gamma = 2e - 4$
- GREEN: c$= 0.467, \sigma = 0.7,\ \gamma = 8e - 05$

The hyperparameters of the RED line being the paper's [4] best, the intuition behind the way we chose the other hyperparameters was determined by the need to keep the proper ratio between the values of c and σ, while trying a smaller learning rate γ than the one used during pre-training of the Vanilla GAN.

As can be seen in Figure 4, the four runs produced decent results, while still being imperfect: the Generator Loss decreases to a stalling minimum before increasing again, which signifies its learning and deteriorating phase, respectively. While the Discriminator Loss increases to a stalling maximum, its confusion is maximal, and then decreases after that. The RED curve is clearly the best, stalling at 0.5 for the Discriminator Loss around epoch 70, and following the global trend the rest of the time.

The RED hyperparameter configuration of a GM-GAN at epoch 70 was then selected as the best overall GAN trained during this assignment. The checkpoints of the training run of this optimal Gaussian mixture GAN were pushed to our GitHub repository and can be found in the `checkpoints` folder, as well as the samples generated with it in the `samples` folder.

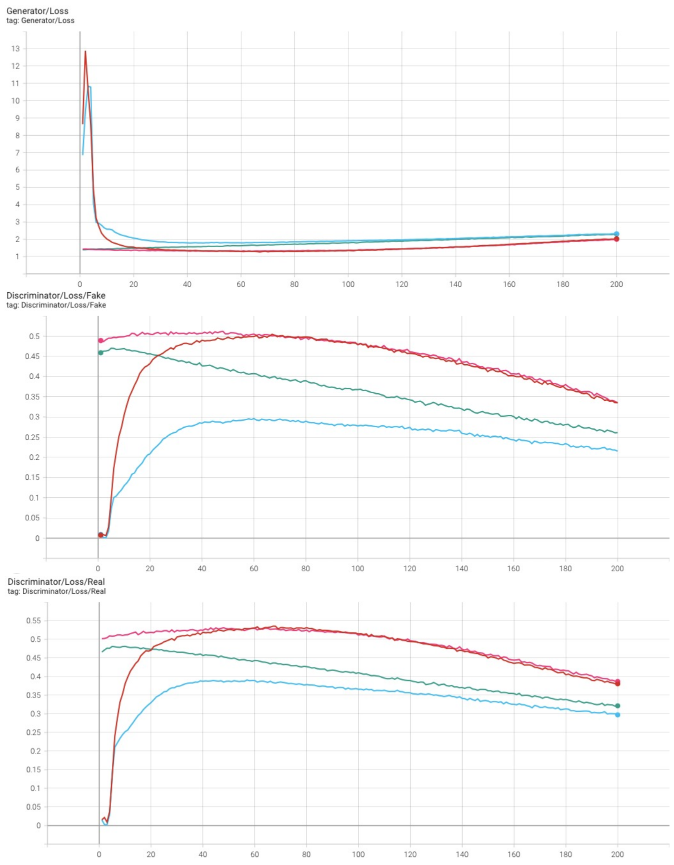
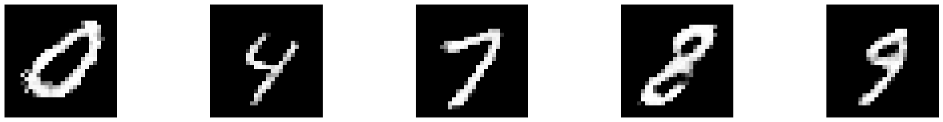
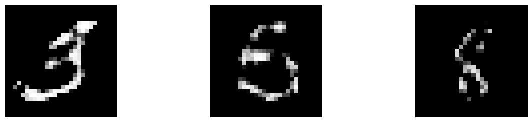


Figure 4: For our improved models, the Generator loss, Discriminator loss for fake images, and Discriminator loss for real images (from top to bottom)

Once we had fully run our improved Gaussian Mixture GAN model with the best hyperparameters, we could see the quality of the MNIST digits produced. Below are some examples of good-looking digits generated by our Gaussian Mixture GAN:



It is worth noting that while our Gaussian Mixture GAN model performed well and produced well-shaped and easily discernible numbers, it is not perfect or without flaws. It also generated some not so good looking digits. Below are some examples of digits that have much more noise, which makes it harder to make out the number present in the digit output.



However, this final Gaussian Mixture model using the recommended hyperparameters [4] proved to generate a higher frequency of better looking digits, and with a a higher variety than our initial tests.

5 Final Results

We can now add to our first table with the results from the Vanilla GAN. This is second table will include the FID, Accuracy, and Recall metric scores for our final statis Gaussian Mixture GAN when tested on the validation platform, as well when calculated on our local testing platforms.

GAN	FID	Precision	Recall
Platform Vanilla GAN	39.45	0.53	0.23
Platform Static GM-GAN	43.67	0.47	0.22
Local Static GM-GAN	19.02	-	-

Table 2: Results of our GANs and Gaussian Mixture Methods

We observed that when tested on the validation platform, our Gaussian mixture GAN did not perform as well as we expected compared to the Vanilla GAN. While our GM-GAN has a very good minimal FID score of 19.02 when tested locally on our servers, the final model did not produce the same results. This is most likely due to the difference in the amount of sample images being used to compute the FID score. While we used less due to computational limits, the validation platform tested with nearly 10,000 sample digits. This is something to take into consideration for future exploration, as well as the expansion of this project into the realm of the dynamic Gaussian mixture GAN.

6 Conclusion

In conclusion, our project successfully enhanced the MNIST dataset's digit generation using Gaussian Mixture GANs (GM-GANs), building upon a well-optimized vanilla GAN. Through rigorous hyperparameter tuning, we established a robust baseline with the vanilla GAN, which significantly informed our approach to refining GM-GANs in an unsupervised learning setting. Our experiments led to notable improvements in image quality and diversity, highlighting the efficacy of GM-GANs over traditional GAN architectures for complex data distributions. We've learned that meticulous hyperparameter optimization plays a crucial role in GAN performance, and the integration of Gaussian mixtures offers a substantial advancement in handling multi-modal data.

For future work, exploring dynamic learning modes in GM-GANs could further validate and enhance our findings. Additionally, implementing Discriminative Rejection Sampling (DRS) [5] could help refect the poor-looking digit images above that were produced by our optimal Gaussian mixture GAN.

References

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *ArXiv e-prints: arXiv:1406.2661*, 2014.

[2] J. Zhuang and M. A. Hasan, "Non-exhaustive learning using gaussian mixture generative adversarial networks," *Indiana University-Purdue University Indianapol. arXiv:2106.14344v2 [cs.LG]*, 2021.

[3] P. Yao, H. Wang, and H. Ji, "Gaussian mixture model and receding horizon control for multiple uav search in complex environment," *Nonlinear Dyn 88, 903–919. https://doi.org/10.1007/s11071-016-3284-1*, 2017.

[4] M. Ben-Yosef and D. Weinshall, "Gaussian mixture generative adversarial networks for diverse datasets, and the unsupervised clustering of images," *School of Computer Science and Engineering, The Hebrew University of Jerusalem, arXiv preprint arXiv:1808.10356*, 2018.

[5] S. Azadi, C. Olsson, T. Darrell, I. Goodfellow, and A. Odena, "Discriminative rejection sampling," *arXiv:1810.06758v3 [stat.ML]*, 2019.