Université Paris Sciences & Lettres

Data Science Lab A2 Report

Improvement methods of GANs

Authors:

Marwa Nair, Anna Krysta, Caio Rocha

Team: GANerators

November 12, 2024

Université Paris Sciences & Lettres
Data Science Lab A2

Improvement methods of GANs

1 Introduction

Generative Adversarial Networks (GANs) have shown considerable promise for high-quality image generation, yet they often face challenges such as mode collapse, vanishing gradients, and discriminator overfitting. This report investigates methods to enhance GAN performance on the MNIST dataset. Specifically, we focus on refining GAN-generated outputs through Latent Space Interpolation (LSI) and/or improving GAN training via Collaborative Sampling (CS) and Differentially Private GAN (DPGAN), with the goal of balancing FID, precision, and recall scores while addressing these issues.

2 Methods of improvement

2.1 Collaborative Sampling (CS)

Collaborative Sampling (CS) introduces a post-training enhancement for GANs, where a feedback loop between the generator and discriminator iteratively refines samples. The generator produces initial samples, which are then refined through gradient-based updates guided by the discriminator, bringing them closer to the real data distribution [1].

Methodology and Improvements CS involves two main steps: iterative sample refinement and discriminator shaping. The frozen generator produces samples, which are refined through discriminator-guided updates over 20–50 steps. Discriminator shaping further improves this by fine-tuning the discriminator on both real and refined samples, smoothing its loss landscape and improving sample quality. This process enables the generator to produce more realistic, diverse samples.

Algorithm The generated samples are continuously refined through Algorithm 1.

Algorithm 1 Collaborative Sampling

Require: Frozen generator G, frozen discriminator D, layer index l for sample refinement, maximum steps K, stopping criterion η
Ensure: a synthetic sample x

for $k = 0, 1, \ldots, K-1$ **do**
  if $D(x^k) < \eta$ **then**
    $x_l^{k+1} \leftarrow x_l^k - \lambda \nabla_l \mathcal{L}_G(x_l^k)$
    $x^{k+1} \leftarrow G_L \circ G_{L-1} \circ \cdots G_l(x_l^{k+1})$
  else
    break
  end if
end for

In Algorithm 1, G_l is the lth layer of the generator, x_l is the corresponding activation input, x is the output of the last layer, $\mathcal{L}_G$ is the generator loss. Collaborative sampling and discriminator shaping are performed alternatively, according to Algorithm 2.

The algorithm implementation is in the `train.py` file in "collab" mode. We used the SGD optimizer with a learning rate of 0.0002 for both the discriminator and generator, and batch size of 32. One epoch in collaborative sampling is one pass through the dataset, with K steps equal to the dataset size divided by the batch size. The stopping criterion η follows the probabilistic construction from the original paper.

Key hyperparameters The algorithm requires selecting a refinement layer, which impacts on the granularity of features; we fine-tuned the last layer. Additionally, it requires tuning the number of refinement steps for the discriminator shaping, which the authors suggest choosing between 20 and 50. We chose 50 refinement steps as it provided the best performance. We also tested performing CS for 1 epoch and for 10 epochs, in order to see the impact of fine-tuning the generator multiple times.

Algorithm 2 Discriminator Shaping

Require: Frozen generator G, pre-trained discriminator D, batch size m
Ensure: Fine-tuned discriminator $\tilde{D}$
1: **for** number of D shaping iterations **do**
2: Draw m refined samples $\{x_c^{(1)}, \ldots, x_c^{(m)}\}$ from the collaborative data distribution $p_c(x)$ according to Algorithm 1
3: Draw m real samples $\{x_r^{(1)}, \ldots, x_r^{(m)}\}$ from the real data distribution $p_r(x)$
4: Shape the discriminator by minimizing the composite loss of $\mathcal{L}_D(x_c)$ added with $\mathcal{L}_D(x_r)$
5: **end for**

Impact on the MNIST dataset CS improves MNIST synthetic data by enhancing quality and diversity, with the redefined discriminator preventing mode collapse and ensuring a more diverse sample set.

2.2 Differential private GAN

The **Differentially Private GAN (DPGAN)** is an adaptation of the standard GAN framework, integrating differential privacy into the discriminator to enable privacy-preserving synthetic data generation [2]. This modification ensures that the GAN can learn from sensitive data without compromising the privacy of individual data points. The DPGAN aims to address the trade-off between privacy guarantees and generation quality. In our implementation, we use the Opacus library to implement the differential private Discriminator, which is included in the `diff_privacy` mode in the `train.py` file.

Methodology and Improvements The DPGAN method incorporates differential privacy into the GAN training process with several key modifications. These changes specifically impact how gradients are processed in the Discriminator to achieve privacy.

To achieve differential privacy, the DPGAN method introduces the following steps:

- **Clipping Gradients to a Maximum Norm** C: Gradient clipping ensures that individual data points cannot disproportionately influence the model. Each gradient vector is scaled down to have a norm of at most C, which limits the sensitivity of each data point.
- **Adding Gaussian Noise to Discriminator Gradients**: To obscure the specific contribution of any individual data point, Gaussian noise with mean 0 and variance proportional to a noise multiplier n_m is added to the clipped gradients. This noise addition satisfies the (ϵ, δ)-differential privacy guarantee by reducing the likelihood of reverse-engineering sensitive data.
- **Performing More Discriminator Steps (n_D) per Generator Step**: To address the imbalance created by the noise-added gradients, the model performs multiple updates to the discriminator for each update to the generator. This compensates for the discriminator's weakened signal, allowing it to better guide the generator despite the added noise.

The modifications in DPGAN are fundamentally *training process improvements*, as opposed to architectural changes. This means that while the underlying structure of the GAN remains the same, the way gradients are computed and updated is adapted to meet privacy constraints, resulting in a balance between privacy and performance.

Specific Hyperparameters and Tuning Challenges Training a DPGAN effectively requires careful tuning of several critical hyperparameters, each affecting both the model's privacy guarantees and performance:

- **Batch Size** (B): Larger batch sizes reduce the variance in gradient estimates, allowing for more stable updates even with noise addition. However, larger batches also increase memory demands, and using very large batch sizes may limit the range of possible settings for other parameters.
- **Learning Rate** (α): The learning rate influences the speed and stability of convergence. In DPGAN, adjusting the learning rate is crucial, as higher noise levels often require a slower learning rate to prevent destabilization in the training dynamics.
- **Discriminator Steps per Generator Step** (n_D): Increasing n_D (the number of discriminator steps for each generator step) helps maintain balance between the generator and discriminator. The added noise weakens the

discriminator's feedback to the generator, and by increasing n_D, the model compensates for this weakened signal, allowing the discriminator to retain its role as a reliable adversary.

- **Clipping Norm** (C): The clipping norm controls the maximum magnitude of each gradient update, thereby limiting individual contributions to the model. A higher clipping norm could allow more information per data point to influence the model, but it requires adding proportionally more noise to satisfy the privacy guarantees.
- **Noise Multiplier** (n_m): The noise multiplier controls the amount of noise added to the discriminator's gradients. Higher values increase privacy but also reduce the discriminator's effectiveness. In practice, finding an optimal noise multiplier that balances privacy and utility is one of the most challenging aspects of DPGAN.

These parameters interact in complex ways. For instance, increasing n_D can partially offset the negative impact of a high noise multiplier, while adjusting the batch size may necessitate changes to the learning rate for stable convergence. Fine-tuning these parameters is computationally expensive, as it involves a balance between ensuring sufficient privacy and maintaining generation quality.

Results and Challenges In our implementation, the FID (Frechet Inception Distance) score—a metric for evaluating the quality and diversity of generated images—was approximately 307.77. We tested several values for n_D from the set $\{1, 2, 5, 10, 20, 50, 100\}$, using a learning rate of 0.0002, batch size of 64, 100 epochs, noise multiplier $m_n = 1.0$, and clipping norm $C = 1.0$. The best performance was achieved with n_D equal to 10. Although this result did not show significant improvement over non-private GANs, several challenges influenced this outcome:

- **Non-Ideal Architecture**: The current implementation uses a linear architecture, while convolutional neural networks (CNNs) are known to perform better for image generation tasks, as recommended by [2].
- **Tuning Complexity**: The differential privacy constraints make parameter tuning time-consuming, as each change requires recalibrating the balance between noise level and gradient effectiveness.

How DPGAN Can Improve MNIST Classification While tuning difficulties impacted our initial FID results, DPGAN still holds significant promise for improving MNIST classification performance. The increased privacy protections ensure that synthetic data generated for training classifiers does not compromise individual privacy, while the modifications to the training process help capture essential patterns in the data:

- **Enhanced Discriminator Feedback**: By increasing the discriminator steps, DPGAN mitigates the noise's negative effects, resulting in a more robust generator. This improvement is particularly useful in MNIST, where distinguishing between classes relies on clear feature patterns, which the discriminator can enforce through multiple updates.
- **Improved Sample Diversity**: DPGAN reduces mode collapse (where the generator outputs limited variations), a common issue in GANs. By balancing the discriminator and generator updates, the model is less likely to produce identical or highly similar samples, which benefits classification tasks by providing a diverse set of training examples.
- **Privacy-Compliant Data Augmentation**: The synthetic data generated by DPGAN is privacy-compliant, meaning it can augment real datasets without privacy concerns. For MNIST, this synthetic data can help diversify the training dataset, potentially improving classifier robustness.

DPGAN represents a meaningful advancement in privacy-preserving machine learning. By integrating differential privacy into the GAN training process, it enables data synthesis that protects individual privacy while still capturing essential patterns for downstream tasks. Although challenges remain in fine-tuning and architecture optimization, the potential of DPGAN for applications like MNIST classification remains strong.

2.3 Latent Space Interpolation (LSI)

To enhance the quality of generated images, we explored an approach called Latent Space Interpolation (LSI), rather than relying solely on direct sampling of latent vectors. In the provided vanilla GAN sampling method, random vectors are drawn from a Normal distribution in the latent space and passed directly to the generator. However, this method can yield inconsistent results due to the irregular coverage of the latent space, potentially leading to low-quality outputs.

LSI aims to address this limitation by interpolating between pairs of randomly sampled latent vectors. This approach is inspired by research [3], [4] that suggests interpolation in the latent space facilitates more coherent transitions and high-quality generations, as it explores latent regions between sampled points rather than isolated points alone. By interpolating, LSI encourages explores more regions of the latent space, increasing the likelihood of discovering regions corresponding to realistic and well-defined images.

In LSI, two randomly selected (sampled from a Normal distribution) latent vectors, z_1 and z_2, are linearly combined at intervals along the interpolation parameter $\alpha \in [0, 1]$, producing a sequence of interpolated latent vectors z such that:

$$z = \alpha z_1 + (1 - \alpha) z_2$$

Each intermediate vector z is then passed through the generator.

Algorithm 3 Latent Space Interpolation (LSI) Generation

Require: Generator model G, total number of samples $num\_samples$, interpolation steps $interpolation\_steps$

Initialize $generated\_images = []$
for $i = 1$ to $num\_samples/interpolation\_steps$ **do**
  Sample two latent vectors $z_1, z_2 \sim \mathcal{N}(0, 100)$
  for α in $np.linspace(0, 1, interpolation\_steps)$ **do**
    $z = \alpha \cdot z_1 + (1 - \alpha) \cdot z_2$
    Append $G(z)$ to $generated\_images$
  end for
end for
return $generated\_images$

LSI effectively broadens the exploration of the latent space by traversing paths between sampled points, producing image samples that demonstrate smoother transitions and more consistent image quality. This technique, therefore, supports a more comprehensive assessment of the model's capacity to generate high-quality images from the latent space.

Key Hyperparameters The effectiveness of LSI relies on a single hyperparameter: $interpolation\_steps$, which specifies the number of intermediate points generated between two latent vectors, thereby influencing the smoothness and granularity of the interpolation. We evaluated values of 5 and 10 locally and selected 10 for the testing platform, as it achieved better overall scores.

A Variant of LSI We also implemented a variant of the standard LSI approach. In this variant, instead of interpolating between only two latent vectors, we generate a convex combination of 10 randomly sampled latent vectors. The combination of these vectors is weighted using a Dirichlet distribution to ensure that the weights sum to 1.

Algorithm 4 Variant of Latent Space Interpolation (LSI) Generation

Require: Generator model G, total number of samples $num\_samples$, interpolation steps $interpolation\_steps$

Initialize $generated\_images = []$
for $i = 1$ to $num\_samples/interpolation\_steps$ **do**
  Sample 10 latent vectors $z_1, z_2, \ldots, z_{10} \sim \mathcal{N}(0, 100)$
  for $j = 1$ to $interpolation\_steps$ **do**
    Generate random weights $w_1, w_2, \ldots, w_{10} \sim \text{Dirichlet}(\mathbf{1})$
    $z = \sum_{i=1}^{10} w_i \cdot z_i$
    Append $G(z)$ to $generated\_images$
  end for
end for
return $generated\_images$

This variant further enriches the exploration of the latent space by introducing a more diverse set of latent vector combinations, potentially resulting in more diverse image generation.

3 Results

Final results of the implemented methods are presented in Table 1. CS does not improve the results compared to VanillaGAN, especially fine-tuning the generator for one epoch. However, applying CS with 5 epochs with LSI consistently yields better results in terms of FID and precision. The best performance is achieved with CS with 5 epochs + LSI-variant, which shows the efficiency of performing LSI with multiple latent vectors.

	FID	**Precision**	**Recall**
Trained VanillaGAN	26.84	0.53	**0.25**
CS with 1 epoch	27.11	0.53	0.23
CS with 5 epochs	26.82	0.53	0.23
Trained VanillaGAN + LSI	21.81	0.57	0.16
CS with 1 epoch + LSI	22.05	0.56	0.18
CS with 5 epochs + LSI	**21.35**	0.57	0.19
CS with 5 epochs + LSI-VARIANT	21.88	**0.58**	0.18

Table 1: *Results of the implemented algorithms.*

4 Conclusion

We tested methods to improve GAN performance, including Collaborative Sampling (CS), which refines samples through discriminator feedback, Differentially Private GAN (DPGAN), which protects individual data while generating high-quality synthetic data, and Latent Space Interpolation (LSI), which smooths latent space transitions. The best results were achieved by combining CS with LSI, particularly with the LSI-variant, highlighting the importance of combining multiple approaches to achieve more realistic and diverse data.

References

[1] Y. Liu, P. Kothari, and A. Alahi, "Collaborative sampling in generative adversarial networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 4948–4956, Apr. 2020.

[2] A. Bie, G. Kamath, and G. Zhang, *Private gans, revisited*, 2023. arXiv: 2302.02936 [cs.LG].

[3] A. Radford, L. Metz, and S. Chintala, *Unsupervised representation learning with deep convolutional generative adversarial networks*, 2015. arXiv: 1511.06434 [cs.LG].

[4] T. Karras, S. Laine, and T. Aila, " A Style-Based Generator Architecture for Generative Adversarial Networks," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 43, no. 12, pp. 4217–4228, Dec. 2021.