Assignment 2 Report

Generative Adversarial Networks

Data Science Lab

IASD 2024-2025

extravaGAN:

Sabina Askerova, Yannis Fontaine, Newman Chen

Introduction

A Generative Adversarial Network (GAN) is a framework that simultaneously trains two adversarial models, a generator and a discriminator. The generator works by taking in a random input, a 'latent vector,' which is transformed by the model to create data that resembles the training data. This 'fake' data that is generated is then passed to the discriminator, whose job is to discriminate between 'fake' and 'real' data. Through the minmax interaction between these two models, we can simultaneously train both, and eventually are able to use this trained generator to generate new fake samples that resemble the original data. In our case, we choose to use the MNIST dataset, with the goal of ultimately generating a set of 10,000 images that represent handwritten digits. This dataset is then evaluated by the Fréchet Inception Distance (FID), as well as precision and recall. For our project we explored the following methods of improving upon the 'vanilla' version of a GAN.

WGAN

One of the most popular and influential improvements is the Wasserstein GAN (WGAN), created by Arjovsky et al. in 2017. This variant aims to improve on many of the shortcomings of vanilla GAN—namely, a new distance metric is introduced which removes the need to balance the training between the discriminator and generator. As a result, WGAN claims to eliminate the problem of mode collapse, which can be common for vanilla GANs and also allows for the training of the discriminator all the way to optimality, which can be shown by plots of the continual estimation of said distance.

The paper uses the **'Earth-moving' or Wasserstein distance**, which is calculated as follows:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\big[\, \|x - y\| \,\big]$$

The $\gamma(x, y)$ here represents the amount of 'earth' that needs to be moved from one distribution to the other to make them identical, with the Earth-moving distance being the cost of the optimal transport plan. The binary cross entropy loss from vanilla GAN, unlike this EM distance, does not measure the distance between the two distributions and is not as interpretable. This makes this loss function better to optimize, and one we can train the discriminator on until optimality.

For our implementation, we mostly followed the algorithm detailed in the paper, which is as follows:

```
**while** $\theta$ has not converged **do**
    **for** $t = 0, ..., n_{\text{critic}}$ **do**
        Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
        Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
        $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$
        $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
        $w \leftarrow \text{clip}(w, -c, c)$
    **end for**
    Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
    $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
    $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
```

Some changes we made from the start are that instead of RMSProp, we used Adam, which was already used for the given implementation of vanilla GAN. Furthermore, we chose to skip weight clipping from the start, a method that clips the weights if it goes outside of the range of [-c, c]. The paper itself describes it as "a clearly terrible way to enforce a Lipschitz constraint;" so we instead opt for a gradient penalty variant which was proposed later in "Improved Training of Wasserstein GANs" by Ishaan Gulrajani. The variant is described in the following section.

Gradient Penalty variant (WGAN-GP)

Instead of clipping the weights, a later more optimized solution proposes to instead enforce a gradient penalty on the discriminator. The loss function is as follows:

$$L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$$

Adding this gradient penalty, like weight clipping, helps to enforce the 1-Lipschitz constraint of the discriminator, but in a much more stable/smooth way. This allows for better convergence since it is more stable/smooth, which also helps to ensure less mode collapsing. After some experimentation and trials, while we did notice some significant improvement from vanilla GAN, we often still ran into issues with mode collapse/vanishing gradients. We experimented with changing the beta values of Adam, λ values for the weight of the gradient penalty, and 'n_critic', or the ratio of training loops we perform for the discriminator to the training loops we do for the generator. At times, even after finishing training and tinkering with these hyperparameters, we would notice that our output would almost completely resemble random noise. One thing that did help was changing the learning rate from the default 0.0002 to 0.0004.

LSGAN

Following research on different methods we could implement to start with, we decided on Least Squares GAN (LSGAN), first theorized by Mao et al. in "Least Squares Generative Adversarial Networks," which uses a **least squares loss function** rather than the binary cross entropy loss implemented by default. LSGAN is otherwise identical to vanilla GAN in terms of the rest of its architecture/algorithm. Mao et al. decided on this approach after seeing the vanishing gradient problem which occurs during the learning process, which involves using the cross entropy loss function for the vanilla GAN. We are left with the following objective function:

$$\begin{aligned}\min_D V_{\text{LSGAN}}(D) &= \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[(D(\boldsymbol{x})-1)^2\right] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z})))^2\right]\\ \min_G V_{\text{LSGAN}}(G) &= \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z}))-1)^2\right].\end{aligned}$$

Mao et al. demonstrate that minimizing this least squares loss is equivalent to minimizing the Pearson χ^2 divergence, which quantifies how one probability distribution differs from another. This in turn leads to a more stable training process as well as higher quality images compared to vanilla GAN. Though on the theoretical basis, LSGAN is flawed in that it does not completely eliminate the vanishing gradient problem and also still provides an opportunity for mode collapse, our results were very promising even beginning from our initial tests. That being said, we wanted to find improvements for these issues.

In order to continue to combat these problems, we decided to combine pieces from the other methods we had implemented.

CN-LSGAN

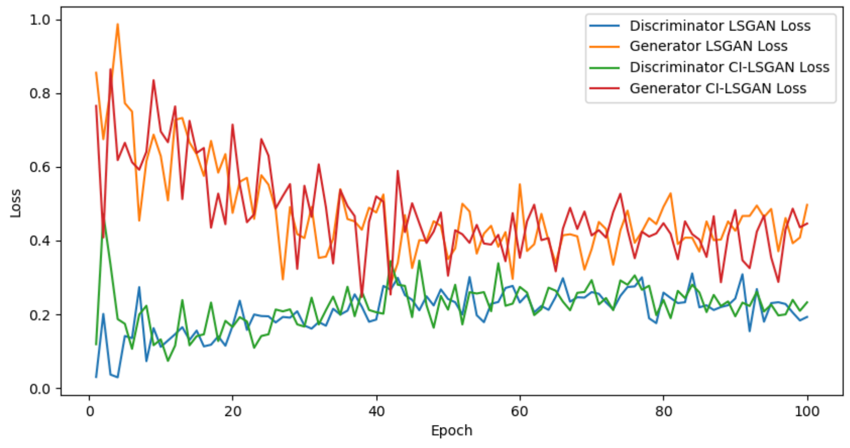
First, we tried out **adding on a gradient penalty**, something we had done with WGAN-GP, but also experimented with a variant in which we add a gradient penalty to the generator as well. Our objective function became the following:

$$\begin{cases}\min_D V(D,G) = E_{x\sim P_{\text{data}}(x)}\left[(D(x)-1)^2\right] + E_{z\sim P_z(z)}\left[(D(G(z)))^2\right] + \lambda_1 E_{\hat{x}\sim P_{\hat{x}}}\left[\left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\right)^2\right],\\ \min_G V(D,G) = E_{z\sim P_z(z)}\left[(D(G(z))-1)^2\right] + \lambda_2 E_{z\sim P_z(z), x\sim P_{\text{data}}(x)}\left[(G(z)-x)^2\right].\end{cases}$$

This change helps to build on the strengths of LSGAN by limiting the strength of the discriminator, which in turn helps with the vanishing gradient problem and encourages smoother training. Additionally, a discriminator that is kept in check discourages mode collapse, which can result when the generator continually provides limited, repetitive inputs to trick the discriminator which has become disproportionately trained.

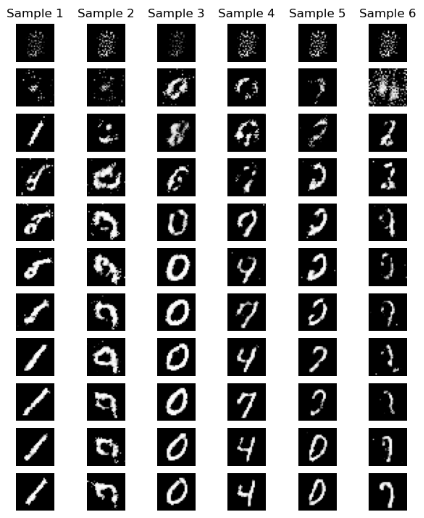
The hyperparameters that give us the best result are $\lambda_1 = 0.01$ (the regularization factor for the discriminator) and $\lambda_2 = 0.001$ (the regularization factor for the generator). Though we noticed some improvements on precision/recall when testing locally, improving by 2-3 hundredths, we were ultimately unable to consistently recreate these changes on the platform.

Here are the different losses of LSGAN and CN-LSGAN:

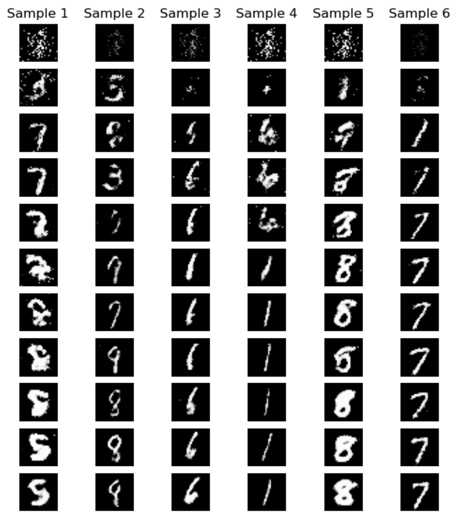


Loss of Discriminator and Generator over epochs for LSGAN and CN-LSGAN

We can see that the losses are almost the same. And here is some generation through epochs for both models:



Generation for LSGAN every 10 epochs



Generation for CN-LSGAN every 10 epochs

Discriminator Rejection Sampling (DRS)

In order to improve each method, we implemented the discriminator rejection sampling. The main idea behind DRS is to use the discriminator scores to better select the generator samples that more closely resemble the real data, rejecting those that are less realistic. This results in an output distribution that is more faithful to the actual data distribution, without the need to retrain the generator. Before starting selection, an estimation phase is used to calculate an approximate maximum value of the discriminator score for the generated samples. This is done by generating a large number of samples and obtaining the maximum of the discriminator scores applied to the samples. This maximum score is called D^{*}_{M}

Sampling by reject :

- For each sample generated by the generator, the discriminator score is used to calculate an acceptance ratio.
- A dynamic threshold, M_{bar}, is progressively adjusted during this phase to account for new maximum values encountered.
- The probability of acceptance of each sample depends on this score and M_{bar}: samples whose score is closer to the maximum

value have a higher probability of acceptance.

Calculation of acceptance probability :

The filter is defined by applying a sigmoid function to a function F(x), calculated as a function of the discriminator score and a dynamic threshold. This acceptance probability ensures only samples with higher discriminator scores are accepted, i.e. samples of better quality. To ensure that a generated sample is accepted, its acceptance probability must exceed a threshold, which is drawn from a uniform distribution between 0 and 1. The argument for using a probabilistic threshold (drawn from a uniform distribution between 0 and 1) instead of a fixed threshold is to allow flexible probabilistic selection, making sample rejection less rigid.

A fixed threshold risks reducing the diversity of accepted samples by focusing only on those with the highest discriminator scores.

By using a probabilistic threshold based on a uniform distribution, the model accepts higher-quality samples more often, while occasionally accepting more diverse samples. This improves sample diversity without sacrificing average quality.

The probability of acceptance based on a comparison with a uniform number makes the rejection procedure less sensitive to extreme values. Instead of having a threshold that might restrict or relax the selection too much, this probabilistic threshold makes the selection progressive, thus stabilizing the quality of the retained samples.

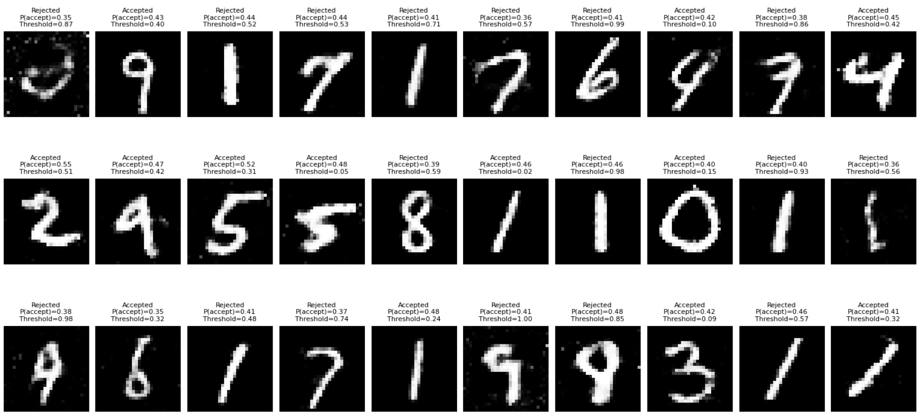
The DRS algorithm is:

```
Data: generator G and discriminator D
Result: Filtered samples from G
$D^* \leftarrow$ KeepTraining($D$);
$\bar{M} \leftarrow$ BurnIn($G$, $D^*$);
$samples \leftarrow \emptyset$;
while $|samples| < N$ do
    $x \leftarrow$ GetSample($G$);
    $ratio \leftarrow e^{\widetilde{D}^*(x)}$;
    $\bar{M} \leftarrow$ Maximum($\bar{M}$, $ratio$);
    $p \leftarrow \sigma(\hat{F}(x, \bar{M}, \epsilon, \gamma))$;
    $\psi \leftarrow$ RandomUniform($0,1$);
    if $\psi \leq p$ then
        Append($X$, $samples$);
    end
end
```

We compute F as follows : $\hat{F}(x) \;=\; \widetilde{D}^*(x) - \widetilde{D}^*_M - \log(1 - e^{\widetilde{D}^*(x) - \widetilde{D}^*_M - \epsilon}) - \gamma$

where ε is a small constant added for numerical stability and γ is a hyperparameter modulating overall acceptance probability. For very positive γ, all samples will be rejected. For very negative γ , all samples will be accepted. Here, we choose γ dynamically for each batch, to the 95th percentile of F(x) for all x in the batch, as the paper suggests.

Here is some example of applying DRS on CI-LSGAN:


Rejected P(accept)=0.35 Threshold=0.87
Accepted P(accept)=0.43 Threshold=0.40
Rejected P(accept)=0.44 Threshold=0.52
Rejected P(accept)=0.44 Threshold=0.53
Rejected P(accept)=0.41 Threshold=0.71
Rejected P(accept)=0.36 Threshold=0.57
Rejected P(accept)=0.41 Threshold=0.99
Accepted P(accept)=0.42 Threshold=0.10
Rejected P(accept)=0.38 Threshold=0.86
Accepted P(accept)=0.45 Threshold=0.42
Accepted P(accept)=0.55 Threshold=0.51
Accepted P(accept)=0.47 Threshold=0.42
Accepted P(accept)=0.52 Threshold=0.31
Accepted P(accept)=0.48 Threshold=0.05
Rejected P(accept)=0.39 Threshold=0.59
Accepted P(accept)=0.46 Threshold=0.02
Rejected P(accept)=0.46 Threshold=0.98
Accepted P(accept)=0.40 Threshold=0.15
Rejected P(accept)=0.40 Threshold=0.93
Rejected P(accept)=0.36 Threshold=0.56
Rejected P(accept)=0.38 Threshold=0.98
Accepted P(accept)=0.35 Threshold=0.32
Rejected P(accept)=0.41 Threshold=0.48
Rejected P(accept)=0.37 Threshold=0.74
Accepted P(accept)=0.48 Threshold=0.24
Rejected P(accept)=0.41 Threshold=1.00
Rejected P(accept)=0.48 Threshold=0.85
Accepted P(accept)=0.42 Threshold=0.09
Rejected P(accept)=0.46 Threshold=0.57
Accepted P(accept)=0.41 Threshold=0.32

One example of using DRS on vanilla GAN: FID: 29.68 → 28.89 / Precision: 0.51 → 0.51 / Recall: 0.24 → 0.26 (On testing platform)

Improved Consistency regularization (ICR)

While Rejection Sampling significantly improved our metrics, it requires substantial computational resources, as it discards some generated samples. We aimed to achieve similar performance with less computational cost, which led us to explore consistency regularization (CR). In representation learning, CR is a technique that enforces neural networks to maintain consistent predictions even when input data is augmented. For GANs, CR has been shown to improve performance by promoting similarity in model outputs for original and augmented inputs, balancing robustness with diversity in generated samples. Furthermore, CR is generally less computationally intensive than Rejection Sampling.

Since both VanillaGAN and our LSGAN sample from a random Gaussian distribution in the latent space, a natural question arises: how could we introduce prior knowledge into the latent space to enable a GAN model to produce more consistent outputs?

Improved Consistency Regularization (ICR) introduces two modifications to standard CR: Balanced Consistency Regularization (bCR) and Latent Consistency Regularization (zCR) (Zhang et al., 2020). These techniques add consistency loss terms to both the generator and discriminator, reducing sensitivity to small perturbations in data and latent spaces.
- **Balanced Consistency Regularization (bCR)** focuses on image consistency by applying transformations to both real and generated images. This approach encourages the discriminator to produce similar outputs for both original and augmented images, avoiding the bias observed in standard CR (Zhang, 2019) where only real images were augmented, causing the discriminator to favor certain features. By augmenting generated images as well, the discriminator learns more robust feature representations, and the generator avoids introducing unwanted artifacts.
- **Latent Consistency Regularization (zCR)** targets latent space consistency by introducing small noise to latent vectors fed to the generator. This helps encourage diverse outputs by producing slightly different images for similar latent codes, thereby reducing mode collapse. In practice, the generator receives both the original and the perturbed latent vectors, producing two distinct images. By maximizing the difference between outputs G(z) and G(T(z)) (where T(z) represents the perturbed latent vector), we aim to encourage as much diversity as possible in generated images from closely related latent codes. Our hypothesis is that introducing prior knowledge about the image set in the form of augmentations will yield improved precision and recall.

We implemented the algorithms from "Improved Consistency Regularization for GANs" (Zhao et al., 2020). The authors reported an FID of 15.87 with ICR applied to DCGAN on the CIFAR-10 dataset. We adapted their method to our LSGAN. In our experiments, we applied horizontal flips as the augmentation for both real and fake images. For the noise of latent space images, we used a noise factor sigma of 0.03. The λ_real=10, λ_fake=10, λ_dis=5, λ_gen=0.5

Algorithm 1 Balanced Consistency Regularization (bCR)

Input: parameters of generator θ_G and discriminator θ_D , consistency regularization coefficient for real images λ_{real} and fake images $\lambda fake$, augmentation transform T (for images, e.g. shift, flip, cutout, etc).

```
for number of training iterations do
    Sample batch $z \sim p(z)$, $x \sim preal(x)$
    Augment both real $T(x)$ and fake $T(G(z))$ images
    $LD \leftarrow D(G(z)) - D(x)$
    $L_{real} \leftarrow ||D(x) - D(T(x))||_2$
    $L_{fake} \leftarrow ||D(G(z)) - D(T(G(z)))||_2$
    $\theta_D \leftarrow AdamOptimizer(LD + \lambda_{real} * L_{real} + \lambda_{fake} * L_{fake})$
    $L_G \leftarrow -D(G(z))$
    $\theta_G \leftarrow AdamOptimizer(L_G)$
end for
```

Algorithm 2 Latent Consistency Regularization (zCR)

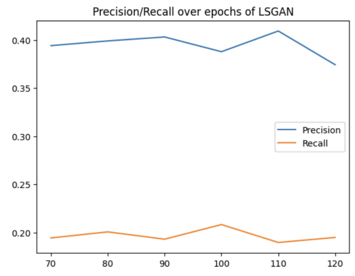
Input: parameters of generator θ_G and discriminator θ_D , consistency regularization coefficient for generator λ_{gen} and discriminator λ_{dis} , augmentation transform T (for latent vectors, e.g. adding small perturbation noise~ $N(0, \sigma_{noise})$).

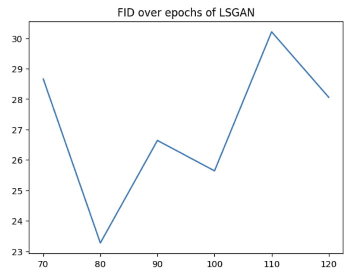
```
for number of training iterations do
    Sample batch $z \sim p(z)$, $x \sim preal(x)$
    Sample perturbation noise $\Delta z \sim N(0, \sigma_{noise})$
    Augment latent vectors $T(z) \leftarrow z + \Delta z$
    $L_D \leftarrow D(G(z)) - D(x)$
    $L_{dis} \leftarrow ||D(G(z)) - D(G(T(z)))||_2$
    $\theta_D \leftarrow AdamOptimizer(L_D + \lambda_{dis} L_{dis})$
    $L_G \leftarrow -D(G(z))$
    $L_{gen} \leftarrow -||G(z) - G(T(z))||_2$
    $\theta_G \leftarrow AdamOptimizer(L_G + \lambda_{gen} L_{gen})$
end for
```

Results

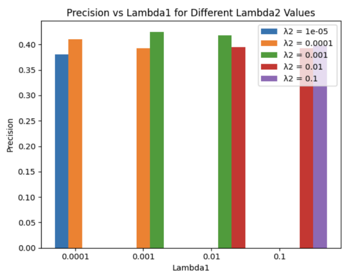
WGAN: alpha = 0.0004, beta = [0.5, 0.9], $\lambda = 10$, n_critic = 5, FID 42.21, precision 0.54, recall 0.25.

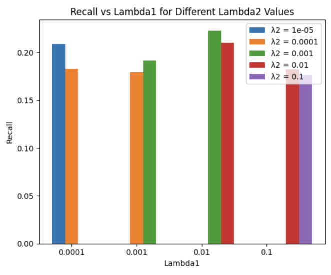
LSGAN: alpha = 0.0004, otherwise default hyperparameters FID 24.15, precision 0.56, recall 0.24.





CN-LSGAN: alpha = 0.0004, $\lambda 1 = 0.01$, $\lambda 2 = 0.001$ FID 27.9, precision 0.52, recall 0.20





ICR LSGAN:

sigma: 0.03	FID 61.626, precision 0.282, recall 0.079
sigma: 0.05	FID 60.286, precision 0.243, recall 0.101
sigma 0.1	FID 54.920, precision 0.299, recall 0.088

Our experiments with ICR LSGAN suggest that increasing noise in the latent space samples led to improvements in generated images. This is likely because increased variance forces the network to explore more directions in the latent space, fostering diversity. However, we noted a side effect: some generated images exhibit noise, even at the smallest tested noise factor. Since we are using LSGAN instead of the DCGAN used by the ICR authors, our model may be more sensitive to the regularization terms. A possible future direction would be to experiment with different values for the λ coefficients in both the discriminator and generator to achieve better regularization balance.

Other Experiments

Gaussian Mixture

We also experimented with a Static Gaussian Mixture VanillaGAN (GM-GAN) and a static gaussian mixture WGAN (GM-WGAN). The core idea behind GM-GAN is to improve the diversity of generated samples by incorporating a Gaussian Mixture Model (GMM) in the latent space of the GAN. Rather than sampling from a single Gaussian distribution as in traditional GANs, GM-GAN samples latent vectors from a mixture of multiple Gaussian distributions.

Implementation details and results

In the first version of static GM-GAN we have focused on a mixture of distributions with different mean values and standard deviation equal to 1. We have selected components for each randomly, chosen its mean value from means arranged in a trigonometric circle of radius 2 in 2D dimension and extended them to latent dimension size by zeroing the rest of the indexes. We thought that capturing means from a circle could help generator samples to preserve the same structure each time they are sampled. This method introduced more structured and diverse latent space sampling compared to uniform sampling from a standard normal distribution.

One possible reason is that the fixed GMM structure may not adapt well to the evolving training dynamics in GANs. Since the generator and discriminator are constantly adjusting, a static prior may limit the generator's flexibility to fully leverage the multimodal structure, leading to suboptimal sample diversity/quality. To improve performance, we can consider experimenting with dynamic Gaussian Mixtures, where the GMM parameters are updated during training, allowing the latent space to evolve in response to the training dynamics. We chose the number of components to be 8, a latent dimension of 100.

Testing it locally, we got FID of 27.034, precision of 0.369, and recall of 0.1766.

In the second version, we sampled from multivariate normal distribution. We initialized mean values uniformly. Although the means sampled uniformly may not be as structured as means sampled from a trigonometric circle, it is a simple method that still introduces more diversity than just a single normal distribution sampling, potentially reducing mode collapse, which was our preoccupation.

In our tests, we increased # of components to 10, latent dimensions was 100, range for mean values was 0.1, standard deviation was 0.15

Testing it locally, we got FID of 84, precision of 0.1612 , and recall of 0.0647.

It would have been good to test it with the parameters similar to the first version, i.e. standard deviation equal to 1, means ranging from -2 to 2, and the same number of components, but this model was too long to train, and having such results with current parameters didn't make us more hopeful for the better ones. It turned out that having means arranged in a circle was more effective.

Conclusion

In conclusion, our project explored various approaches to enhance GANs by focusing on different aspects such as stability, diversity of generated samples, and visual quality. Though some of these models perform well, as shown by FID, precision, and recall, none of them really stood out from the others in terms of results. Furthermore, after combining some of these different methods together to continue to combat some of the shortcomings, we did not find any improvements that were both significant and replicable. The challenges of capturing the entire real distribution and completely eliminating mode collapse still remain even with our changes.

References

Arjovsky, M., Chintala, S., & Bottou, L. (2017, January 26). *Wasserstein GAN*. arXiv.org. https://arxiv.org/abs/1701.07875

Mao, X., Li, Q., Xie, H., Lau, R. Y. K., Wang, Z., & Smolley, S. P. (2016, November 13). Least squares generative adversarial networks. arXiv.org. https://arxiv.org/abs/1611.04076

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014, June 10). Generative adversarial networks. arXiv.org. https://arxiv.org/abs/1406.2661

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. (2017, March 31). *Improved training of Wasserstein GANs*. arXiv.org. https://arxiv.org/abs/1704.00028

Zhang, H., Zhang, Z., Odena, A., & Lee, H. (2019, October 26). Consistency regularization for generative adversarial networks. arXiv.org. https://arxiv.org/abs/1910.12027

Zhao, Z., Singh, S., Lee, H., Zhang, Z., Odena, A., & Zhang, H. (2020, February 11). Improved consistency regularization for GANs. arXiv.org. https://arxiv.org/abs/2002.04724

Nie, Jianghua, Xiao, Yongsheng, Huang, Lizhen, Lv, Feng, Time-Frequency Analysis and Target Recognition of HRRP Based on CN-LSGAN, STFT, and CNN, *Complexity*, 2021, 6664530, 10 pages, 2021. https://doi.org/10.1155/2021/6664530

Azadi, Samaneh, Olsson, Catherine, Darrell, Trevor, Goodfellow, Ian, Odena, Augustus, Discriminator Rejection Sampling, arXiv preprint arXiv:1810.06758, 2019, 15 pages. https://doi.org/10.48550/arXiv.1810.06758

Youngjung. (s. d.). improved-precision-and-recall-metric-pytorch/improved_precision_recall.py at master · youngjung/improved-precision-and-recall-metric-pytorch. GitHub. https://github.com/youngjung/improved-precision-and-recall-metric-pytorch/blob/master/improved_precision_recall.py