Data Science Lab

Assignment 2: GANs

Master IASD
Université PSL

ShenaniGANs

Evan AZOULAY, Alejandro JORBA, Aziz AGREBI

13/11/2024

Contents

1 Problem Statement

The goal of the project is to train a Generative Adversarial Network (GAN) to generate synthetic images of handwritten digits that closely resemble those in the MNIST dataset.

To do so, we explored the following approaches:

- Vanilla GANs
- WGANs
- WGANs with Gradient Penalty
- Latent Space Adaptation

2 GANs

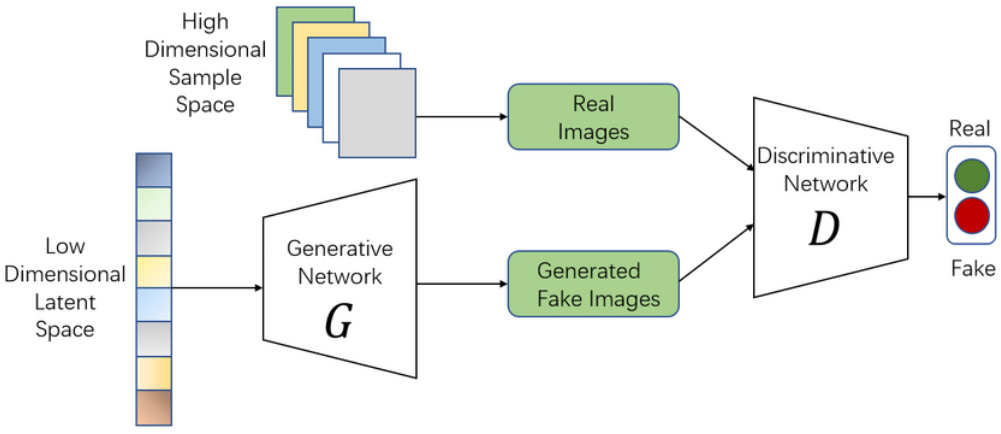
GANs follow an architecture comprised of two parts: a generator that takes a noise vector (sampled from a random distribution) and outputs a synthetic image, and a discriminator that receives either a real image (from the training data) or a generated image and outputs a probability indicating whether the image is real or fake. The training objective can be formalized through the following min-max optimization problem:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

where the generator seeks to minimize $\log(1 - D(G(\boldsymbol{z})))$, which encourages it to produce outputs that the discriminator classifies as real, and the discriminator attempts to maximize $\log D(\boldsymbol{x})$ for real images and $\log(1 - D(G(\boldsymbol{z})))$ for generated images, ensuring it correctly differentiates between real and synthetic data.

We used a Vanilla GAN as a baseline for our experiments. In this implementation, the Discriminator is a feedforward neural network consisting of four fully connected layers with Leaky ReLU activations, and a Sigmoid activation function in the output layer. The Generator is also a feedforward network, with an input noise vector of dimension 100, and the output is a flattened generated image of size 784 (28x28). The network uses Leaky ReLU activations between layers and a Tanh activation function at the output to ensure the generated images fall within the appropriate range. We trained both of them using the Adam optimizer for 100 epochs with learning rate 0.0002.

All of the approaches were based upon this architecture, and (where applicable) only the Discriminator was modified.



Architecture of a GAN, illustrating the Generator and Discriminator components.

3 Introduction to WGANs

Vanilla GANs face several issues that hinder their training stability and performance. These issues include:

- **Mode Collapse:** the generator produces limited variations, resulting in similar outputs.
- **Training Instability:** due to the use of Jensen-Shannon (JS) divergence, the model often struggles when the supports of the data and generated distributions are disjoint.
- **Vanishing Gradients:** occurs when the generator cannot improve due to small gradient updates.

To address these challenges, Wasserstein GAN (WGAN) was introduced. WGAN replaces the JS divergence with the Wasserstein distance, which provides smoother gradients and improves training stability.

4 Improvements Introduced by WGAN

4.1 Wasserstein Distance

WGAN replaces the traditional GAN's loss function, which uses JS divergence, with the Wasserstein distance. The Wasserstein distance provides a more informative gradient, especially when distributions are disjoint. The Wasserstein distance between two probability distributions P and Q is defined as:

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

where $\Pi(P,Q)$ is the set of all joint distributions with marginals P and Q. This distance measures the minimal "effort" required to transform one distribution into the other.

Using the Kantorovich-Rubinstein duality:

$$W(P_r,P_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim P_r}[f(x)] - \mathbb{E}_{x\sim P_\theta}[f(x)]$$

The loss function of the WGAN is given by:

$$\min_G \max_{\|D\|_L \leq 1} \mathbb{E}_{x\sim P_r}[D(x)] - \mathbb{E}_{x\sim P_G}[D(x)]$$

The discriminator must be 1-Lipschitz continuous.

4.2 Architecture Changes

In WGAN, the discriminator is replaced with a "critic" that outputs values in the range $(-\infty,\infty)$ instead of classifying into binary outputs. The WGAN critic does not use a sigmoid activation at the output. Additionally, weight clipping is applied to enforce the Lipschitz constraint, which stabilizes the gradients.

5 WGAN with Gradient Penalty (WGAN-GP)

5.1 Gradient Penalty

To ensure stability, WGANs require the critic to satisfy the Lipschitz continuity condition ($L \leq 1$). In the **original WGAN approach**, this continuity is enforced through **weight clipping**. However, it can limit the capacity of the critic, leading to poor convergence. WGAN-GP addresses this issue by introducing a gradient penalty that penalizes the norm of the critic's gradient to enforce the Lipschitz constraint without clipping. This penalty is added to the loss function to ensure the critic's gradients remain close to 1.

To calculate the gradient penalty, we first define an interpolated sample $\hat{x}$ as:

$$\hat{x} = \alpha \cdot x_{\text{real}} + (1-\alpha)\cdot x_{\text{fake}}$$

where α is sampled from a uniform distribution $U[0, 1]$.

The gradient penalty term is then expressed as:

$$\text{gradient\_penalty} = \lambda \cdot \mathbb{E}\left[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2\right]$$

Finally, the WGAN-GP loss function is given by:

$$L = \underbrace{\mathbb{E}_{\tilde{x}\sim P_G}[D(\tilde{x})] - \mathbb{E}_{x\sim P_r}[D(x)]}_{\text{Original WGAN Discriminator loss}} + \lambda \underbrace{\mathbb{E}_{\hat{x}\sim P_{\hat{x}}}\left[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2\right]}_{\text{Gradient penalty}}.$$

where λ is a hyperparameter that adjusts the strength of the gradient penalty.

5.2 Benefits of Gradient Penalty

Adding the gradient penalty offers several advantages:

- **Better Training Stability:** Gradient penalty avoids issues like vanishing or exploding gradients, providing a smoother learning signal.
- **Improved Sample Quality:** Generates higher-quality and more diverse samples.
- **Smoother Gradients:** Provides a stable learning signal for the generator, reducing mode collapse.

6 Algorithm

The WGAN-GP algorithm modifies the traditional GAN training procedure by enforcing the gradient penalty. Below is an outline of the WGAN-GP algorithm:

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.
while θ 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.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

WGAN

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.
while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m}\sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

WGAN-GP

7 Latent Space Adaptation

Beyond the adjustments made in WGAN and WGAN-GP, we investigated an alternative approach to enhance a Vanilla GAN's performance on the MNIST dataset. This method involves refining the latent space distribution to better suit each digit class, without altering the GAN training process itself.

Initially, the Vanilla GAN's latent space is sampled from a standard Gaussian distribution of dimension 100. To adapt this distribution for more accurate and diverse digit generation, we implemented the following steps:

7.1 Step 1: Training a Classifier

We first trained a Convolutional Neural Network (CNN) on the MNIST dataset to classify digits, achieving a 99% accuracy. This classifier serves as an oracle, providing a reliable "true" class for evaluating the generated images. By using this classifier, we establish a baseline for assessing GAN output quality by comparing generated images to real samples of their predicted class.

7.2 Step 2: Evaluating Generated Images

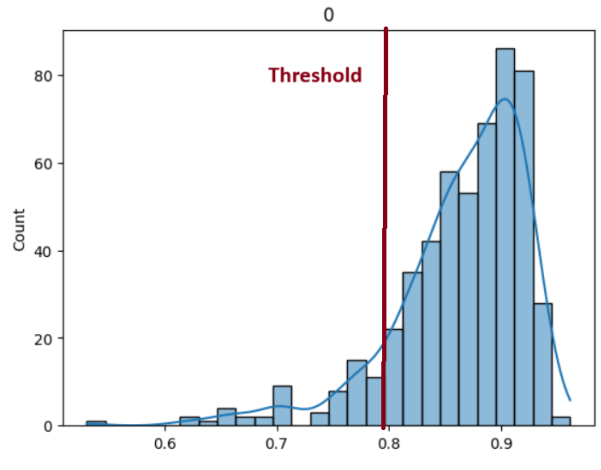
For each generated image $G(z)$, where z is a latent vector, we used the CNN to predict its class. We then assessed the similarity between $G(z)$ and the real MNIST images x_{real} from that predicted class using cosine similarity:

$$\text{cosine}(G(z), x\text{real}) = \max_{x \in x_{\text{real}}} \left(\frac{G(z) \cdot x}{\|G(z)\| \cdot \|x\|} \right),$$

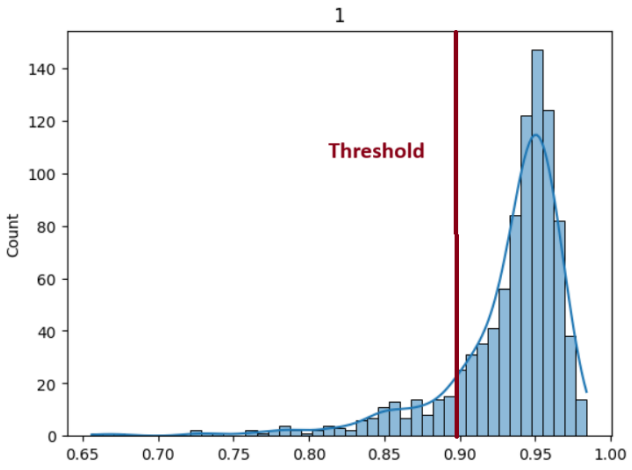
This similarity metric quantifies how close the generated image $G(z)$ is to real images from the same predicted class.

7.3 Step 3: Identifying High-Quality Samples

The cosine similarity scores were then used to categorize generated samples as either high- or low-quality. By examining the similarity score distribution for each class, we established class-specific thresholds, ensuring generated images meet quality standards while preserving diversity. This separation allows us to isolate high-quality samples for further latent space adaptation.



(a) Cosine similarity distribution for class 0.



(b) Cosine similarity distribution for class 1.

Threshold selection for high-quality samples based on cosine similarity for classes 0 and 1.

7.4 Step 5: Generating Optimized Latent Vectors

With these class-specific Gaussian parameters, we generated optimized latent vectors tailored for each class. For each digit class y, we sampled latent vectors z_y from $N(\mu_y, \text{diag}(\sigma_y^2))$ and used the GAN generator to produce high-quality images for that class:

$$z_y \sim N(\mu_y, \text{diag}(\sigma_y^2)),$$

where μ_y and σ_y are estimated from high-quality samples for each class. This adaptation enables the generation of class-specific high-quality images while preserving diversity across the generated samples. The class-specific thresholds used to select well-generated images (and their corresponding latent vectors) were carefully chosen: high enough to ensure quality, but low enough to maintain intra-class diversity.

Furthermore, this approach allows precise control over the number of samples generated for each class. By setting class-specific sample counts, we generated a balanced dataset of 10,000 images, with exactly 1,000 images for each digit class. This process mitigates the class imbalance issues seen with the original Vanilla GAN, where some classes (e.g., digit 2) were underrepresented, thus enhancing diversity and quality across all digit classes

Example of generated images

8 Results

Approach	FID	Precision	Recall
Vanilla GAN	27.96	0.54	0.21
WGAN-CP	31.99	0.52	0.21
Latent Space Adaptation	24.17	0.56	0.2

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, *Generative Adversarial Networks*, arXiv preprint arXiv:1406.2661, 2014.

[2] Alec Radford, Luke Metz, Soumith Chintala, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, arXiv preprint arXiv:1511.06434, 2015.

[3] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein GAN*, arXiv preprint arXiv:1701.07875, 2017.

[4] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, *Improved Training of Wasserstein GANs*, arXiv preprint arXiv:1704.00028, 2017.

[5] T. Issenhuth, U. Tanielian, D. Picard, and J. Mary, *Latent reweighting, an almost free improvement for GANs*, arXiv preprint arXiv:2110.09803, 2021.
`https://arxiv.org/abs/2110.09803`