Generative Adversarial Networks

Presented by **“GANerators”** team

Marwa Nair, Anna Krysta, Caio Rocha

Data Science Lab

06/11/2024

01.

PROBLEM STATEMENT

Problem Statement

Goal:

Train a GAN on the MNIST dataset, with a fixed generator architecture, and implement techniques to improve the data generation.

Problems with Vanilla GANs:

- Mode Collapse: The generator produces a limited variety of samples, ignoring parts of the data distribution.
- Vanishing gradients: The discriminator becomes too strong, resulting in poor generator updates.
- Overfitting: The discriminator becomes too focused on the generator's distribution, and fails to generalize.

02.

TECHNIQUES AND PRELIMINARY RESULTS

During training

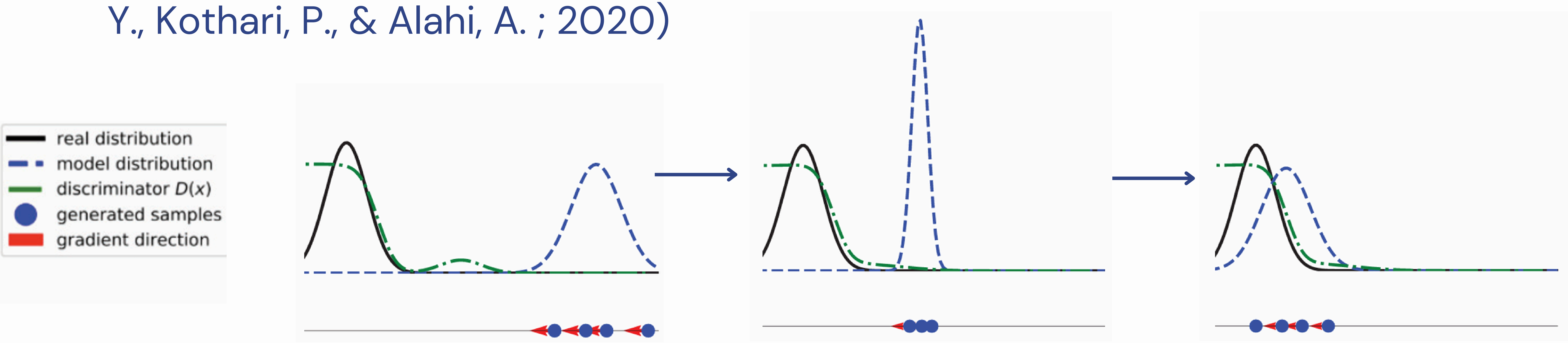
Differential Privacy GAN (Alex B., Gautam K. & Guojun Z.; 2023)

- We clip the Discriminator gradients and add Gaussian noise
- Generator is unchanged
- We perform more Discriminator steps per Generator step to preserve balance between the Generator and Discriminator

During training

Collaborative Sampling (CS)

- Given a fixed generator G and a fixed discriminator D.
- Refines the generated samples through gradient-based updates at the last layer of the generator.
- Shifts the generator distribution closer to the real data distribution. (Liu, Y., Kothari, P., & Alahi, A. ; 2020)



During training

Collaborative Sampling (CS)

1. Iteratively refines the generated samples

$$x_l^{k+1} = x_l^k - \lambda \nabla_l \mathcal{L}_G(x_l^k),$$

$$x^{k+1} = G_L \circ G_{L-1} \circ \ldots G_l(x_l^{k+1}),$$

2. Minimizes the loss of the generator

$$\mathcal{L}_G = -\mathbb{E}_{z \sim p_z}[\log D(G(z))]$$

3. Minimizes the loss of the disciminator

$$\mathcal{L}_D = -\mathbb{E}_{x \sim p_r}[\log D(x)] - \mathbb{E}_{x' \sim p_c}[1 - \log D(x')],$$

During training

Collaborative Sampling (CS)

1. Iteratively refines the generated samples

Output of layer l

$$x_l^{k+1} = x_l^k - \lambda \nabla_l \mathcal{L}_G(x_l^k),$$

Output of the last layer

$$x^{k+1} = G_L \circ G_{L-1} \circ \dots G_l(x_l^{k+1}),$$

2. Minimizes the loss of the generator

$$\mathcal{L}_G = -\mathbb{E}_{z \sim p_z}[\log D(G(z))]$$

3. Minimizes the loss of the disciminator

$$\mathcal{L}_D = -\mathbb{E}_{x \sim p_r}[\log D(x)] - \mathbb{E}_{x' \sim p_c}[1 - \log D(x')],$$

During training

Collaborative Sampling (CS)

1. Iteratively refines the generated samples

$$x_l^{k+1} = x_l^k - \lambda \nabla_l \mathcal{L}_G(x_l^k),$$

$$x^{k+1} = G_L \circ G_{L-1} \circ \ldots G_l(x_l^{k+1}),$$

2. Minimizes the loss of the generator

$$\mathcal{L}_G = -\mathbb{E}_{z\sim p_z}[\log D(G(z))]$$

Refined sample

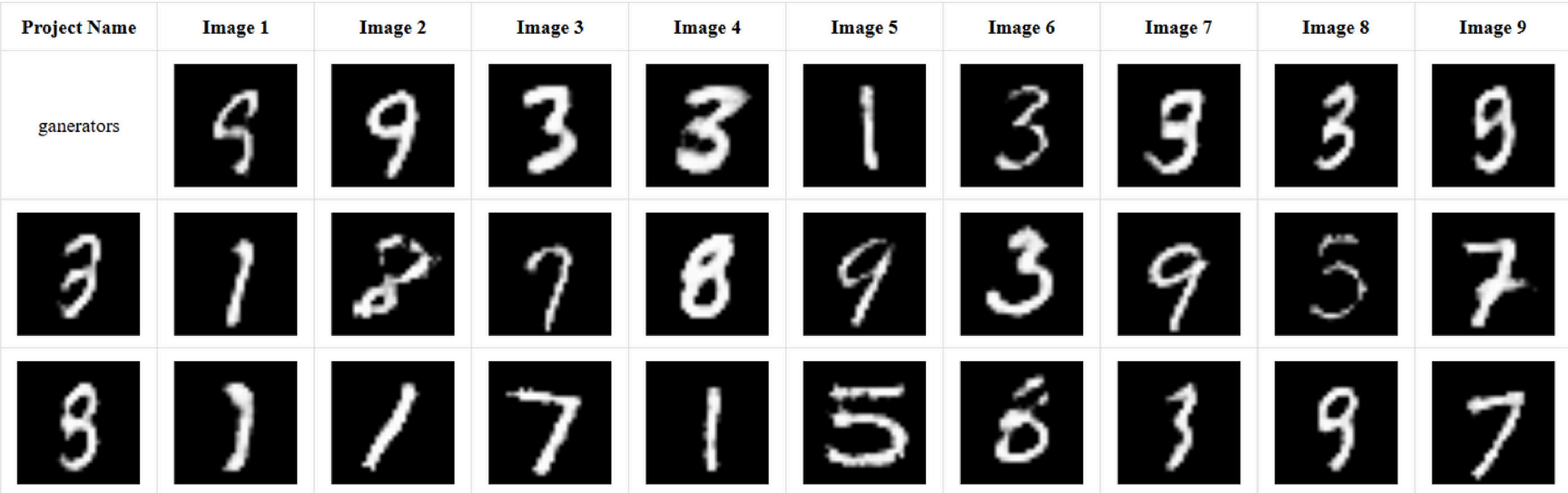
3. Minimizes the loss of the disciminator

$$\mathcal{L}_D = -\mathbb{E}_{x\sim p_r}[\log D(x)] - \mathbb{E}_{x'\sim p_c}[1 - \log D(x')],$$

Results

Collaborative Sampling (CS)

	FID	Precision	Recall
CS	26.82	0.53	0.23



During generation

Latent Space Interpolation (LSI)

Purpose Improve GAN output quality by sampling interpolated points in latent space, creating smoother, more realistic transitions.

Intuition Blending features in the latent space explores intermediate regions for greater image consistency. *(Radford et al., 2015 ; Karras et al., 2019)*

During generation

Latent Space Interpolation (LSI)

Pseudo-algorithm

```
for i in range(nb_images // interpolation_steps):
```

- Sample two random latent vectors z_1 and z_2
- Linearly interpolate between them at different intervals defined by **`interpolation_steps`:**

  $$z_{\text{interp}} = (1 - \alpha) \cdot z_1 + \alpha \cdot z_2 \quad \text{for} \quad \alpha \in \text{linspace}(0, 1, \text{interpolation\_steps})$$

- Feed each z_{interp} into the generator to generate new images.

During generation

Latent Space Interpolation (LSI)

Results

	FID	Precision	Recall
VanillaGAN + LSI	21.81	**0.57**	0.16
CS + LSI	**21.35**	**0.57**	**0.19**

03.

COMPARISON OF RESULTS

Comparative Analysis

	FID	Precision	Recall
VanillaGAN	26.84	0.53	**0.25**
CS	26.82	0.53	0.23
VanillaGAN + LSI	21.81	**0.57**	0.16
CS + LSI	**21.35**	**0.57**	0.19

THANK YOU FOR YOUR ATTENTION

Any questions?

Collaborative sampling algorithms

Algorithm 1 Collaborative Sampling

Input: a frozen generator G, a frozen discriminator D, the layer index for sample refinement l, the maximum number of steps K, the stopping criterion η
Output: a synthetic sample x
Randomly draw a latent code z
$x^0 \leftarrow \text{ProposeSample}(G, z)$
for $k = 0, 1, \ldots, K-1$ **do**
    if $D(x^k) < \eta$ **then**
        $g_l^k \leftarrow \text{GetGradient}(D, x_l^k)$,
        $x_l^{k+1} \leftarrow \text{UpdateActivation}(g_l^k, x_l^k)$, (Eq. 3)
        $x^{k+1} \leftarrow \text{UpdateSample}(G, x_l^{k+1})$, (Eq. 4)
    else
        break
    end if
end for

Collaborative sampling algorithms

Algorithm 2 Discriminator Shaping

1: **Input:** a frozen generator G, a pre-trained discriminator D, the batch size m
2: **Output:** a fine-tuned discriminator $\tilde{D}$
3: **for** number of D shaping iterations **do**
4: Draw m refined samples $\{x_c^{(1)}, \dots, x_c^{(m)}\}$ from the collaborative data distribution $p_c(x)$ according to Algorithm 1
5: Draw m real samples $\{x_r^{(1)}, \dots, x_r^{(m)}\}$ from the real data distribution $p_r(x)$
6: Shape the discriminator by minimizing the objective function Eq. 6
7: **end for**

Differential privacy GAN

Algorithm 1 TrainDPGAN($D; \phi_0, \theta_0, \texttt{OptD}, \texttt{OptG}, n_{\mathcal{D}}, T, B, C, \sigma, \delta$)

Input: Labelled dataset $D = \{(x_j, y_j)\}_{j=1}^{n}$. Discriminator $\mathcal{D}$ and generator $\mathcal{G}$ initializations ϕ_0 and θ_0. Optimizers $\texttt{OptD}$, $\texttt{OptG}$. Hyperparameters: $n_{\mathcal{D}}$ ($\mathcal{D}$ steps per $\mathcal{G}$ step), T (total number of $\mathcal{D}$ steps), B (expected batch size), C (clipping norm), and σ (noise level). Privacy parameter δ.

$q \leftarrow B/|D|$ and $t, k \leftarrow 0$ ▷ Calculate sampling rate q, initialize counters.

while $t < T$ **do** ▷ Update $\mathcal{D}$ with DPSGD.

  $S_t \sim \text{PoissonSample}(D, q)$ ▷ Sample a real batch S_t by including each $(x, y) \in D$ w.p. q.

  $\tilde{S}_t \sim \mathcal{G}(\cdot; \theta_k)^B$ ▷ Sample fake batch $\tilde{S}_t$.

  $g_{\phi_t} \leftarrow \sum_{(x,y)\in S_t} \text{clip}\left(\nabla_{\phi_t}(-\log(\mathcal{D}(x, y; \phi_t))); C\right)$
  $\quad + \sum_{(\tilde{x},\tilde{y})\in \tilde{S}_t} \text{clip}\left(\nabla_{\phi_t}(-\log(1 - \mathcal{D}(\tilde{x}, \tilde{y}; \phi_t))); C\right)$ ▷ Clip per-example gradients.

  $\widehat{g}_{\phi_t} \leftarrow \frac{1}{2B}(g_{\phi_t} + z_t)$, where $z_t \sim \mathcal{N}(0, C^2\sigma^2 I)$ ▷ Add Gaussian noise.

  $\phi_{t+1} \leftarrow \texttt{OptD}(\phi_t, \widehat{g}_{\theta_t})$

  $t \leftarrow t + 1$

  if $n_{\mathcal{D}}$ divides t **then** ▷ Perform $\mathcal{G}$ update every $n_{\mathcal{D}}$ steps.

    $\tilde{S}'_t \sim \mathcal{G}(\cdot; \theta_k)^B$

    $g_{\theta_k} \leftarrow \frac{1}{B} \sum_{(\tilde{x},\tilde{y})\in \tilde{S}'_t} \nabla_{\theta_k}(-\log(\mathcal{D}(\tilde{x}, \tilde{y}; \phi_t)))$

    $\theta_{k+1} \leftarrow \texttt{OptG}(\theta_k, g_{\theta_k})$

    $k \leftarrow k + 1$

  end if

end while

$\varepsilon \leftarrow \text{PrivacyAccountant}(T, \sigma, q, \delta)$ ▷ Compute privacy budget spent.

Output: Final $\mathcal{G}$ parameters θ_k and (ε, δ)-DP guarantee.