RDFIA - Generative Adversarial Networks (2-de)

Bourzag Mohamed Chakib, Missoum Youcef

December 2025

1 Introduction

This document contains the exercises and written solutions for the Generative Adversarial Networks practical work (RDFIA TP2-de). We'll start with a brief asummary of the subject before moving to the exercises:

2 Summary

2.1 Context and Motivation

Data generation is a pivotal concept in modern computer vision, addressing critical challenges such as data scarcity and domain shifts. As explored in previous labs (e.g., Domain Adaptation), generating synthetic data can serve either as a powerful form of data augmentation or as a bridge between different domains.

While Variational Autoencoders (VAEs) provide a generative framework by encoding images into a probabilistic latent space, their training objective fundamentally relies on minimizing a reconstruction error, typically the Mean Squared Error (MSE). This constraint often leads to blurry predictions, as the model averages multiple plausible outputs to minimize risk. To achieve sharper and more photorealistic results, we turn to **Generative Adversarial Networks (GANs)**, which replace fixed reconstruction losses with a learnable, adversarial objective.

2.2 Theoretical Framework

GANs are grounded in game theory and are formulated as a **minimax game** between two competing neural networks:

- **The Generator (G):** Its objective is to model the data distribution p_{data} by mapping a latent noise vector z, sampled from a simple prior distribution (e.g., a Gaussian), to a synthetic image x_{fake}.
- **The Discriminator (D):** Acts as a binary classifier tasked with distinguishing real samples drawn from the dataset from fake samples generated by G.

This adversarial process implicitly drives the generator to minimize the Jensen–Shannon divergence (closely related to the Kullback–Leibler divergence) between the generated distribution and the true data distribution. Ideally, training converges to a **Nash equilibrium** where the generator produces perfectly realistic samples and the discriminator can no longer distinguish between real and fake data, yielding a prediction probability of $P = 0.5$.

2.3 Architectural Evolution

In practice, GAN training is notoriously unstable and prone to issues such as mode collapse. To address these challenges, several architectural advancements have been proposed:

- **DCGAN (Deep Convolutional GAN):** Established a standard for stable training by replacing pooling operations with strided convolutions and incorporating Batch Normalization.
- **ProGAN:** Addressed high-resolution image generation by progressively growing the network, starting from 4×4 images and incrementally adding layers during training.
- **StyleGAN:** Introduced a mapping network and Adaptive Instance Normalization (AdaIN), enabling explicit control over visual attributes at different scales, from coarse structures to fine details.

2.4 Practical Part I: Unconditional Generation (DCGAN)

The first part of this practical focuses on reproducing the DCGAN architecture using the MNIST dataset. The main objective is to understand the delicate balance required for successful adversarial training. Specific architectural choices are enforced, such as using LeakyReLU activations in the discriminator to avoid vanishing gradients and ReLU activations in the generator, enabling the transformation of random noise into coherent handwritten digits.

2.5 Practical Part II: Conditional Generation (cGAN)

Standard GANs offer limited control over the generation process, as they sample randomly from the learned data manifold. The second part of the practical addresses this limitation by implementing **Conditional GANs (cGANs)**. By conditioning both the generator and discriminator on auxiliary information c—in this case, the class label—the task is transformed from generating an arbitrary digit to generating a specific one (e.g., the digit 7).

This conditioning paradigm is fundamental to more advanced applications, where conditions may include text embeddings (text-to-image generation) or even input images themselves (image-to-image translation, such as Pix2Pix).

3 Exercices

Exercise 1. ***Generative Adversarial Networks***

1. *Interpret the equations (6) and (7). What would happen if we only used one of the two ?*

   Solution.

   - *Equation (6) showcases that the goal of the generator is to generate fake images as realistic as possible to fool the discriminator to classify them an real.*
   - *Equation (7) describes that the goal of the discriminator is to maximize real images' and fake generated images by the generator's recognition and classify them as real and fake respectively.*
   - *If we use only one of the two equations:*
     - ***Only train D**: The discriminator becomes perfect, the generator never improves and no generation capability can be achieved.*
     - ***Only train G**: There woulb be no learning signal, the generator would not have a notion of "realism" as the discriminator would not be able to provide it, and training would collapse immediately as a result.*

   □

2. *Ideally, what should the generator G transform the distribution $P(z)$ to ?*

   Solution. *$P(z)$ would ideally converge to the real data distribution $P_{data}(x)$. The discriminator as a result would output a uniform distribution with 0.5 for all values, meaning that it can't distinguish between real and fake images.* □

3. *Remark that the equation (6) is not directly derived from the equation 5. This is justified by the authors to obtain more stable training and avoid the saturation of gradients. What should the "true" equation be here ?*

   Solution.

   - *The "true" equation is present in equation (5) and is in fact:*

   $$\min_G \; E_{z\sim p(z)} \left[\log\left(1 - D(G(z))\right)\right]$$

   - *This is not used in practise because of the vanishing gradient problem that occurs when $D(G(z))$ has values near to 0 at early training. That's why they moved to equation (6) that gives the same fixed point with more stability and stronger gradients.*

   □

4. *Comment on the training of of the GAN with the default settings (progress of the generations, the loss, stability, image diversity, etc.)*

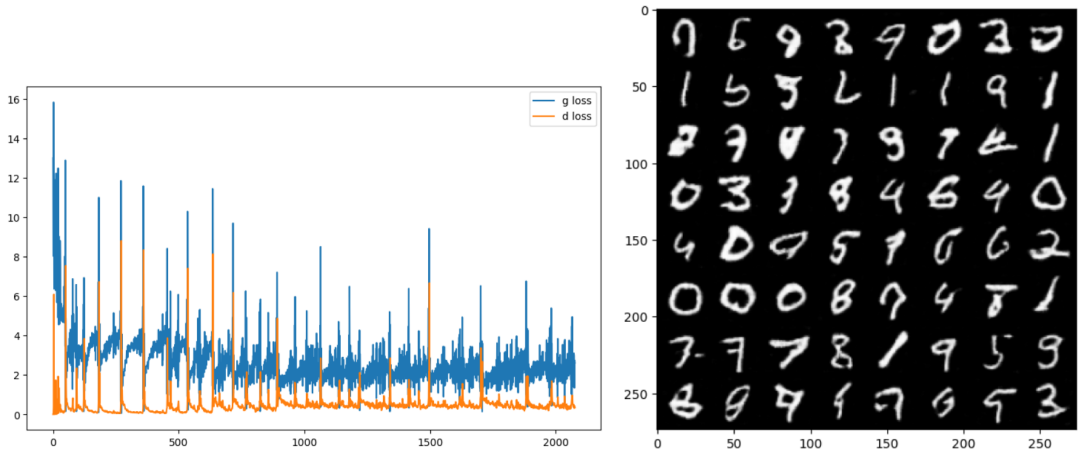
   Solution.
   The results are showcased in figure 1:

   - *We can notice the progress of the generated images from a basic noise to clearer digit images.*
   - *We can notice some ups and downs in both generator and discriminator loss curves, showcasing the competitive training aspect between the two, as one tries to beat the other and the other comes back.*
   - *Those ups and downs introduce some unstability to the training process.*

   □

5. *Comment on the diverse experiences that you have performed with the suggestions above. In particular, comment on the stability on training, the losses, the diversity of generated images, etc.*

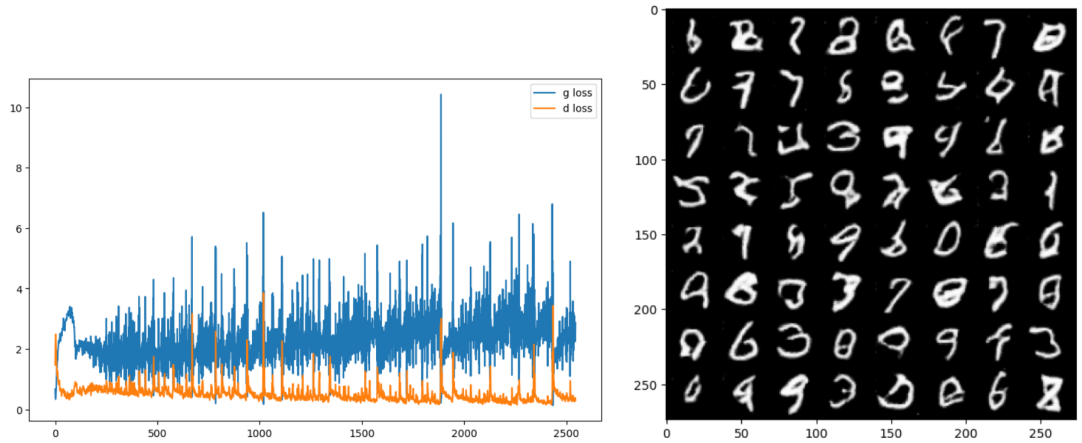
   Solution.
   We tried the following experiments (with the remaining hypeparameters as the default ones):



(a) Training losses. (b) Generated images.

Figure 1: Default configuration results.

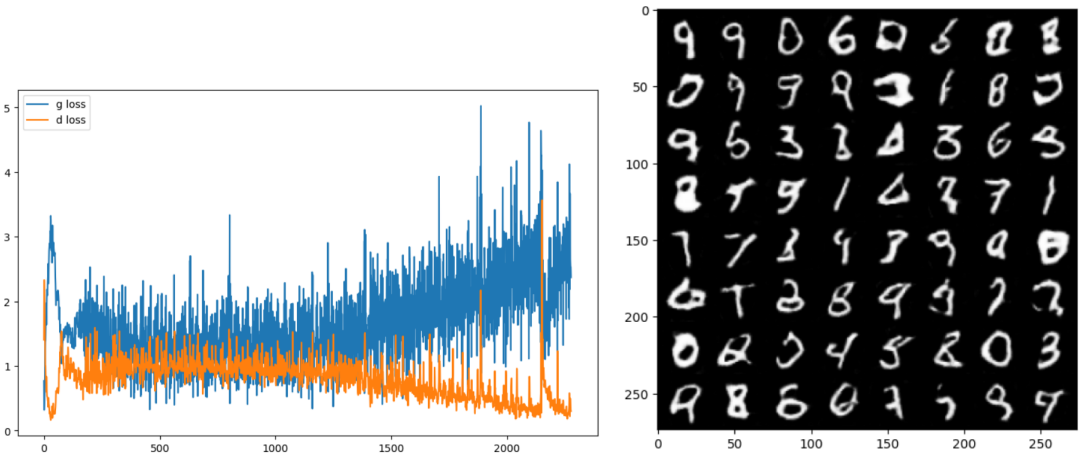
- $nz = 10000$*:*
  - *We notices a much slower training due to the high dimensionnality of the vector.*
  - *In addition, the results, for the same amount of epochs was much worse as it can be seen in the figure 2. This tells that we either have to train it for much more epochs, or simply having a much complex input noise vector can give really bad results depending on the complexity of the generated images. This size should be chosen carefully.*
  - *Looking at the loss curves, we can see that the training was also more unstable that the default configuration.*



(a) Training losses. (b) Generated images.

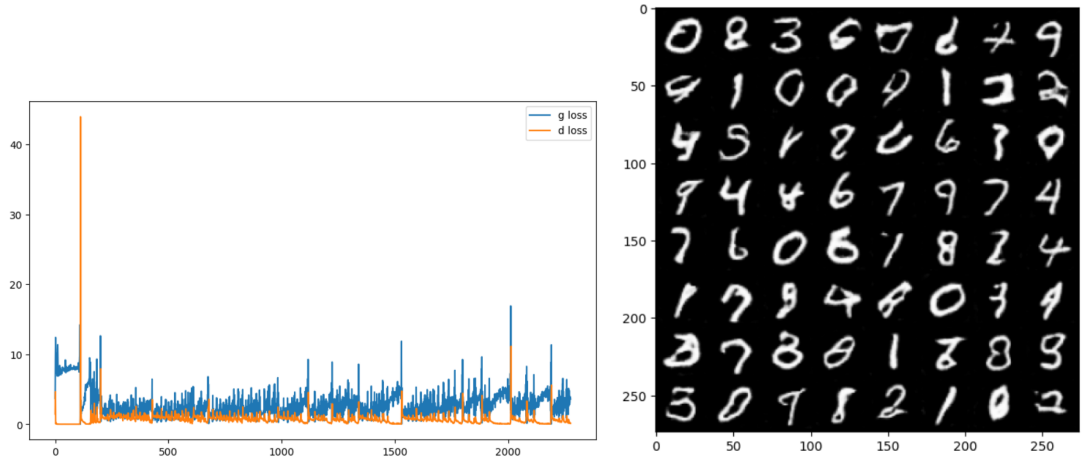
Figure 2: nz=10000 configuration results.

- ***Replacing the custom weight initialization with pytorch's default initialization****: The results are presented in figure 3:*
  - *The generated images seem pretty much the same as the default configuration at first sight.*
  - *However, looking at the loss curves, we can see at the end of the training that the generator loss increases while the discriminator's decreases, meaning that the discriminatore is outperforming the generator. This might stop the results from improving as the generator is not able to fool the discriminator. This phenomenon was not seen in with the custom weights (figure 1a), meaning that they serve stabilizing more the training.*
- ***Incresing*** *ndf* ***to 256 with*** $ngf = 128$***, making it bigger this time****: The results are presented in figure 4:*
  - *We also noticed here a much slower training.*
  - *The training was quite unstable at the beginning but it became more stable as the training progressed.*
  - *We noticed a bit a similar phenomenon to the previous experiment.*
- ***Test on CIFAR-10 dataset****: We need to adjust the input channels from 1 to 3 in the code before executing. The results are presented in figure 5:*



(a) Training losses. (b) Generated images.

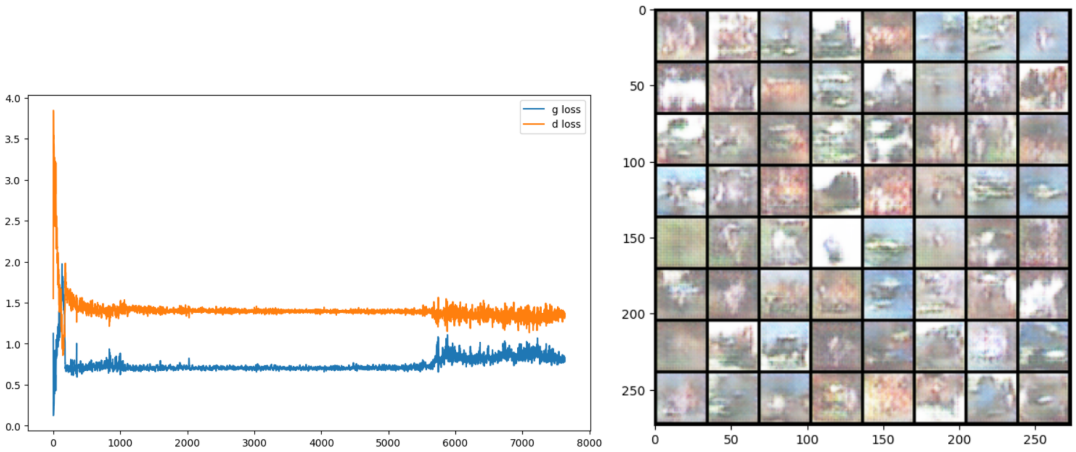
Figure 3: normal weights configuration results.



(a) Training losses. (b) Generated images.

Figure 4: ndf=256 configuration results.

– *We tried running it for 25 epochs.*
– *As we can see, the losses tend to converge but at a certain they start tangling again. They also present really low values.*
– *The images are converging to the actual ones; it may need more training just to reach a suitable result.*



(a) Training losses. (b) Generated images.

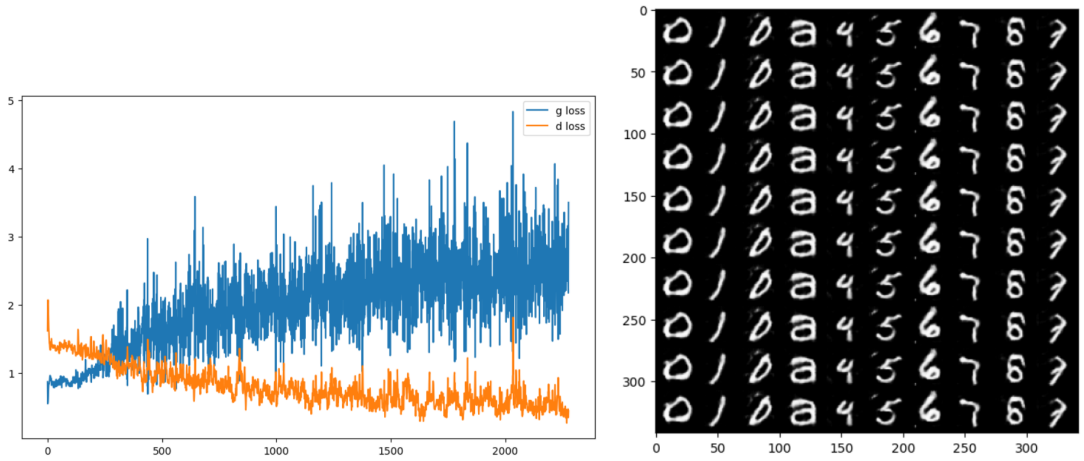
Figure 5: CIFAR results.

□

Exercise 2. ***Conditional Generative Adversarial Networks***

- *6. Comment on your experiences with the conditional DCGAN.*

  Solution.
  The results are presented in figure 6, the training was done using the default configuration but even having $nz = 1000$ *gave similar results:*



(a) Training losses. (b) Generated images.

Figure 6: cGAN default configuration results.

– *As we can see, there is an alignement of the digits since the output label is given an an input.*
– *We can notice an outperforming discriminator in terms of loss as it was noticed in previous experiences.*
– *Some digits are well-distinguishible while others are not (like 2 and 3) compared to the original GAN.*

□

- *7. Could we remove the vector y from the input of the discriminator (so having* $cD(x)$ *instead of* $cD(x; y)$*) ?*

  Solution.
  No, we cannot. *In a Conditional GAN, the discriminator has two specific tasks:*

  1. *Judging if the image is realistic (Real vs. Fake).*

2. *Judging if the image* ***matches*** *the condition y (Class consistency).*

If we remove y from the discriminator's input, it effectively becomes an unconditional discriminator $D(x)$ that only checks for realism. The generator G would then learn to fool D by generating any realistic digit (e.g., always generating a perfect "1") regardless of the input label y. It would ignore the condition entirely, failing the objective of conditional generation. □

- *8. Was your training more or less successful than the unconditional case ? Why ?*

  Solution.
  The training appeared ***less successful*** *(or generated lower quality images) compared to the unconditional case. While Conditional GANs theoretically reduce the search space, they introduce new challenges in practice:*

  - ***Increased Complexity:*** *The discriminator has a harder task: it must verify not only if the image looks real, but also if it matches the specific class label y. If the network capacity is limited, this dual objective can make optimization more difficult.*
  - ***Conditioning Mechanism:*** *In simple implementations (like concatenating labels to the input), the conditioning might not be efficiently integrated, causing the generator to struggle to balance image quality with class constraints.*
  - ***Hyperparameters:*** *The hyperparameters (learning rate, batch size) that worked for the unconditional model might not be optimal for the conditional architecture, leading to instability or mode collapse.*

  □

- *9. Test the code at the end. Each column corresponds to a unique noise vector z. What could z be interpreted as here ?*

  Solution.
  In this visualization, z can be interpreted as the ***style*** *(or handwriting characteristics) of the digit, while y represents the* ***content****.*
  Since the code `z = sample_z(n_ex, nz).repeat(n_classes, 1)` *fixes the latent vector z for an entire column while varying the label y across rows, we observe that digits in the same column share similar visual attributes (e.g., thickness, slant, size). So, we can say that the noise z controls the variation within the class (style), and the condition y controls the class identity.* □

4 Conclusion

To conclude, this practical provided a comprehensive exploration of Generative Adversarial Networks (GANs), highlighting both their generative power and their notorious training instability.

In the first part, we successfully reproduced the **DCGAN architecture** on MNIST. Our experiments demonstrated that while GANs can generate sharper, more realistic images than VAEs, they are highly sensitive to hyperparameters. We observed that:

- **Latent Space Dimensionality:** Increasing the noise vector size ($nz = 10000$) proved detrimental, likely due to the curse of dimensionality making the mapping learning significantly harder.
- **Initialization and Capacity:** Custom weight initialization was crucial for stability. Standard initialization led to the discriminator overpowering the generator, preventing effective learning (vanishing gradients).
- **Dataset Complexity:** Applying the model to CIFAR-10 showed that generating RGB natural images requires significantly more training time and network capacity than simple handwritten digits.

In the second part, we extended the architecture to **Conditional GANs (cGANs)**. This allowed us to control the output class, effectively transforming the noise vector z into a "style" encoding (controlling handwriting thickness/slant) while the label y controlled the "content." However, we noted that cGANs can be practically more difficult to train than unconditional ones. The discriminator's dual task of verifying both realism and class consistency can lead to optimization challenges, resulting in slightly lower visual quality in our specific experiments.

Ultimately, this work confirms that while the adversarial minimax game is a powerful framework for data generation but requires careful architectural design and rigorous hyperparameter tuning.