RDFIA - M2 IMA

Practical work report 2-de
Generative Adversarial Networks

Done by :

Carlos GRUSS & Oussama RCHAKI

25/09/24

SORBONNE UNIVERSITÉ

Contents

Objective

1 – Generative Adversarial Networks

Contrary to what are known as "discriminative" approaches, which aim to learn the distribution $P(Y|X)$, for example, in classification tasks, **generative** models aim to learn the joint distribution $P(X, Y)$. Note that this implies learning the likelihood of X.

In particular, this report will examine Generative Adversarial Networks (GAN) - although they don't correspond to the traditional definition of generative models, they are very close. Since their debut in 2014, GANs have had a massive impact on unsupervised learning, requiring little or no labels. Their capacity to model the underlying structures of data have placed them in the heart of state-of-the-art methods in generation, completion, and edition of images.

The goal of a "classic" GAN is to be capable of generating an image $\tilde{X}$ from a noise vector z sampled from a predefined distribution.

$$\tilde{X} = G(z), \quad z \sim P(z) \tag{1}$$

This framework can be extended to conditional GANs (cGANs), in which the goal is to produce an image $\tilde{X}$ from a noise vector z as well as the input y, which guides the generation process. For example, y can be a class label, an image, a sentence, etc.

$$\tilde{X} = G(z, y), \quad z \sim P(z) \tag{2}$$

1.1 – General Principle

As the legend goes, during a lunch break in a restaurant, Ian Goodfellow had a revolutionary idea: the creation of a model capable of generating data indistinguishable from real data. This marked the birth of the **Generative Adversarial Network (GAN)**, as introduced in his seminal work in 2014.

GANs are a class of deep learning models that consist of two neural networks—the **generator** and the **discriminator**—which are trained together in a competitive manner. The generator learns to produce realistic data (e.g., images), while the discriminator aims to distinguish real data from fake data generated by the generator. This interplay forms an adversarial "game," as described in game theory, where both networks improve iteratively.

In this part of the practical work, we focus on the **Deep Convolutional GAN (DCGAN)** architecture introduced by Radford et al. (2016). DCGAN forms the foundation of many state-of-the-art GAN models today. The main objective is to generate realistic data that aligns with the unknown probability distribution $P(X)$ of the real dataset. Since the exact distribution is unknown, we only have access to samples from it, denoted as $\{x_i \sim P(X)\}_{i=1}^{N}$.

The generator G transforms a random input z, sampled from a fixed distribution (commonly $U[-1, 1]$ or $\mathcal{N}(0, I)$), into a realistic image $\tilde{x} = G(z)$. Meanwhile, the discriminator D evaluates whether an image is real (from the dataset) or fake (produced by the generator). Ideally, $D(x^*) = 1$ for real images and $D(\tilde{x}) = 0$ for generated images.

GAN training alternates between optimizing the generator and the discriminator using the following objective functions:

$$\min_G \max_D \Big(\mathbb{E}_{x^* \sim P(X)}[\log D(x^*)] + \mathbb{E}_{z \sim P(z)}[\log(1 - D(G(z)))] \Big). \tag{3}$$

In this section, we will delve into the DCGAN architecture and explore its adversarial training process. We will analyze how the generator learns to fool the discriminator and how the discriminator adapts to distinguish real and fake images. Additionally, we will discuss the binary cross-entropy loss used to guide

both networks during training and examine the practical implementation of GANs through alternating updates of the two networks.

Question 1

$$\max_G \mathbb{E}_{z\sim P(z)}[\log D(G(z))] \tag{4}$$

This equation represents the objective function of the generator. The generator G aims to produce samples $\tilde{x} = G(z)$, starting from a random input z drawn from a fixed distribution $P(z)$, such that the discriminator D assigns a high probability (close to 1) to these generated samples. Intuitively, the generator tries to "fool" the discriminator by making its outputs $\tilde{x}$ as realistic as possible, so the discriminator thinks they are real data.

$$\max_D \Big(\mathbb{E}_{x^*\in \text{Data}}[\log D(x^*)] + \mathbb{E}_{z\sim P(z)}[\log(1 - D(G(z)))]\Big) \tag{5}$$

This equation represents the objective function of the discriminator. The discriminator D has two goals:

1. Assign a high probability (close to 1) to real data samples x^* from the dataset.
2. Assign a low probability (close to 0) to the fake samples $\tilde{x} = G(z)$ generated by the generator.

The discriminator is trained to distinguish real data from fake data as accurately as possible.

Now if we only used the equation 4, the generator would generate data $\tilde{x} = G(z)$ without any feedback from a discriminator. Without the discriminator to provide signals about whether the generated samples are realistic or not, the generator would have no way of improving its outputs. As a result, the generator would produce meaningless or random outputs that do not resemble the real data distribution.

In the other hand, if we only used the equation 5, the discriminator would learn to classify real samples x^* as real and randomly generated samples $\tilde{x}$ as fake. However, without the generator updating its outputs, the discriminator would only train on fixed random samples $G(z)$ that do not improve over time. This would render the training ineffective because the discriminator would not face any challenging or evolving fake samples, defeating the purpose of adversarial training.

Question 2

Ideally, the generator G should transform the distribution $P(z)$ (the latent noise distribution) into the distribution of real data $P(X)$. This means that for any $z \sim P(z)$, the output of the generator $G(z)$ should follow the same distribution as the real data X. Mathematically, this can be expressed as:

$$G(P(z)) \approx P(X) \tag{6}$$

Question 3

The true equation should be:

$$\min_G \mathbb{E}_{z\sim P(z)}\Big[\log\big(1 - D(G(z))\big)\Big]. \tag{7}$$

This equation minimizes the likelihood that the discriminator correctly identifies the generated samples $G(z)$ as fake. However, directly optimizing this objective can lead to very small gradients when $D(G(z))$ approaches 0. To overcome this, the authors instead maximize the term:

$$\max_G \mathbb{E}_{z\sim P(z)}\Big[\log D(G(z))\Big]$$

This reformulation ensures that the generator focuses on producing samples that the discriminator classifies with high confidence as real, leading to more stable and efficient training.

1.2 – Architecture of the networks

Question 4

In the observed GAN experiments, as training progresses, the generated images gradually begin to resemble the target dataset, such as digits. During the early iterations, the discriminator loss is initially larger than the generator loss. However, this situation quickly reverses, with the generator loss becoming smaller than the discriminator loss after a few iterations. For the remainder of the training, the discriminator loss, in general, remains smaller than the generator loss.

The observed reversal of losses can be explained by the adversarial dynamics of GAN training. In the early iterations, the generator produces random outputs that are far from resembling real samples. The discriminator, being in its initial state, has not yet learned to effectively classify these outputs, which can lead to higher initial discriminator loss values. As training progresses, the discriminator quickly adapts, learning to classify real and fake samples more effectively, which reduces its loss. Simultaneously, the generator begins to improve, producing outputs that challenge the discriminator, leading to a slight increase in generator loss as it struggles to deceive the increasingly effective discriminator. This dynamic reflects the delicate balance between the two networks during GAN training, where improvements in one network directly impact the performance of the other.

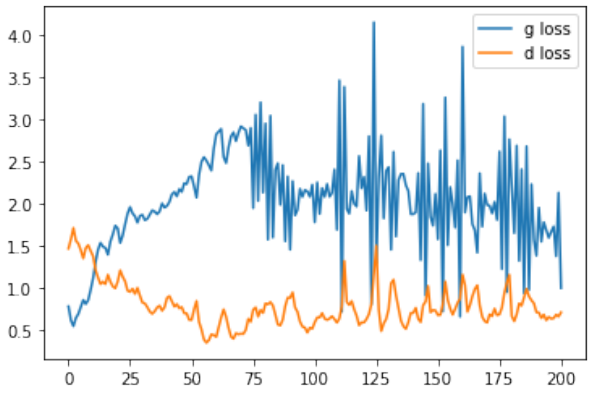
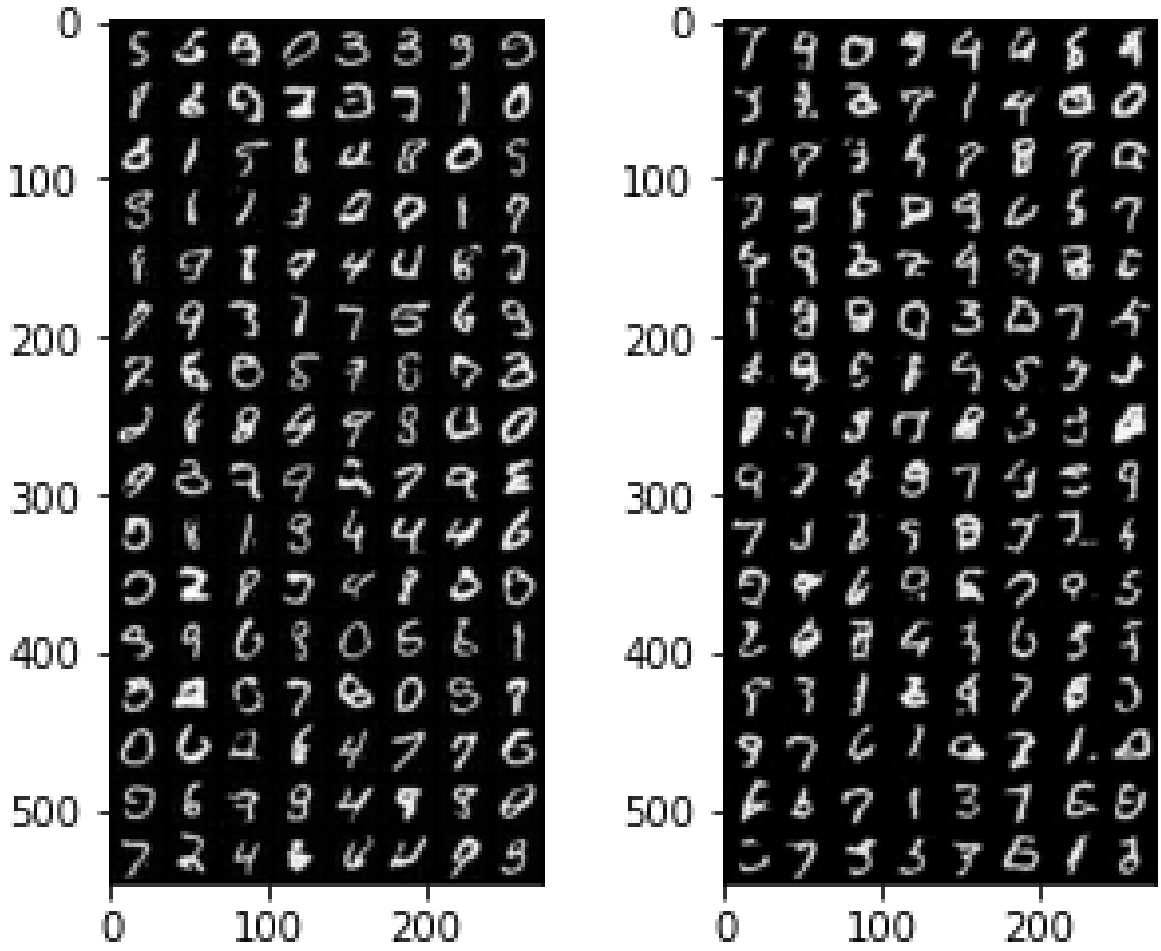


Figure 1 : The discriminator and generator losses at early iterations.

Question 5

Increasing Epochs

The very first experiment we can conduct involves increasing the number of epochs from 5 to 30. This is generally an effective approach to ensure better stability and convergence for the model, as it allows the generator to learn more comprehensively. The figure below illustrates that after 30 epochs, the generated images are significantly clearer and more accurate.



(A) Generated digits after 30 epochs. (B) Generated digits after 5 epochs.

FIGURE 2 : The generated digits.

INCREASING NGF AND NDF

Upon modifying the architecture with a significantly larger `ngf = 1000` for the generator and `ndf = 32` for the discriminator, we observed that the discriminator's loss rapidly decreases to zero while the generator's loss increases. This behavior indicates that the discriminator becomes overly confident in distinguishing real images from fake ones, thereby dominating the training process. This occurs because the generator's increased complexity leads to more pronounced discrepancies between the generated and real data **early** in training. These discrepancies allow the discriminator to **quickly** learn clear decision boundaries, improving its ability to distinguish real from fake images. Consequently, the discriminator dominates the training process from the very first iterations and continues to outperform the generator throughout the training epochs. As the generator and discriminator are in competition, this imbalance allows the discriminator to "win" the adversarial game in this setup.

We observed a similar behavior when the roles were reversed, i.e., with `ngf = 32` for the generator and `ndf = 1000` for the discriminator. In this case, the discriminator's larger capacity enables it to quickly learn and dominate the generator, as the smaller generator struggles to produce outputs that are challenging enough for the discriminator. This further highlights the importance of balancing the capacities of the generator and discriminator to ensure stable and meaningful training.

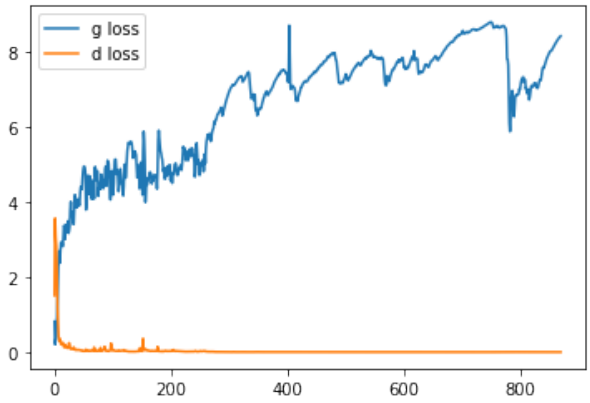
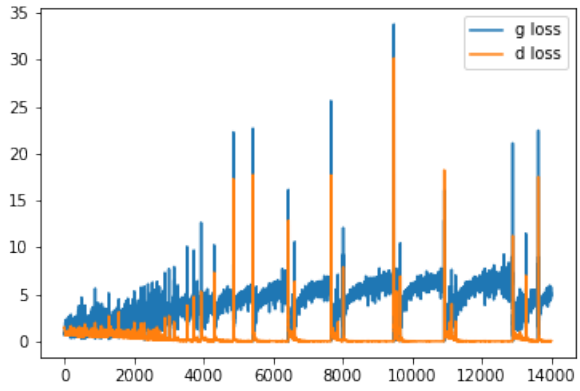


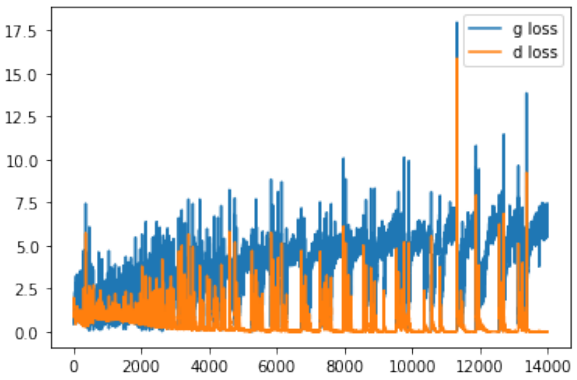
FIGURE 3 : The generator and discriminator losses for `ngf = 1000` and `ndf = 32`.

WEIGHTS INITIALIZATION

When using PyTorch's default initialization, the training process becomes less stable. The generator loss increases over time, reaching values as high as 35, suggesting that the generator struggles to improve. The discriminator loss exhibits sharp spikes and significant oscillations, reflecting an unstable learning process. This instability can be attributed to the default initialization not prioritizing the delicate balance required for adversarial training. In contrast, the DCGAN initialization provides a more effective and stable framework for GAN training. This stability is achieved due to the careful initialization of small, random weights, which prevents vanishing or exploding gradients and ensures **meaningful competition** between the generator and discriminator. Furthermore, it is evident that with the DCGAN initialization, the generator and discriminator engage in a more balanced and dynamic competition compared to PyTorch's default initialization. This increased competition is a positive indicator for adversarial training, as it allows both networks to improve iteratively and achieve better overall performance. Ultimately, this demonstrates that proper weight initialization plays a crucial role in stabilizing GAN training and enabling the networks to reach convergence effectively.



(A) G and D losses with Pytorch initialization.



(B) G and D losses with the gaussian initialization.

FIGURE 4 : Influence of changing the weights initialization.

INCREASING AND DECREASING n_z

The parameter n_z represents the dimensionality of the latent space, which determines the size of the random noise vector z fed into the generator. Reducing n_z (e.g., $n_z = 10$) results in a lower-dimensional latent space, which restricts the generator's capacity to learn complex features and limits the diversity of generated samples. This often leads to less realistic outputs, as the generator cannot encode sufficient variation to produce detailed and distinct images. On the other hand, increasing nz significantly (e.g.,

$n_z = 1000$) provides a higher-dimensional latent space, enabling the generator to learn more intricate features and produce a wider variety of samples. However, an excessively large n_z can increase the computational complexity and potentially lead to overfitting, where the generator memorizes specific patterns rather than generalizing well. Therefore, choosing an appropriate nz is crucial for achieving a balance between diversity, realism, and training efficiency.

Changing the dataset

In this section, we aim to generate images from different datasets: CIFAR-10 and the hand-drawn sheep dataset.

We first attempted to generate CIFAR-10 images without modifying the generator and discriminator originally designed for the MNIST dataset. The results were very poor, with a significantly high loss for the generator compared to the discriminator. This is because the GAN architecture was too simple to generate CIFAR-10 images, which contain more complex patterns and features. Consequently, the discriminator easily outperformed the generator, making it nearly impossible for the generator to learn meaningful features.

To address this, we modified the DCGAN architecture to make it more complex by adding additional convolutional and transposed convolutional layers. We also increased the number of epochs to 100, set the latent vector size (nz) to 10,000, and decreased lr_d to slow the discriminator. With these modifications, the generated images showed noticeable improvement compared to the previous attempt. However, the results were still far from satisfactory. While we could vaguely distinguish some objects, such as horses and cars, the overall image quality remained poor.

This highlights the limitations of the current DCGAN architecture and suggests that more advanced techniques, such as Progressive GANs, StyleGANs, or the use of attention mechanisms, might be necessary to achieve better results. Additionally, optimizing hyperparameters like learning rates, batch size, and experimenting with different loss functions could further enhance the quality of the generated images.

Figure 5 : Generated images of our generator trained on Cifar-10.

We now try with simpler data, such as the hand-drawn sheep dataset. The results shown in figure 6 were clear and highly satisfying.

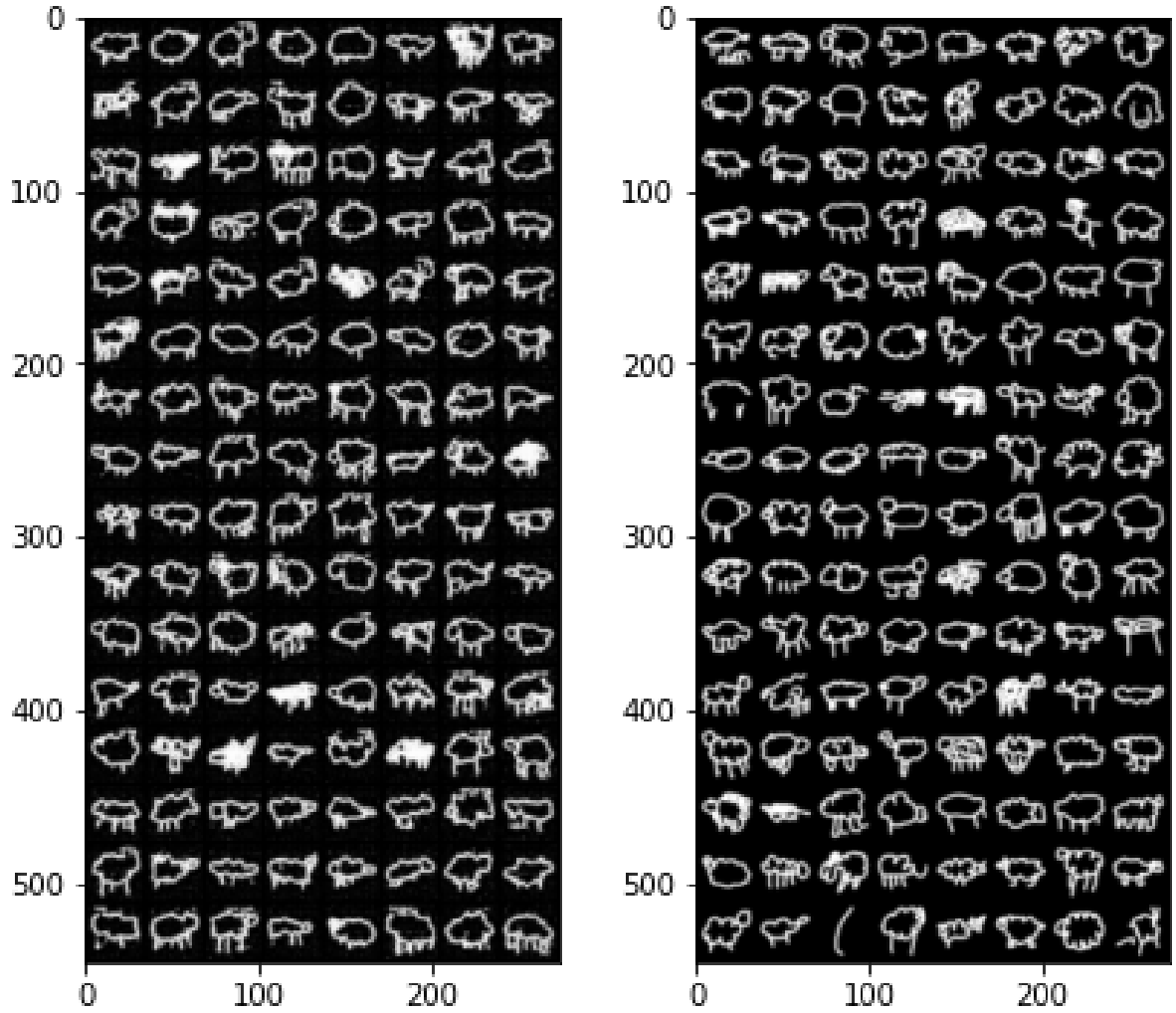


FIGURE 6 : Can you guess which one is the real hand-drawn sheep?.

2 – CONDITIONAL GENERATIVE ADVERSARIAL NETWORKS

2.1 – GENERAL PRINCIPLE

Generative Adversarial Networks (GANs) can be extended to *conditional models* when both the generator and discriminator are provided with an additional piece of information (often called y). According to Mirza and Osindero (2014), such conditioning can take various forms, including class labels (e.g., in MNIST) or attribute vectors (e.g., *wearing glasses*, *hair color*, etc.). Moreover, one could even use textual descriptions, sketches, or segmentation maps to guide the image generation process.

In the particular framework described in the text, we define two key models:

Generator (cG(z, y)).

- A noise vector z is sampled from a prior distribution $P(z)$.
- An attribute vector y (corresponding to real data) is provided as a condition.

- The generator maps (z, y) into a synthesized image $\tilde{x}$:

$$\tilde{x} = \mathrm{cG}(z, y).$$

Discriminator (cD(x, y)).

- Given an image x and a corresponding attribute vector y, the conditional discriminator outputs a value in $[0, 1]$ indicating whether (x, y) is a genuine pair or not.
- The ideal behavior is:

$$D(\tilde{x}, y) = 0 \quad (\tilde{x} \text{ is generated}) \quad \text{and} \quad D(x^*, y) = 1 \quad (x^* \text{ is real}).$$

- Consequently, the discriminator learns to distinguish authentic images from generated images, taking into account the attribute y.

2.2 – cDCGAN Architectures for MNIST

Question 6

Our tests indicated that the conditional DCGAN achieved notably more stable training than the unconditional version. In particular, it generated highly distinct and recognizable images, emphasizing the value of conditioning the model. These findings point to the significant gains in learning efficiency and targeted image generation that arise from integrating conditional variables into the GAN framework. Nonetheless, the trade-off is extended training time compared to a non-conditional GAN, which must be weighed against the greater control and stability in the image synthesis process.

Question 7

Eliminating $\mathbf{y}$ from the discriminator's input would fundamentally change how the conditional cGAN operates. The discriminator would then only verify the realism of the images, without verifying whether they match the intended condition. In other words, it wouldn't be able to decide if a generated digit corresponds to its label.

Although the generator would still receive $\mathbf{y}$, the lack of enforcement from the discriminator to conform to the condition would diminish the incentive for the generator to produce condition-specific outputs. As a result, while the generated images might still look realistic, they would no longer necessarily reflect the specified conditions.

Question 8

Incorporating the conditioning variable $\mathbf{y}$ generally leads to more successful training results than the unconditional setup. By guiding image generation through a specific attribute or class label, the model can produce more diverse and controlled outputs. In the unconditional scenario, the generator has no indication of the type of image it should create, raising the risk of mode collapse—where the generator might produce highly similar or even identical images.

Conversely, in the conditional setting, each label $\mathbf{y}$ corresponds to a distinct attribute or class. The generator, therefore, must produce images that are not only realistic but also accurately reflect the chosen condition. Simultaneously, the discriminator learns to evaluate both the realism and the correctness of the conditional output, fostering a more robust training process and a richer data representation.

Question 9

Here, the noise vector z can be interpreted as the random seed that introduces variability in the generator's outputs. Essentially, z serves as the stochastic component that, when combined with class labels y, allows the generator to create a wide range of images within each specified category.