Training a Generative Adversarial Network: Generating Handwritten-like Images

Nikola Prodanov, June 2025

This project is a practical exploration of Generative Adversarial Networks (GANs), completed as part of the DeepLearning.AI specialization. The goal was to build and train a GAN capable of generating handwritten digit images resembling those from the MNIST dataset. Through this hands-on assignment, I gained a deeper understanding of adversarial training, neural network architecture, and the challenges involved in stabilizing GAN training. The project demonstrates the progressive improvement of generated images over training steps and provides insights into the behavior of the generator and discriminator through loss function analysis.

Introduction

GANs are a class of machine learning models that consist of two competing neural networks: a generator and a discriminator. Introduced by Ian Goodfellow in 2014, GANs are capable of learning to generate new data samples that resemble a given dataset, through an adversarial training process.
In this project, I implement and train a GAN to generate images of handwritten digits, using the MNIST dataset as a reference. The MNIST dataset consists of 60,000 grayscale images of handwritten digits from 0 to 9, see Figure 1, and is widely used as a benchmark for image generation and classification tasks. The objective of the generator is to produce realistic, digit-like images from random noise, while the discriminator learns to distinguish between real MNIST images and those generated by the generator. Over time, the generator improves its ability to produce images that the discriminator cannot easily distinguish from real data.

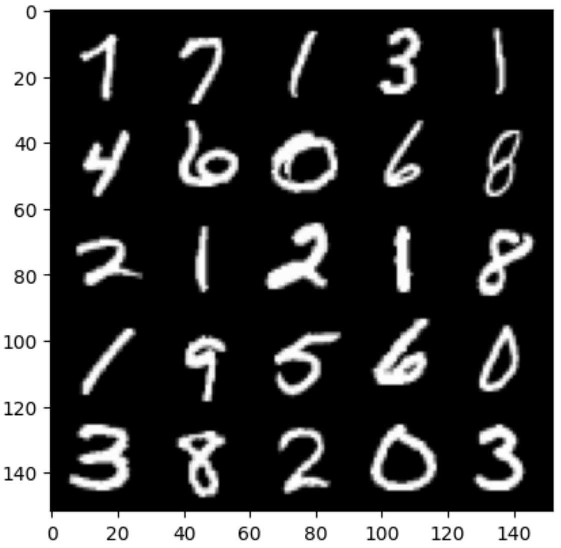


Figure 1: Sample images from the MNIST dataset of handwritten digits from 0 to 9. These real images serve as the training data for the GAN.

GAN Architecture Description

The GAN consists of two core components: the generator and the discriminator, trained adversarially.

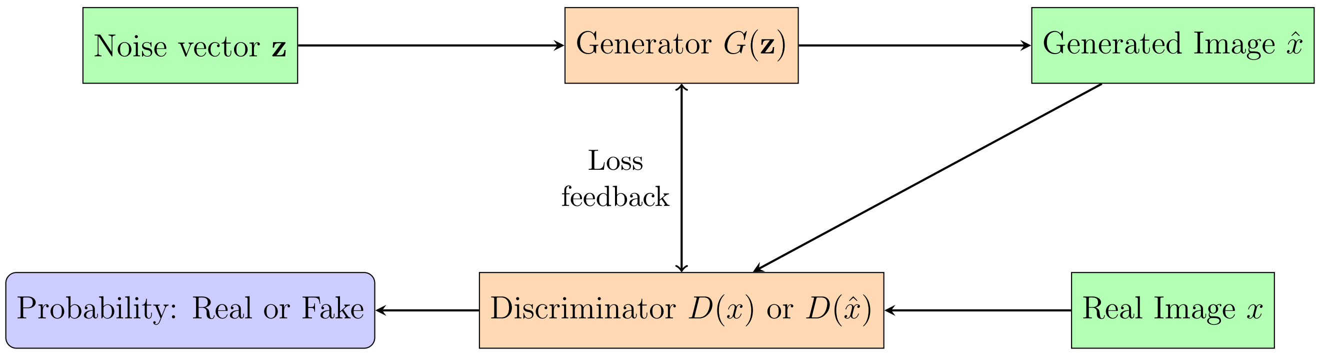

Noise vector $\mathbf{z}$
Generator $G(\mathbf{z})$
Generated Image $\hat{x}$
Loss feedback
Probability: Real or Fake
Discriminator $D(x)$ or $D(\hat{x})$
Real Image x

Figure 2: Overview of the GAN architecture: noise input is transformed by the Generator into a fake image; both generated and real images are fed to the Discriminator, which outputs a probability indicating real or fake.

Figure 2 illustrates the overall architecture and workflow of the GAN used in this project. The process begins with a random noise vector $\mathbf{z}$, sampled from a normal distribution. This noise is input into the **Generator** $G(\mathbf{z})$, a neural network designed to produce images resembling handwritten digits. The Generator outputs a **fake image** $\hat{x}$, which is evaluated alongside a **real image** x from the MNIST dataset by the **Discriminator** D. The Discriminator acts as a binary classifier, receiving both real and generated images and outputting a probability indicating whether each image is real or fake. This output guides the training process: the Discriminator learns to better distinguish real from fake images, while the Generator receives **loss feedback** from the Discriminator, enabling it to improve its ability to create more convincing fake images.
These two networks engage in an adversarial training loop, where the Generator tries to fool the Discriminator, and the Discriminator strives to become more accurate. The feedback loop, depicted as a bidirectional arrow between the Discriminator and Generator, is key to the GAN's ability to progressively generate realistic handwritten digits.

Loss Functions

Training a GAN involves two loss functions: one for the Discriminator and one for the Generator. Both are based on Binary Cross-Entropy (BCE) loss and are implemented using `BCEWithLogitsLoss` in PyTorch:

The Discriminator loss is defined as:

$$\mathcal{L}_D = -\mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\log D(x)\right] - \mathbb{E}_{z\sim p_z(z)}\left[\log\left(1 - D(G(z))\right)\right]$$

The Generator loss is given by:

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

In these equations, $z \sim p_z(z)$ denotes a noise vector sampled from a standard normal distribution, which is used as input to the Generator $G(z)$ to produce a fake image. The term $x \sim p_{\text{data}}(x)$ refers to a real image drawn from the training dataset. The Discriminator D takes either a real image x or a generated image $G(z)$ and returns the probability that the input is real. The Discriminator is trained to maximize its accuracy on distinguishing real from fake images, while the Generator is trained to fool the Discriminator into classifying generated images as real. Thus, the Generator improves by minimizing the log probability that the Discriminator correctly identifies its output as fake. This adversarial training setup allows the Generator to progressively learn how to produce more realistic images.

Results

The GAN was trained on the MNIST dataset using the following settings: training lasted for `200` epochs with a batch size of `128`. The latent noise vector had a dimensionality of `64`, and the learning rate for both the Generator and Discriminator was set to `0.0002`. The training was conducted on a GPU provided by the DeepLearning.AI course.

Before training, the MNIST images were preprocessed through a sequence of PyTorch transforms: they were converted to tensors and normalized to the range $[-1, 1]$ using `transforms.Normalize((0.5,), (0.5,))`. The dataset was loaded with PyTorch's `DataLoader`, using shuffling to randomize the order of batches during training.

In addition to assessing the visual quality of the generated images, the loss functions of both the Generator and Discriminator were monitored to gain insight into the training dynamics. Figure 3 depicts the progression of these loss values over time.

To track the improvement in image quality, generated samples were saved every 500 iterations. Figure **??** presents a selection of these outputs, illustrating the Generator's progressive ability to produce realistic handwritten digits as training advances.

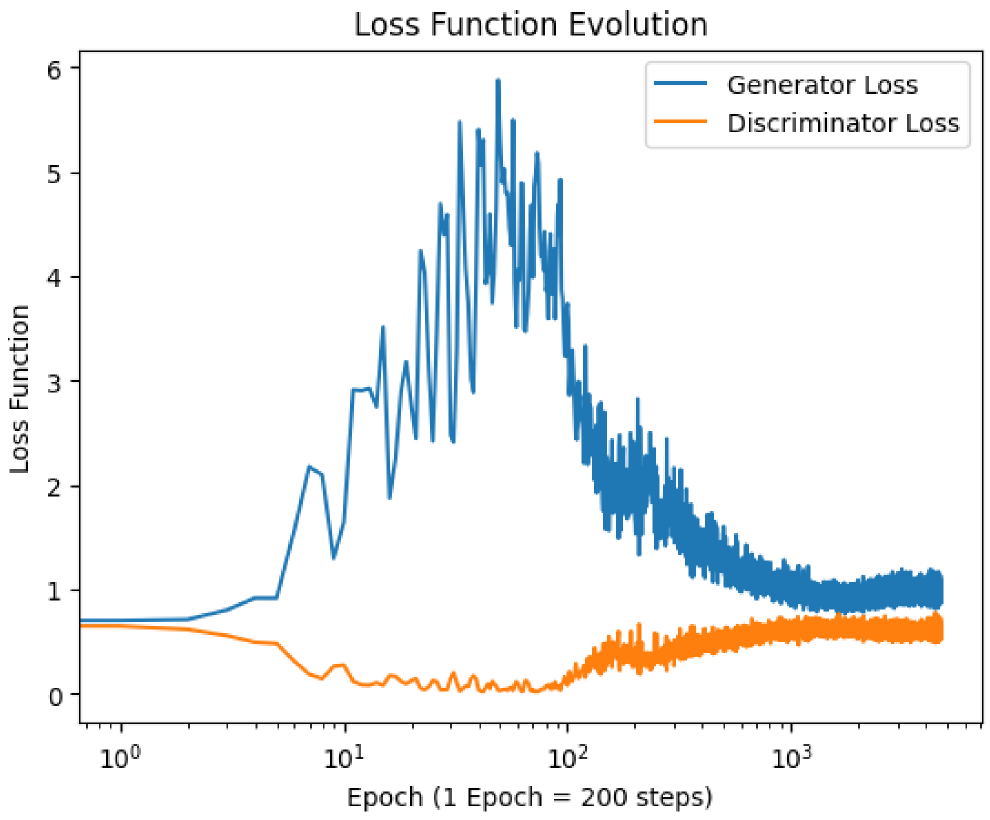
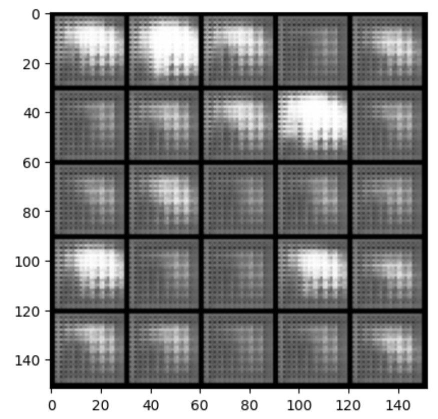
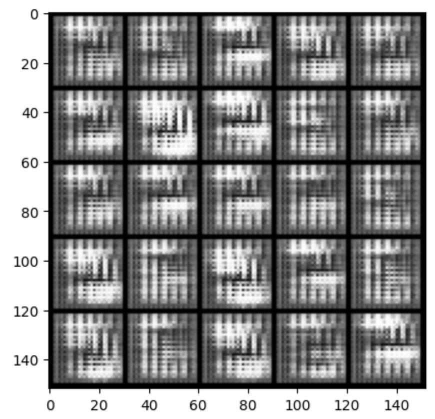


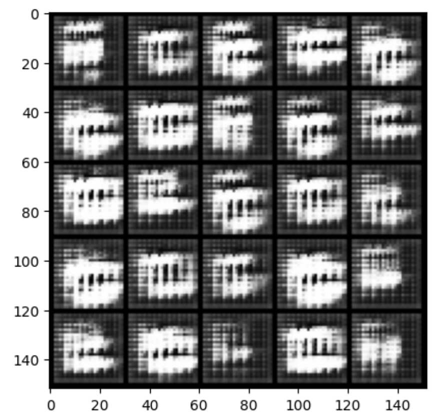
Figure 3: Training loss for Generator and Discriminator across epochs. It can be observed that the Discriminator learns faster initially, but eventually the Generator catches up as training progresses.



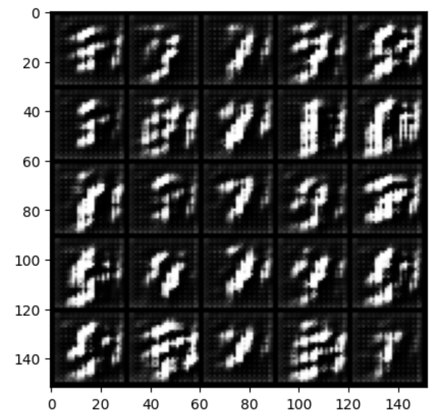
(a) Step 500



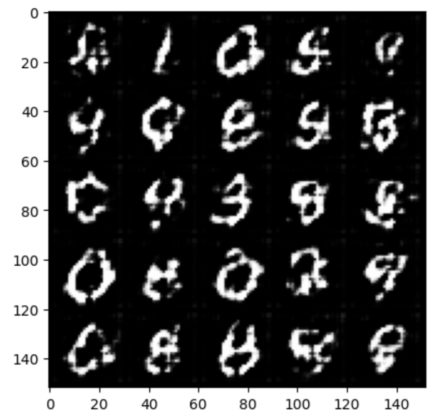
(b) Step 1000



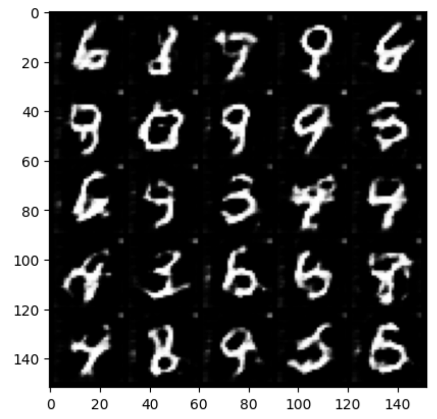
(c) Step 1500



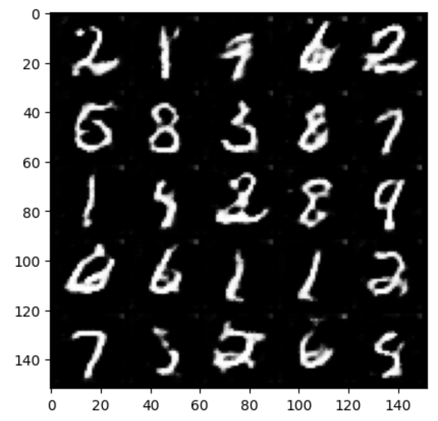
(d) Step 2000



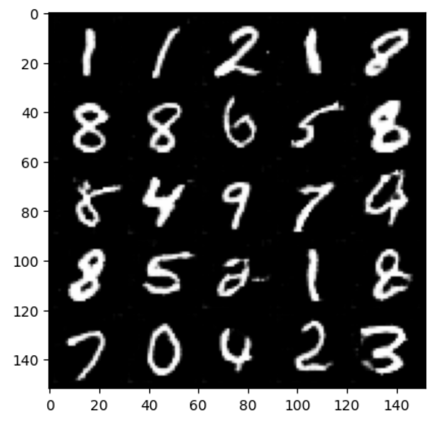
(e) Step 4000



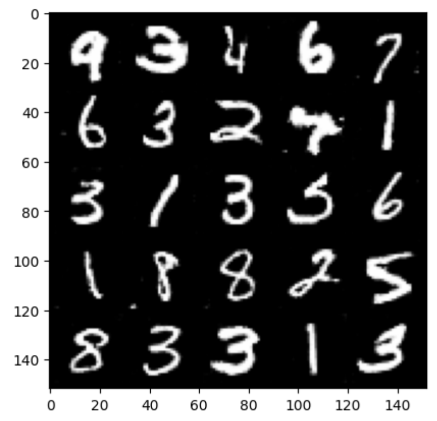
(f) Step 8000



(g) Step 10 000



(h) Step 20 000



(i) Step 90 000

Figure 4: Generated digit samples at different training steps showing the progressive improvement in realism. The numbers begin to take shape around steps 4,000–8,000, become clearly readable by step 10,000, and reach realistic quality around step 90,000, although occasional imperfections remain.

Conclusion

This project implemented a GAN trained on the MNIST dataset to generate realistic handwritten digits. Adversarial training allowed the Generator to improve image quality while the Discriminator enhanced its accuracy. Tracking losses and sample evolution highlighted their dynamic interaction. The results confirm GANs' ability to learn complex data and produce convincing synthetic images.