My Journey with Building a Generative Adversarial Network

My Journey with Building a Generative Adversarial Network

I embarked on this project with the goal of creating a Generative Adversarial Network (GAN) using PyTorch, and along the way, I learned a lot about model design, tweaking parameters, and dealing with the challenges that come with GANs.

Getting Started: Building the Generator and Discriminator

Before building the generator and discriminator, the first reality check I had was with downloading and loading the dataset. It is from that very moment I had to grab water and my jotter because this won't be a walk through project.

The Generator:

At the very beginning, I experimented with a basic setup using simple linear layers paired with ReLU activations. It was a humble start, if only the generator could talk, I was sure it would have been telling me "Dey Play!" but I quickly realized that this approach wasn?t giving (the kind of detailed images I was hoping for). After some trial and error, I refined the generator into a more robust network made up of several fully connected layers. I kept the ReLU activations but added a final Tanh activation at the end. This tweak was important because it helped scale the output correctly, turning a 100 dimensional latent vector into a neat 28×28 image perfect for the MNIST dataset.

The Discriminator:

On the other hand, I designed the discriminator to be a bit more complex. Using a series of convolutional layers, I was able to capture the spatial details of the images more effectively. With LeakyReLU activations and batch normalization, the network stayed stable throughout training. The last step was to apply a Sigmoid

My Journey with Building a Generative Adversarial Network

function, which lets the discriminator output a probability indicating whether the image was real or generated.

Key Hyperparameters:

To keep things consistent and reproducible, I set up a few crucial parameters, at first these were not the parameters used?it had to be tweaked so that the models can function properly:

- Latent Dimension: 100
- Batch Size: 128
- Learning Rate: 0.0002
- Optimizer: Adam with $\beta_1 = 0.5$ and $\beta_2 = 0.999$
- Number of Epochs: 50

Also, I made sure to normalize the images using a transformation pipeline, and I downloaded the MNIST dataset locally so that I wouldn't run into any data access issues.

Tackling Training Stability: Hard Lessons Learned

Early Struggles:

Using the linear based generator, the generated images were, frankly, disappointing. They lacked everything, the generated images were just like black dots with nothing to show. I spent a lot of time trying to tweak the hyperparameters when I should have just dumped the linear generator.

What Changed:

Through a lot of dedicated tweaking and modifications, a few key changes turned things around:

- Architecture Upgrades: After dumping the linear based generator, I decided to use ConvTranspose2d and

BatchNorm2d. I revamped the generator with additional layers and, importantly, the Tanh activation at the

end. This made a noticeable difference in how the images turned out.

- Dataset Management: By downloading the MNIST dataset locally, I avoided any errors related to connectivity or data inconsistency.

- Hyperparameter Tweaks: Finetuning the learning rate and adjusting the Adam optimizer's settings helped me bring balance between the generator and discriminator, reducing the gap in their losses and leading to a smoother training process.

Seeing the Results: Real vs. Generated Images

As training progressed, I kept a close eye on the outputs. Here's what I observed:

- Generated Images:

At first, the images were rough and unpolished. However, as I continued to refine the network, the generator started capturing the basic shapes and features of handwritten digits. Even though the images weren't perfect, they showed clear signs of improvements after each epoch.

- Real Images:

I regularly compared these generated images with real MNIST samples. This side-by-side comparison was incredibly useful. It allowed me to visually assess how far the network had come and how the generated images improved after each epoch.

Reflecting on the Experience: Practical Insights

Building and fine-tuning a GAN wasnt without its challenges. Here are a few insights from my journey:

- Sensitivity to Changes:

GANs are notoriously volatile. Even the smallest modification in the architecture or hyperparameters can lead to significant changes in performance. This project was a constant learning experience, with each

experiment providing valuable feedback.

- Importance of Data Management:

Keeping the MNIST dataset local was a simple but key step. It meant I could focus on tweaking the model rather than worrying about external data issues.

- Iterative Improvements:

The whole process was very iterative. I started with a basic model, observed its shortcomings, and then gradually introduced enhancements. Regular visual feedback was very important. It was my guiding light throughout the entire training process.

- Stability and Monitoring:

Maintaining a balance between the generator and discriminator wasn't easy, but monitoring their progress closely helped me make the right adjustments when necessary.