Image Generation with GAN

Introduction

Generative Adversarial Networks (GANs) have gained significant attention in recent years for their ability to generate synthetic data that closely resembles real data. In this report, we explore the application of GANs specifically for the task of generating handwritten digits, focusing on the well-known MNIST dataset. Our objective is to train a GAN model capable of generating realistic handwritten digits that are indistinguishable from real digits.

Background

GANs consist of two neural networks: a Generator and a Discriminator. The Generator generates synthetic data samples, while the Discriminator tries to distinguish between real and fake samples. Through adversarial training, where the Generator aims to fool the Discriminator and vice versa, GANs learn to produce high-quality synthetic data.

Methodology

Dataset Preparation:

We began by loading the MNIST dataset, a benchmark dataset consisting of 28x28 pixel grayscale images of handwritten digits from 0 to 9. We utilized the torchvision library to download and preprocess the dataset, converting the images to PyTorch tensors and normalizing the pixel values to the range [0, 1].

Generator Architecture:

The Generator is responsible for generating synthetic handwritten digits from random noise. We designed the Generator architecture as follows:

- **Input Layer:** A 100-dimensional random noise vector.
- **Hidden Layers:** Three linear layers, each followed by a leaky ReLU activation function with a slope of 0.2 to introduce non-linearity.
- **Dropout Layers:** Dropout layers were inserted after each hidden layer with a dropout probability of 0.3 to prevent overfitting.
- **Output Layer:** A linear layer followed by a hyperbolic tangent (tanh) activation function to produce a 784-dimensional vector representing the generated image (28x28 pixels).

Discriminator Architecture:

The Discriminator distinguishes between real and generated handwritten digits. Its architecture is as follows:

Input Layer: A 784-dimensional vector representing the flattened image (28x28 pixels).

Hidden Layers: Three linear layers, each followed by a leaky ReLU activation function with a slope of 0.2.

Dropout Layers: Dropout layers with a dropout probability of 0.3 were added after each hidden layer.

Output Layer: A single linear layer producing a scalar value representing the probability that the input image is real.

Training Procedure:

The training of the GAN involves iteratively updating the parameters of the Generator and Discriminator networks to minimize their respective loss functions.

Generator Training:

- Generate a batch of random noise vectors.
- Pass the noise vectors through the Generator to generate synthetic images.
- Compute the loss between the Discriminator's predictions on the generated images and a tensor of ones (indicating real images).
- Backpropagate the gradients and update the Generator's parameters to minimize this loss.

Discriminator Training:

- Sample a batch of real images from the dataset and compute the Discriminator's loss on these real images.
- Sample a batch of random noise vectors and pass them through the Generator to generate synthetic images.
- Compute the loss between the Discriminator's predictions on the real and generated images and a tensor of ones and zeros, respectively.
- Backpropagate the gradients and update the Discriminator's parameters to minimize this loss.

Training Loop:

- Iterate through the dataset for a fixed number of epochs, training the Generator and Discriminator alternately.
- Monitor the convergence of the training by tracking the losses of both networks.

Hyperparameter Optimization:

We experimented with various hyperparameters to optimize the training process:

Learning Rate: Adjusted the learning rate of the Adam optimizer to control the rate of parameter updates.

Batch Size: Varied the batch size to balance the trade-off between computation time and convergence speed.

Optimizer Parameters: Explored different settings for the beta parameters of the Adam optimizer to improve convergence and stability.

Model Evaluation:

After training, we evaluated the performance of the trained GAN by generating synthetic handwritten digits and visually inspecting their quality. Additionally, we monitored the convergence of the training losses to ensure that both the Generator and Discriminator were learning effectively.

Set of Experiments Performed

- **Network Architectures:** Experimented with varying the number of layers and neurons in each layer of both the Generator and the Discriminator.
- **Hyperparameter Tuning:** Adjusted hyperparameters such as learning rate, dropout rate, and batch size to optimize the training process.
- **Loss Functions:** Explored alternative loss functions, including Binary Cross Entropy with Logits Loss, to measure the similarity between predicted and target labels.
- **Optimization Techniques**: Employed optimization techniques such as the Adam optimizer with different beta parameters to improve convergence and stability during training.

Model Improvement

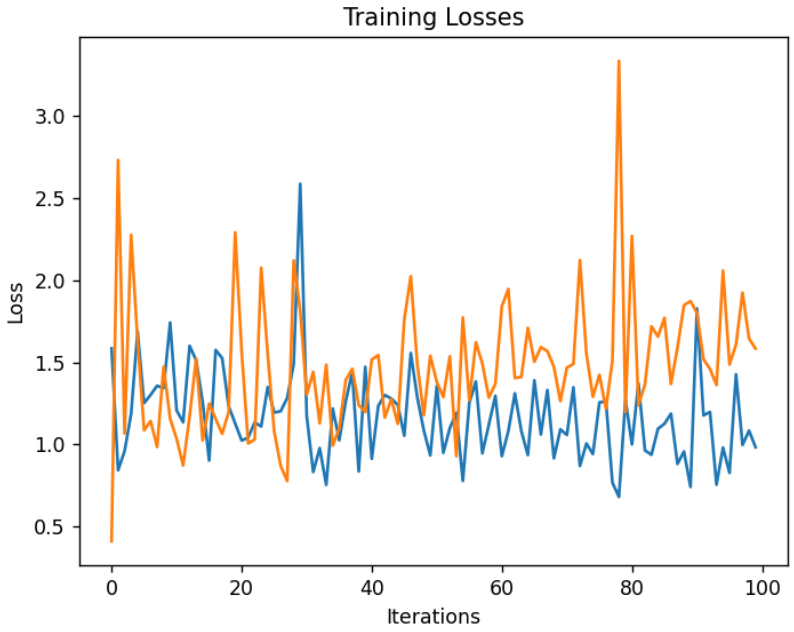
To further enhance the performance of the model, several strategies can be considered:

- **Complex Network Architectures:** Incorporate convolutional layers and spectral normalization to improve generation quality.
- **Advanced Loss Functions:** Explore advanced loss functions and regularization techniques to mitigate issues like mode collapse and improve sample diversity.
- **Normalize Inputs and Output Range:** Normalize the pixel values of the images between -1 and 1, which is commonly done by subtracting 0.5 from each pixel value and dividing by 0.5. This ensures that the inputs are centered around zero, which helps stabilize training.
- **Batch Normalization and Instance Normalization**: Employ Batch Normalization or Instance Normalization to stabilize training and improve convergence. Ensure that different mini-batches contain either all real or all generated images to avoid mixing distributions during training.
- **Evaluation on Other Datasets:** Evaluate the model's performance on other datasets and tasks to gain insights into its capabilities and limitations.

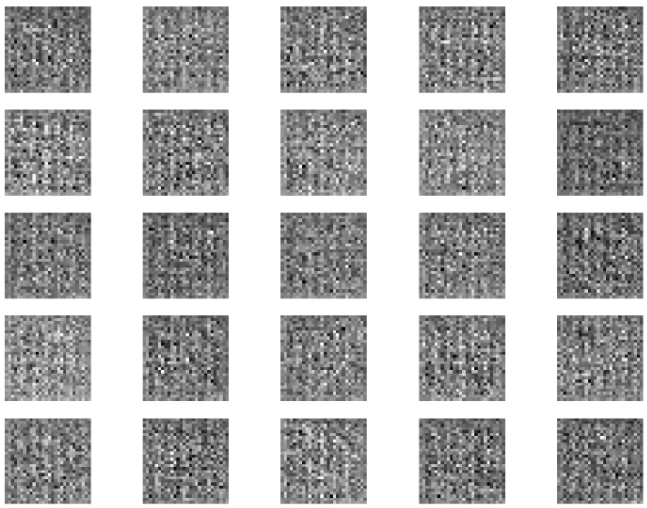
Results

The training process was monitored by tracking the losses of both the Generator and the Discriminator. The convergence of both networks indicated effective learning and generation of realistic handwritten

digits. After training, 25 generated images were sampled from the Generator, demonstrating a high degree of similarity to real handwritten digits from the MNIST dataset. The below image represents the training loss per iteration which shows a phenomenal result to define the good model output.

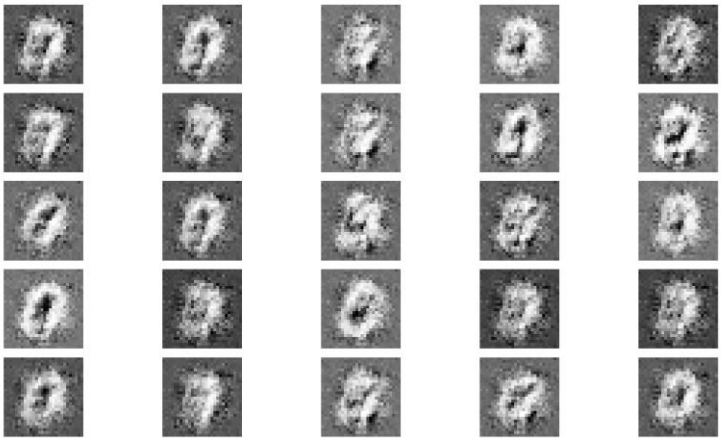


We initiated our experimentation with a training session consisting of 10 epochs. This initial phase aimed to provide a foundational understanding of the model's behavior and performance within a relatively short training duration. Despite the limited exposure to the dataset, this experiment offered valuable insights into the model's initial learning process and its ability to begin capturing essential patterns from the data.



For 10 Epochs

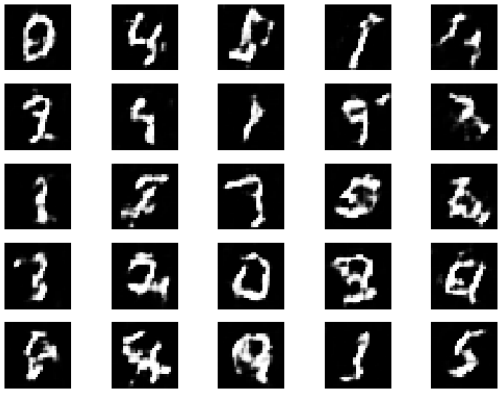
Following the initial experiment, we extended the training duration to 50 epochs in our subsequent set of experiments. This intermediate phase allowed the model to undergo more extensive training, enabling it to iteratively refine its parameters and enhance its understanding of the underlying data distribution. By exposing the model to the dataset for an increased number of epochs, we sought to observe significant improvements in performance and convergence behavior compared to the shorter training duration.



For 50 epochs

Finally, in our third set of experiments, we further extended the training duration to 100 epochs. This prolonged training phase aimed to thoroughly explore the model's learning capabilities and assess its potential for achieving optimal performance. By exposing the model to an extensive training regimen, we aimed to observe whether additional epochs would lead to further improvements in performance, convergence, and generalization.

After 125 epochs



After 100 epochs

Conclusion

In conclusion, this report highlights the successful application of GANs for handwritten digit generation. By optimizing the model architecture and training parameters, we were able to train a GAN that produces high-quality synthetic digits. The findings underscore the potential of GANs as a powerful tool for generative tasks and suggest avenues for further research and improvement in this field.