**GAN Training on MNIST Dataset**

**Background and Methodology:**

Generative Adversarial Networks (GANs) are cutting-edge deep learning models known for their ability to generate high-quality synthetic data. In this project, we applied GANs to the MNIST dataset, a widely used benchmark in machine learning. The goal was to train a GAN to generate realistic handwritten digits, leveraging the adversarial training paradigm where a generator and a discriminator are trained simultaneously.

**Dataset and Task Description:**

The MNIST dataset consists of 60,000 training images and 10,000 test images of handwritten digits (0-9).

A yellow and blue squares

Description automatically generated with medium confidence

Our task involved training a GAN to generate new images that closely resemble the ones in the dataset. By doing so, we aimed to produce synthetic handwritten digits indistinguishable from real ones.

**Implementation Details:**

1) Data Preprocessing:

Image Scaling: Before feeding the images into the GAN model, we performed image scaling to normalize the pixel values. The pixel values of the input images were scaled to the range [-1, 1]. This normalization step helps stabilize the training process and improves the convergence of the model.

Batch Processing: To enhance computational efficiency during training, we batched the dataset into batches of size 256. Batch processing allows the model to process multiple images simultaneously, leading to faster convergence and more stable training dynamics.

Noise Injection: In addition to scaling the images, we introduced noise to the discriminator’s inputs. By adding noise to both real and fake images, we slowed down the discriminator’s learning process, ensuring a balanced training dynamic between the discriminator and generator.

2) Model Architecture:

Generator:

The generator architecture consisted of fully connected layers followed by reshape operations to produce 28x28 images.

We used LeakyReLU activation functions to introduce non-linearity and prevent the vanishing gradient problem.

The final layer of the generator utilized the tanh activation function to ensure that the generated images were in the range [-1, 1].

A screenshot of a computer

Description automatically generated

Discriminator:

The discriminator was designed as a binary classifier with flattened layers followed by fully connected layers.

LeakyReLU activation functions were employed to introduce non-linearity and improve the discriminative capability of the model.

The final layer of the discriminator used the sigmoid activation function to produce a probability score indicating the likelihood of the input image being real.

A screenshot of a computer program

Description automatically generated

3) Training Process:

Adversarial Training:

The training process involved alternating between training the discriminator and the generator.

During the discriminator training phase, real and fake images were fed into the discriminator, and binary cross-entropy loss was calculated to train the discriminator to distinguish between real and fake images.

A screenshot of a computer

Description automatically generated

Subsequently, during the generator training phase, the generator produced fake images, and the discriminator’s feedback was used to update the generator’s weights. The generator aimed to produce images that could fool the discriminator into classifying them as real.

4) Loss Functions and Optimization:

Binary cross-entropy loss functions were utilized for both the discriminator and the generator.

Adam optimizer with a learning rate of 0.0002 and a beta\_1 value of 0.5 was employed for optimization.

**Challenges and Solutions:**

Training Time with CNNs: Initially, we experimented with CNN architectures for both the generator and discriminator. However, training with CNNs resulted in longer training times due to the complex nature of convolutional operations. To expedite training, we switched to dense layers for both the generator and discriminator.

Discriminator Training Speed: One crucial aspect we encountered was the balance between the training speeds of the discriminator and generator. Slowing down the training of the discriminator was essential to prevent it from overpowering the generator too quickly. We achieved this by introducing noise to the discriminator’s inputs. By adding noise to the real and fake images fed into the discriminator, we forced it to learn more slowly, allowing the generator more time to improve its performance. This strategy helped to stabilize the training process and ensure a more balanced adversarial learning dynamics.

**Results:**

During the initial training phases, the generated images exhibited artifacts and lacked clarity. However, as training progressed, the quality of the generated images improved significantly. After approximately 150 epochs, the generated images began to closely resemble the digits 0-9 from the MNIST dataset. The digits became clearer, and the generated samples exhibited more distinct features characteristic of handwritten digits. This improvement over the training epochs demonstrated the effectiveness of the GAN model in learning and generating realistic handwritten digits.

A graph of a number of data

Description automatically generated with medium confidence

The graph above depicts the discriminator and generator loss over the training epochs. The generator loss (g\_loss) started at 1.4, increased to 3.0, and then gradually decreased to 0.5 around 150 epochs. Subsequently, the generator loss remained relatively stable around 0.5 for the remaining epochs, indicating that the generator’s performance improved steadily throughout the training process.

On the other hand, the discriminator loss (d\_loss) started at 0.5 and gradually increased to 0.75 over the first 50 epochs. Afterward, the discriminator loss maintained a relatively stable value around 0.75 for the remaining training epochs. This behavior suggests that the discriminator’s ability to distinguish between real and fake images remained consistent throughout the training process, contributing to the overall stability of the GAN training.

Images generated before training.

A green and blue pixelated square

Description automatically generated

Images generated after training for 350 epochs.

A group of numbers in squares

Description automatically generated

**Improvement Methods:**

In our implementation, we adopted several strategies to enhance the training stability and accuracy of the GAN model:

Image Scaling and Batch Processing: We scaled the pixel values of the input images to the range [-1, 1], which is a common preprocessing step for GANs. Additionally, we batched the dataset into batches of size 256 to improve computational efficiency during training.

Dropout in Discriminator: To prevent overfitting and improve generalization, we introduced dropout regularization in the discriminator network. Dropout randomly drops a certain percentage of neurons during training, forcing the model to learn more robust features.

Optimization Techniques: We employed the Adam optimizer with a learning rate of 0.0002 for both the generator and discriminator. Adam is an adaptive optimization algorithm that adjusts the learning rate during training to optimize convergence.

Binary Cross-Entropy Loss: Binary cross-entropy loss functions were utilized for both the generator and discriminator. This loss function is well-suited for binary classification tasks and helps guide the training process towards generating more realistic images.

Hyperparameter Fine-Tuning: We fine-tuned hyperparameters such as the learning rate and the number of hidden layers in both the generator and discriminator networks. These adjustments were crucial for achieving optimal convergence and model performance.

Noise Injection in Discriminator: To ensure a balanced training process between the discriminator and generator, we introduced noise to the discriminator’s inputs. By adding noise to both real and fake images, we slowed down the discriminator’s learning process, giving the generator more time to learn and improve its performance. This strategy helped stabilize the adversarial training dynamics and prevent the discriminator from overpowering the generator prematurely.

These techniques collectively contributed to enhancing the stability, convergence, and overall performance of the GAN model on the MNIST dataset.

**Contributions:**

Anirudh Ashok Patil

1. Data Pre-processing,
2. Generator model,
3. Hyper parameter tuning
4. Report.

Manoj Tirukovela

1. Discriminator Model,
2. tensorflow.keras.model for training GAN,
3. Hyper parameter tuning
4. Report.