Exploring Handwritten Digit Generation with Conditional Generative Adversarial Networks

Atharva Tiwari
IIT Gandhinagar
Deep Learning

27/02/2024

Abstract

This report delves into the application of Conditional Generative Adversarial Networks (CGANs) for the generation of handwritten digits, aiming to tackle the challenges of synthetic data generation in a controlled manner. CGANs are explored due to their ability to generate new data instances that mimic real data, conditioned upon a class label. The efficacy of CGANs in generating diverse and realistic digits offers significant promise for enhancing data augmentation techniques in machine learning workflows, thereby aiding in the robust training of models where real data is scarce or expensive to obtain.

1 Introduction

Handwritten digit generation is a critical task in the field of computer vision, with implications across educational technology, data augmentation for deep learning models, and secure digit verification systems. Traditional methods of digit recognition and generation relied heavily on extensive datasets, which are often limited in diversity and volume. Generative Adversarial Networks (GANs), and specifically their conditional variants (CGANs), present a novel solution by generating new, realistic samples of data based on learned data distributions.

2 Problem Statement

Despite the advancements in generative models, the challenge of producing high-quality, diverse handwritten digits that are indistinguishable from real digits remains. This project focuses on the application of CGANs to generate such digits, aiming to evaluate the quality and diversity of the generated images and explore their potential in practical applications such as improving the performance of optical character recognition (OCR) systems.

3 Theoretical Background

3.1 Generative Adversarial Networks

GANs are a class of artificial intelligence algorithms used in unsupervised machine learning, implemented by a system of two neural networks contesting with each other in a zero-sum game framework. This setup helps learn to generate new data with the same statistics as the training set.

3.2 Conditional Generative Adversarial Networks

CGANs modify the GAN architecture by adding the condition variable to both the generator and discriminator. This approach allows the generation of targeted data samples, providing control over the modes of the data creation process.

3.3 Relevance of CGANs to Digit Generation

CGANs leverage the label information to generate digits that are not only realistic but also specific to given label conditions, thus enhancing control and quality of the generation process.

4 Methodology

4.1 Model Architecture

4.1.1 Generator

The generator architecture is designed to incorporate both noise and conditional labels:

Input Layer: Concatenation of noise $z \sim \mathcal{N}(0, 1)$ and label y,
Hidden Layers: Fully connected layers with ReLU activation, batch normalization,
Output Layer: Fully connected layer with Tanh activation.

4.1.2 Discriminator

The discriminator architecture assesses the authenticity of images, conditioned on labels:

Input Layer: Concatenation of image and label,
Hidden Layers: Fully connected layers with LeakyReLU activation, batch normalization,
Output Layer: Sigmoid activation.

4.1.3 Visual Representation of CGAN Architecture

Below is a visual representation of the Conditional Generative Adversarial Network (CGAN) architecture used in this project:

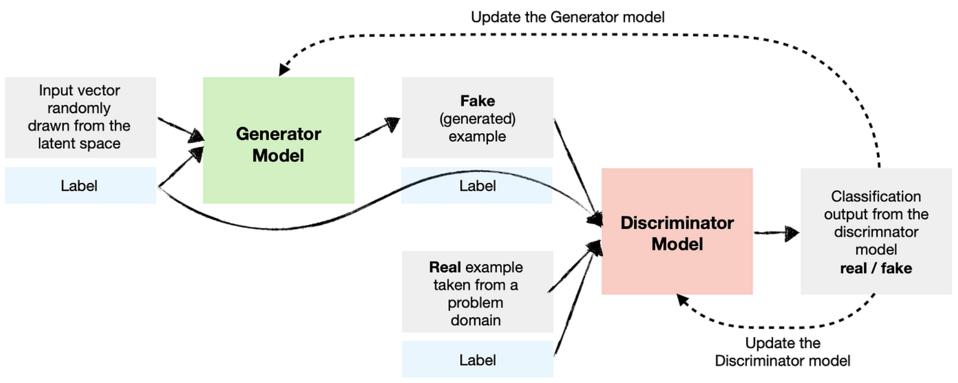
Update the Generator model
Input vector randomly drawn from the latent space
Label
Generator Model
Fake (generated) example
Label
Discriminator Model
Classification output from the discrimnator model real / fake
Real example taken from a problem domain
Label
Update the Discriminator model

Figure 1: Architectural diagram of the Conditional Generative Adversarial Network (CGAN).

4.2 Training Procedure

The training of CGANs follows an iterative process where both networks improve concurrently:

1. The discriminator trains by alternating between real and generated data, adjusting its parameters to maximize its accuracy in distinguishing between the two sources.
2. The generator updates its parameters based on the gradient of $\log(1 - D(G(z|y)))$, promoting deception to fool the discriminator into believing that the samples it generates are real.

This training methodology ensures that the generator produces increasingly realistic data as training progresses, conditioned on the specified labels.

4.3 Mathematical Foundation

4.3.1 Generator and Discriminator in GANs

Generative Adversarial Networks (GANs) consist of two neural networks, the generator (G) and the discriminator (D), which are trained simultaneously in a game-theoretic scenario. The generator aims to map an input noise vector z from a prior noise distribution $p_z(z)$ to the data space as $G(z; \theta_g)$, where θ_g are the parameters of G. The discriminator, $D(x; \theta_d)$, outputs a scalar representing the probability that x came from the training data rather than G. The objective of a GAN can be represented as a minimax game with the following value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

4.3.2 Modifications in CGANs

Conditional GANs modify the generator and discriminator to condition on an additional label y, enhancing the model's ability to direct the data generation process. This conditioning is achieved by modifying the input of both the generator and discriminator to include the label y. The modified objective function is:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x|y)] + \mathbb{E}_{z \sim p_z(z), y \sim p_y(y)}[\log(1 - D(G(z|y)))]$$

In this setting, G and D learn a conditional model where both are provided with label information to ensure that the generated images correspond to specific classes.

5 Implementation

5.1 Architecture Details

5.1.1 Generator

The generator combines a 100-dimensional noise vector with a 10-dimensional one-hot encoded label, resulting in a series of fully connected layers that output a 784-dimensional vector, reshaped into a 28x28 image.

5.1.2 Discriminator

The discriminator evaluates 784-dimensional vectors concatenated with 10-dimensional labels, using fully connected layers to output a single probability.

5.2 Optimization

Adam optimizer with learning rate 0.0002 and betas (0.5, 0.999) was used to minimize both generator and discriminator losses.

5.3 Mathematical Justification of Architectural Choices

The architectural and functional choices within the CGAN framework are meticulously designed to ensure stability, efficiency, and effectiveness in the learning process, addressing common challenges in training deep neural networks.

5.3.1 Batch Normalization

Batch Normalization (BN) is a technique to improve the training of deep neural networks that standardizes the inputs to a layer for each mini-batch. This addresses the issue of internal covariate shift, where the distribution of network activations varies significantly during training, slowing down the training process and necessitating lower learning rates. Mathematically, BN transforms an input x into:

$$y = \gamma \left(\frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \beta$$

where μ_B and σ_B^2 are the mean and variance of a batch, γ and β are learnable parameters, and ϵ is a small constant to avoid division by zero. This normalization helps maintain a stable distribution of activations throughout the training process, contributing to faster convergence and improved training dynamics.

5.3.2 Leaky ReLU and Tanh Activations

The Leaky ReLU activation function is used instead of the standard ReLU to mitigate the problem of dying neurons, where neurons permanently deactivate during training, resulting in model capacity reduction. It is defined as:

$$f(x) = \max(0.01x, x)$$

This slight slope for negative values keeps all neurons somewhat active, maintaining a healthy gradient flow through the architecture.

The Tanh activation function in the generator's output layer scales the output to the range $[-1, 1]$, aligning with the pre-processed input data of the model. The Tanh function is mathematically expressed as:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

This activation helps in learning the complex data distribution effectively by ensuring the generated outputs match the scale and distribution of the training data, facilitating smoother gradients during backpropagation.

5.3.3 Binary Cross-Entropy (BCE) Loss

The Binary Cross-Entropy Loss is crucial for training GANs, where it measures the distance between the distribution of the data generated by the GAN and the real data distribution. For discriminator D and generator G, the BCE loss is applied as follows:

$$L_D = -\mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] - \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$$
$$L_G = -\mathbb{E}_{z\sim p_z}[\log D(G(z))]$$

These equations encourage the discriminator to correctly classify real and generated data, while the generator aims to produce data that the discriminator mistakenly believes to be real. This setup creates a dynamic feedback loop, driving both networks towards optimal performance through adversarial training.

6 Evaluation Metrics

6.1 Fréchet Inception Distance (FID)

6.1.1 Definition and Importance

The Fréchet Inception Distance (FID) is a metric used to evaluate the quality of images generated by GANs. It calculates the distance between feature vectors calculated for real and generated images. The lower the FID score, the more similar the distributions of the generated images to the real images, implying better generation quality.

6.1.2 Mathematical Formulation

The FID score is computed by comparing the statistics of generated images to those of real images using the following equation:

$$\text{FID} = ||\mu_r - \mu_g||^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2})$$

where:

- μ_r and μ_g are the feature-wise means of the real and generated images, respectively.
- Σ_r and Σ_g are the covariance matrices of the real and generated images, respectively.
- Tr denotes the trace of a matrix (sum of diagonal elements).

The feature vectors are typically obtained from an intermediate layer of the Inception v3 model, which is pretrained on ImageNet. The Inception model captures high-level features of the images, making these features suitable for assessing the quality of generated images.

6.1.3 Implementation in Code

In the provided implementation, real and generated images are processed through the Inception model to obtain their feature vectors. These vectors are then used to compute the FID score as follows:

1. Compute the mean and covariance of the feature vectors for both real and generated images.
2. Calculate the squared difference between the means of the real and generated features.
3. Compute the trace of the sum of the covariance matrices minus twice the square root of the product of the covariance matrices.
4. Sum the above two quantities to obtain the FID score.

This metric provides a robust measurement of similarity between the generated image distribution and the real image distribution, which is critical for assessing the performance of generative models in producing realistic images.

6.1.4 Conversion of Grayscale Images to RGB for FID Calculation

The Fréchet Inception Distance (FID) score, commonly utilized to evaluate the quality of images generated by GANs, necessitates the use of the Inception V3 model. This model was originally trained with RGB images from the ImageNet dataset and thus expects three-channel input for accurate performance.

Necessity for RGB Conversion When employing datasets that consist of grayscale (single-channel) images, such as MNIST, it becomes essential to adapt these images to match the expected input format of the Inception V3 model. This adaptation involves converting the grayscale images into RGB format by replicating the single grayscale channel across the three RGB channels. This process ensures that the input retains the original image content but aligns with the model's input requirements.

Technical Implementation The conversion is typically implemented in the following manner in the codebase:

```
def to_rgb(x):
    # x should be of shape (N, 1, H, W)
    return x.repeat(1, 3, 1, 1)  # Repeat the channel dimension
```

Here, `x` represents the input tensor of grayscale images with dimensions $(N, 1, H, W)$, where:

- N is the number of images,
- 1 indicates a single color channel,
- H and W are the height and width of the images, respectively.

The method `repeat(1, 3, 1, 1)` effectively replicates the single channel three times along the channel axis, converting each grayscale image into a pseudo-RGB image. This transformation is crucial as it allows the pre-trained Inception V3 model to process the images and generate meaningful features for the FID computation.

Impact on FID Score By ensuring the images are in RGB format, the Inception V3 model can accurately process the input and extract robust features that are essential for computing the FID score. The FID score then quantitatively measures the similarity between the generated images and real images based on these deep features, thus reflecting the visual quality of the generated images in a more reliable manner.

6.1.5 Qualitative Assessment

Generated images were visually inspected to assess diversity and realism. Sample images from different training epochs demonstrate the progression of the generator's ability.

7 Results

7.1 Performance Evaluation of Conditional GAN on MNIST

The training process of the Conditional Generative Adversarial Network (CGAN) on the MNIST dataset shows varying trends for the discriminator and generator losses over epochs. Initially, the discriminator's loss decreases, suggesting it is learning effectively to differentiate between real and fake images. However, as training progresses, the discriminator's loss begins to increase slightly, indicating a challenge in maintaining its discriminative performance against an improving generator.

The generator's loss, conversely, shows an increasing trend initially, which is expected as it struggles to fool the discriminator. Over time, the generator's loss starts to decrease, reflecting its growing capability in generating images that are more convincing to the discriminator. This oscillation in loss values is typical in GAN training dynamics, illustrating the adversarial push and pull between the generator and discriminator as they both evolve.

The training dynamics underscore the challenges in stabilizing GAN training, where achieving a balance where neither the generator nor discriminator becomes too powerful too quickly is crucial for model convergence. This behavior is a direct consequence of the adversarial training setup, where the two networks compete and thereby improve each other.

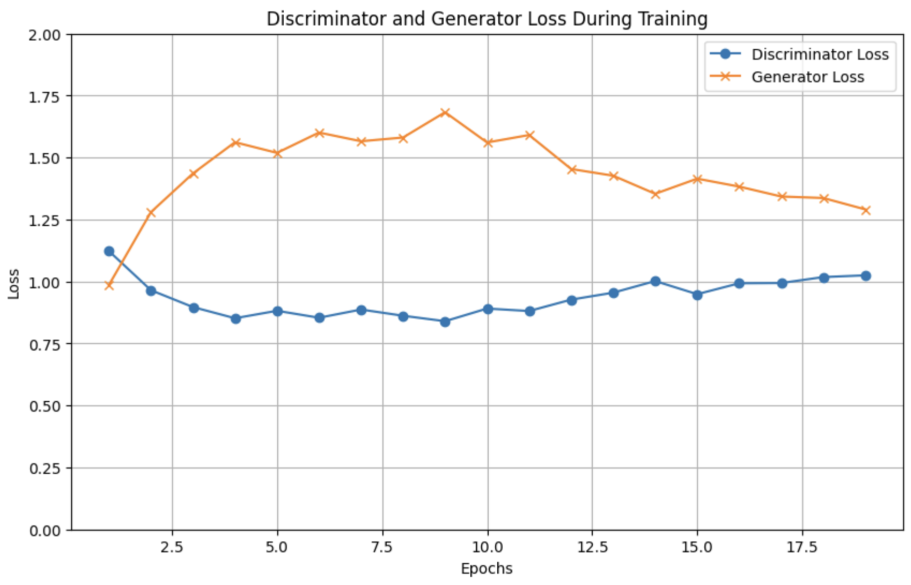

Figure 2: Discriminator and Generator Losses over Epochs

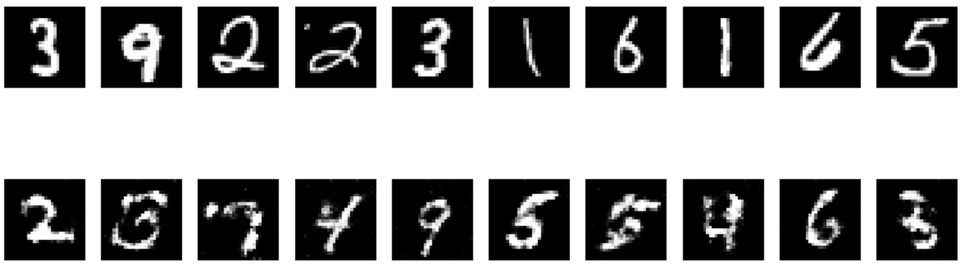

Figure 3: Real(upper row) VS Fake(lower row) images

8 Conclusion

CGANs represent a powerful tool for generating realistic, label-conditioned handwritten digits. This technology not only aids in data augmentation but also has potential applications in educational technology and secure digit verification systems. Future work will focus on scaling the model to handle more complex image datasets and exploring the integration of CGANs into semi-supervised learning frameworks.

9 Citations

References

[1] J. Hui, "GAN: How to measure GAN performance," Medium, Apr. 5, 2018. [Online]. Available: `https://jonathan-hui.medium.com/gan-how-to-measure-gan-performance-64b988c47732`. [Accessed: Jul. 19, 2024].

[2] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," *arXiv preprint arXiv:1411.1784*, Nov. 2014. [Online]. Available: `https://arxiv.org/abs/1411.1784`.