Comparative Analysis of Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) on MNIST Dataset

Thilak Reddy Kanala

University of Florida
tkanala@ufl.edu

Abstract

The aim of this research project is to compare Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) on the MNIST dataset, with the goal of understanding each technique well enough to customize them for new problems. Both techniques are generative models that work on the same dataset to fit a probability distribution, and the project seeks to identify differences in network size, training time, and errors in generated examples. This report includes brief overview of the architectures, along with implementations details and a discussion of the nature of the qualitative aspects of the generated output along with the quantitative aspect of the model performance.

1. Introduction

The report compares the performance of Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) on MNIST and Fashion MNIST datasets. The models were trained and tested using Google Colab GPUs and evaluated based on parameters such as network size and objective function. The aim is to gain insights into the strengths and weaknesses of each technique for generative modeling. The base code for VAE and GAN was referenced from different research papers and articles. [1, 2, 3]

The structure of this paper is as follows. Section 2 includes a detailed review of the background related to generative modeling, GANs, and VAEs. Section 3 describes the methodology used for training and testing both models on the MNIST data sets. Section 4 presents the results of the experiments and compares the performance of GANs and VAEs based on various parameters. Finally, in Section 5, the discussion is concluded by summarizing the main findings and contributions.

2. Background

Generative modeling involves learning a probability distribution from a given set of data and then using it to generate new data that resembles the original set. In this section, Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) techniques are described in detail. GANs and VAEs both have the common objective of learning the underlying probability distribution of the data, but they differ in their approaches to achieve this.

GANs were introduced by Goodfellow et al. in 2014 [4] and consist of two neural networks, a generator and a discriminator, that play a two-player minimax game. The generator network takes as input a random noise vector and tries to generate data that resembles the original data, while the discriminator network tries to distinguish between the generated and real data. The generator network is trained to fool the discriminator network, while the discriminator network is trained to correctly classify the data. Through this process of competition, the generator network learns to generate data that is indistinguishable from real data.

VAEs, on the other hand, were introduced by Kingma and Welling in 2013 [5] and are based on the concept of encoding and decoding data. VAEs consist of an encoder network that maps the input data to a latent space representation and a decoder network that maps the latent space representation back to the input data. The objective of VAEs is to learn the probability distribution of the latent space representation and use it to generate new data. VAEs use the reparameterization trick to sample from the learned distribution, which allows them to backpropagate gradients through the sampling operation during training.

Both GANs and VAEs have their advantages and disadvantages. GANs have been shown to produce high-quality samples that are visually appealing, but they can suffer from mode collapse, where the generator network learns to produce only a limited set of samples. VAEs, on the other hand, have been shown to be more stable and can capture a wider range of the data distribution, but they may produce blurry samples.

Several studies have compared the performance of GANs and VAEs on different datasets and have shown that the choice of model depends on the application and the dataset. Some studies have found that GANs perform better on image datasets with complex structures, while VAEs perform better on datasets with smooth and simple structures [6]. Other studies have found that combining the strengths of both GANs and VAEs can lead to better performance [7].

In this project, the following parameters are explored - the size of the networks used, the time taken to train, and the quality of the generated samples. By comparing the performance of GANs and VAEs on this dataset, we hope to gain insights into their strengths and weaknesses and identify the best model for this particular application.

2.1. Generative Adversarial Networks (GAN)

The architecture of a GAN typically consists of multiple layers of neural networks, with the generator and discriminator networks being connected in a loop. The generator network takes a random vector as input and transforms it through a series of layers into a data sample that is meant to resemble a sample from the training data. The discriminator network takes a sample as input and produces a binary output indicating whether the input sample is real or fake.

The generator and discriminator networks are trained using an adversarial loss function. This loss function measures how well the generator is able to fool the discriminator and how well the discriminator is able to distinguish between real and fake samples. During training, the generator updates its parameters to improve its ability to produce realistic samples, while the discriminator updates its parameters to improve its ability to correctly identify real and fake samples.

There are several variations of the basic GAN architecture, including Conditional GANs, Wasserstein GANs (WGANs), and StyleGANs [8]. These variations incorporate additional features and modifications to improve the stability and quality of generated samples.

Minmax game objective of GAN:

$$\min_G \max_D E_{x\sim p_{data}(x)}[\log D(x)]+E_{z\sim p_z(z)}[1-\log D(G(z))] \tag{1}$$

2.2. Variational Autoencoder (VAE)

VAEs use an encoder-decoder architecture, similar to GANs. The encoder takes an input data point x and maps it to a set of latent variables z. The decoder then takes these variables and maps them back to a reconstructed version of the original input data. The goal is to learn the parameters of the encoder and decoder such that the reconstruction error is minimized.

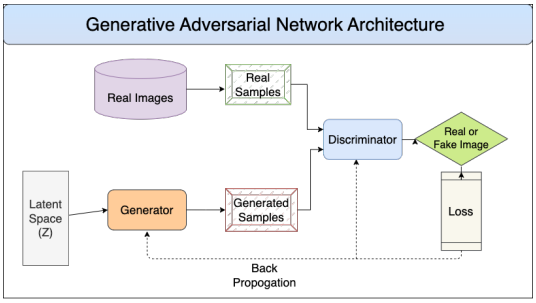

Figure 1. GAN Architecture

The key difference between VAEs and traditional autoencoders is that the latent variables are not directly used to encode the input data. Instead, the encoder maps the input data to a distribution over the latent variables, typically a Gaussian distribution with a mean and a standard deviation. The decoder then samples from this distribution to generate a set of latent variables, which are then used to reconstruct the input data. This introduces a stochastic element to the encoding process, allowing the model to capture more complex and diverse distributions of the data.

During training, the VAE minimizes a loss function that consists of two terms: a reconstruction loss, which measures how well the reconstructed data matches the original input, and a regularization term that encourages the distribution of the latent variables to be close to a standard Gaussian distribution. This regularization term is known as the KL divergence, and it ensures that the model does not overfit the data and generates diverse samples.

Kullback-Leibler Divergence(KL-Divergence)

$$KL(P||Q) = \sum_x P(x)\log(\frac{P(x)}{Q(x)}) \tag{2}$$

Reparameterization trick:

To generate latent variable z so that

$$z \sim q_{\mu,\sigma}(z) = \mathcal{N}(\mu, \sigma^2) \tag{3}$$

we sample,

$$\epsilon \sim \mathcal{N}(0, 1) \tag{4}$$

which gives us z,

$$z = \mu + \epsilon \cdot \sigma \tag{5}$$

2.3. Architectural Differences Between GAN and VAE

The key difference between GANs and VAEs is in how they define and optimize their objective functions. GANs

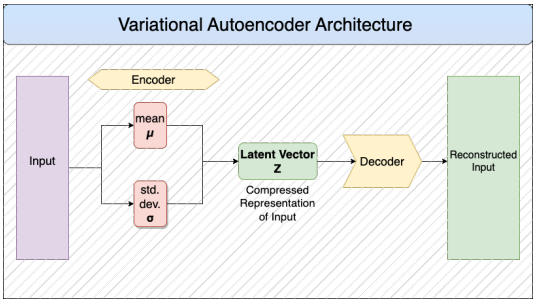

Figure 2. VAE Architecture

use a discriminative model and a generative model that compete against each other in a two-player minimax game. In contrast, VAEs use a probabilistic encoder and decoder, and optimize the evidence lower bound (ELBO) to learn the generative model. GANs are focused on learning to generate realistic samples through adversarial training, while VAEs are focused on learning a compressed representation of the input data that can be used for generating new samples.

3. Methodology

3.1. Generative Adversarial Networks (GAN)

This section describes the implementation of a Deep Convolutional Generative Adversarial Network (DCGAN) that was used for this project. The generator and discriminator have separate loss functions, which are optimized through a training loop.

A DCGAN class which is a keras.Model class is implemented, which overrides the train_step method. The discriminator and generator models are passed to the DCGAN class, along with the size of latent space (latent dimensions) of the random noise. The compile method is used to specify the optimization algorithm and the loss function for the discriminator and generator models. Adam optimizer, with a learning rate of 0.0002, and the Binary Cross Entropy loss functions are used for this implementation. In the train_step method, the DCGAN takes real images as input and uses them to calculate the discriminator loss. Then, the DCGAN generates fake images from random noise and uses them to calculate the generator loss. The gradients are calculated for both the discriminator and generator loss and used to update the discriminator and generator weights, respectively. The mean loss of each is calculated and returned.

The architecture of the generator is as follows:

- Input a random noise vector of size latent dimensions.
- Dense layer reshapes the noise to 7x7x256.
- Batch normalization and ReLU activation are applied
- Reshape to 3D tensor with a shape of 7x7x256.
- Two Conv2DTranspose with 128 and 64 filters upsample the tensor to 14x14x64 and then to 28x28x1.
- Batch normalization and ReLU activation are applied
- The final layer is a Conv2D with a kernel size of 5x5, outputting a single channel image with a tanh activation function.

The architecture of the discriminator is as follows:

- Input an image with shape (28, 28, 1)
- Conv2D layer with 64 filters, a kernel size of 5x5, and a stride of 2x2
- Batch normalization and a leaky ReLU activation, alpha=0.2
- A similar layer is applied with 128 filters, a kernel size of 5x5, and a stride of 2x2
- The output is flattened and a dropout layer is applied with a rate of 0.3
- A fully connected layer with a sigmoid activation function outputs a single value that represents the probability of the input being real (as opposed to fake)

3.2. Variational Autoencoder (VAE)

This section describes the implementation of a convolutional Variational AutoEncoder (VAE). The VAE consists of an encoder and a decoder. The encoder takes an input image of shape (28, 28, 1) and maps it to a latent space of two dimensions. The decoder then maps the latent representation back to an image.

The architecture of the encoder is as follows:

- A Conv2D layer with 32 filters, a kernel size of 3, a stride of 2, same padding, and ReLU activation
- A second Conv2D layer with 64 filters, a kernel size of 3, a stride of 2, same padding, and ReLU activation
- A Flatten layer to flatten the output of the second Conv2D layer
- A Dense layer with 16 units and ReLU activation
- Two Dense layers, one with latent_dim (2) units that produces the mean of the latent space, and another with latent_dim units that produces the logarithm of the variance of the latent space

The architecture of the decoder is as follows:

- A Dense layer with 7x7x64 units and ReLU activation
- A Reshape layer to reshape the output of the first Dense layer to a 3D tensor with shape (7, 7, 64)
- A Conv2DTranspose layer with 64 filters, a kernel size of 3, a stride of 2, same padding, and ReLU activation
- A second Conv2DTranspose layer with 32 filters, a kernel size of 3, a stride of 2, same padding, and ReLU activation
- A final Conv2DTranspose layer with a single filter, a kernel size of 3, same padding, and sigmoid activation that maps the output of the second Conv2DTranspose layer to an image of the same shape as the input image.

The VAE is trained using the Adam optimizer with a custom train_step method. The training loss is the sum of the reconstruction loss and the Kullback-Leibler divergence (KL divergence) between the learned latent distribution and a standard normal distribution. The reconstruction loss measures the pixel-wise difference between the input image and the reconstructed image. The KL divergence encourages the learned latent distribution to be similar to a standard normal distribution, which helps to regularize the latent space and improves the quality of the generated images. The train_step method computes the gradients of the loss with respect to the trainable weights and applies them to update the weights. During training, the VAE tracks the total loss, the reconstruction loss, and the KL loss.

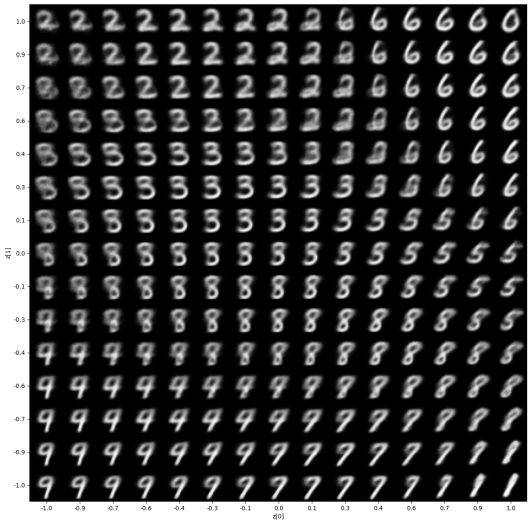

Figure 4. VAE Results on Digits MNIST Dataset Trained for 15 minutes on Google Colab GPUs. This output represents the latent space which is a normal distribution

4. Results

4.1. Digits MNIST

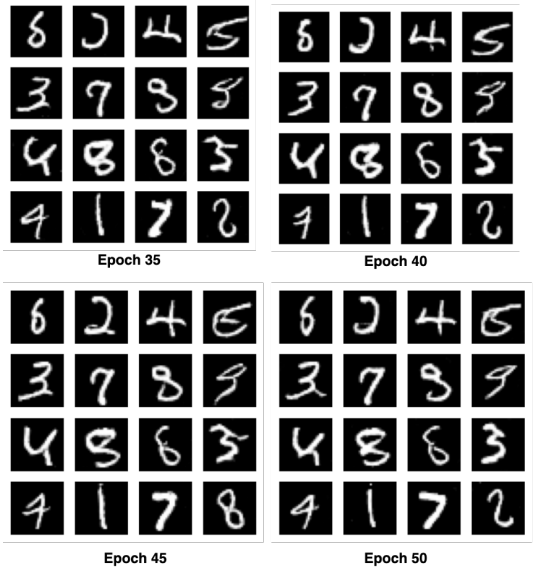

Figure 3. GAN Results on Digit MNIST Dataset Trained for 35 minutes on Google Colab GPUs

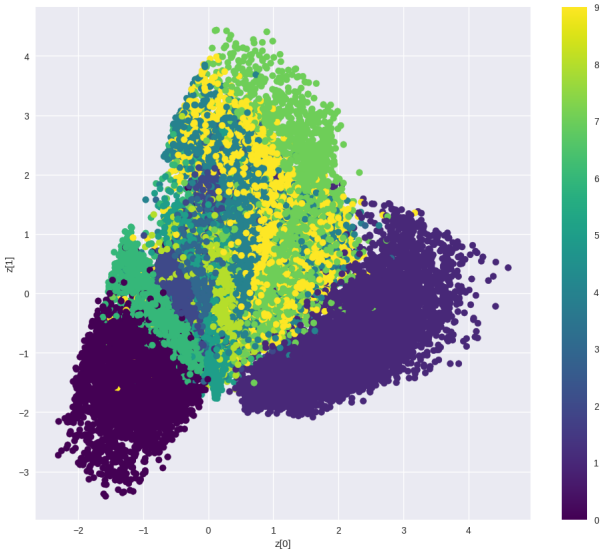

Figure 5. Latent Space Clusters for Different Digit Classes for the VAE Model

4.2. Fashion MNIST

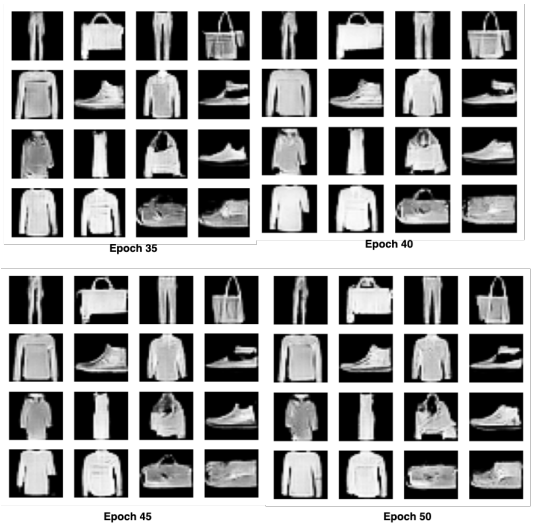

Figure 6. GAN Results on Fashion MNIST Dataset Trained for 40 minutes on Google Colab GPUs

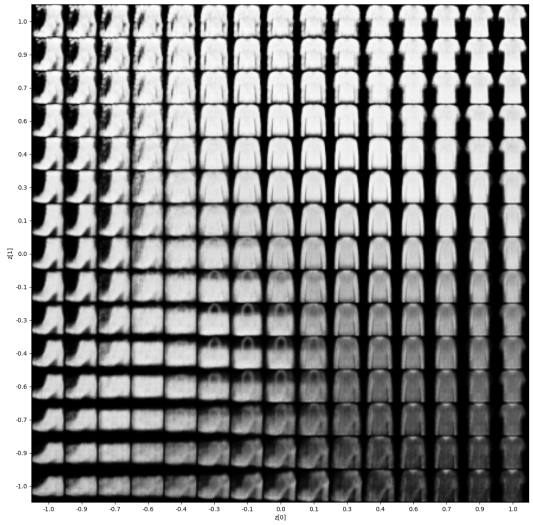

Figure 7. VAE Results on Fashion MNIST Dataset Trained for 18 minutes on Google Colab GPUs. This output represents the latent space which is a normal distribution

5. Conclusion

- In terms of the number of trainable parameters, GANs typically have much higher parameter counts than VAEs. This is because GANs have two separate networks - the generator and the discriminator - that need to be trained simultaneously. The generator network must learn to produce samples that are indistinguishable from the real data, while the discriminator network must learn to correctly identify whether a sample is real or fake. This requires a large number of parameters and often results in larger model sizes. On the other hand, VAEs only have one network to train, which typically results in fewer parameters and smaller model sizes.

- The training time for GANs is generally longer than for VAEs due to the fact that GANs require multiple forward and backward passes through both the generator and discriminator networks at each training iteration. This can result in slower training times, especially for larger models with more parameters. In contrast, VAEs only require one forward and backward pass through the network per iteration, which can result in faster training times overall.

- Previous testing and observations have indicated that GANs often produce more visually pleasing output compared to VAEs. This is because GANs train to recreate the input data and can often produce highly realistic samples that closely resemble the real data. However, it is important to note that this comes at the cost of potentially sacrificing diversity in the output. This is because GANs only generate samples that are similar to the training data, whereas VAEs generate samples that are not limited to the training set and can produce a wider range of diverse outputs.

- While GANs can produce highly realistic output, they often lack diversity and can generate inconsistent results. This is because GANs generate samples by interpolating between points in the latent space, which can result in limited diversity in the output. In contrast, VAEs capture the underlying latent distribution of the data and are able to generate more diverse output by sampling from this distribution. This often results in a wider range of diverse and novel outputs compared to GANs.

References

[1] Ali Borji , aliborji@gmail.com; (n.d.). Pros and cons of GAN evaluation measures: New developments.

[2] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y. (2014, June 10). Generative Adversarial Networks. arXiv.Org. https://arxiv.org/abs/1406.2661

[3] Gur, S., Benaim, S., Wolf, L. (2020, June 22). Hierarchical patch VAE-GAN: Generating diverse videos from a single sample. arXiv.Org. https://arxiv.org/abs/2006.12226

[4] Jabbar, A., Li, X., Omar, B. (2021). A survey on generative adversarial networks: Variants, applications, and training. ACM Computing Surveys, 54(8), 1–49. https://doi.org/10.1145/3463475

[5] Kingma, D. P., Welling, M. (2013, December 20). Auto-Encoding variational bayes. arXiv.Org. https://arxiv.org/abs/1312.6114

[6] Kingma, D. P., Welling, M. (2019, June 6). An introduction to variational autoencoders. arXiv.Org. https://arxiv.org/abs/1906.02691

[7] Maynard-Reid, M. (2021, September 13). Intro to generative adversarial networks (gans). PyImage-Search. https://pyimagesearch.com/2021/09/13/intro-to-generative-adversarial-networks-gans/

[8] Team, K. (n.d.). Keras documentation: Variational AutoEncoder. Retrieved April 20, 2023, from https://keras.io/examples/generative/vae/