A Study of Generative Adversarial Networks & Variational Autoencoders for MNIST Image Generation

Sri Maruti Keerthana Ponnuru[1]
University of Florida
UFID 3394 5082
ponnurus@ufl.edu

Abstract

Generating high-quality images is a pertinent topic in computer vision for the longest time. Recently, Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) have gained popularity as two effective methods for generating images. In this study, I conducted a comprehensive analysis of the performance of the implementation of GAN and VAE in generating images of handwritten digits from the MNIST dataset. We compare the quality of images produced by both models using various metrics. I also investigated the impact of different architectural choices, hyperparameters, and loss functions on the models' performance. For future scope I have made an attempt at training a GAN model on face image dataset. Through this study it is understood that the choice of loss function also plays a significant role in the quality of generated images for both models. This work provides valuable insights into the strengths and limitations of GANs and VAEs in MNIST image generation, which can guide future research to improve image generation using these models.

Keywords: Computer Vision, Generative Adversarial Networks, Variational Autoencoders, MNIST, loss function.

1. Introduction

For the purpose of image generation, Generative models have been sought after and one possible reason could be the difficulty in finding data that has been supervised. A generative model aims to analyze and understand a set of sample data used for training and then acquire knowledge of the probability distribution that produced them[1]. In this paper, we implemented the GAN and VAE models on the MNIST dataset. The dataset consists of grayscale images of size 28x28, which contain handwritten digits. The aim is that we need a distribution that is fit to this dataset and then must be able to sample from this distribution, such that every IND sample should look like the desired corresponding digit. Generative Adversarial Networks (GANs) create such examples on the basis of the estimated probability distribution [2]. The GAN architecture comprises a generator and discriminator, which are trained in an adversarial manner. The generator is a neural network it first samples from the latent space and produces an image based on that sample and weights are set such that it generates an image, it is also called the deconvolutional network. The discriminator takes an image and determines the probability of it being real or not, which basically is an adversial network. Variational Auto Encoders (VAEs) capture the probability distribution of the input data to generate new samples by randomly sampling from that learned distribution[3]. It uses a random variable to generate new samples by encoding data into a low-dimensional space. This allows the model to balance between producing samples that are similar to the training set and creating new and unique examples.

2. Mathematical Notation

The mathematical expression for both GAN and VAE are crucial as we implement the loss function based on that.

2.1. GAN

The purpose of the generator is to generate images that should be good enough so that the discriminator does not detect it. On the other hand the discriminator intends to get better at distinguishing if the given image is a result of the generator or is it an original image.

Initial objective is to learn the Generator's distribution Pg on data y. An input noise variable is defined on probability distribution of latent variables Pz(z) [4]. The generative network mapping can be represented as G(z, θ_g). The Loss function is given as:

$$V(D, G) = \min_G \max_D [\log D(x)) + \log(1 - D(G(z)))]$$

The Discrimantor network mapping is D(x, θ_d). Here D(x) is the probability that x has come from original data. d is trained such that the probability of assigning the correct labels is maximum and G is trained to maximize log likelihood log(1-D(G(z))).

2.2. VAE

The logic behind VAE is a two step process for generating data , consider latent variables z with distribution P(z), using a distribution p(x|z) data is generated [5]. Evaluation of p(z |x) is a tougher task since p(x) is not tractable.

$$q(z|\ x) = N\ (z \mid \mu\ , \sigma^2)$$

here x and y are mean and standard deviation. The encoder in VAE is responsible for approximating the posterior distribution Q(Z|X) given the input data X. Once the posterior distribution is optimally approximated, it is sampled from and the samples are fed into the recognition model. L(

$$p(x|\ z) = \text{Bernoulli}\ (x|\theta)$$

here theta is an argument to the decoder. The recognition model aims to maximize the likelihood log(P(X)). This likelihood is computed as the sum of the KL Divergence between the priors P(Z|X) and Q(Z|X) and the evidence lower bound.

$$L(\theta, \Phi; x) = E[\log P(x|z;\theta)] - KL[Q(z|x; \Phi) \mid\mid P(z; \theta)]$$

Since the KL Divergence value is always greater than 0, the likelihood is guaranteed to be greater than or equal to the lower bound.

3. Literature Survey

A great deal of research has been conducted on the Generative models in the recent times. These papers have provided a path way for many upcoming variations on the models.

One such prominent work on GAN is by Ian Goodfellow and his team. They introduced the concept of Generative Adversarial Networks (GANs) in 2014 [1]. The initial architecture comprised a generator and a discriminator network, both being feedforward neural networks. The generator network generated samples by taking random noise as input, while the discriminator network classified the samples as real or fake. The authors demonstrated that GANs could create high-quality images that resembled those from the training data and could be applied to other tasks, including style transfer and image super-resolution. Since the publication of the original GAN paper, researchers have proposed several modifications and extensions to the basic architecture.

Also, another eminent research work done on VAE is Auto-Encoding Variational Bayes (AEVB) is a powerful method for variational inference in probabilistic models which was introduced by Kingma and Welling in 2013 [2], AEVBprovides a way to estimate the intractable posterior distribution over latent variables in a Bayesian model by training an encoder and decoder neural network to map inputs to latent variables and back again. This approach allows for efficient inference and generative modeling, and has found numerous applications in computer vision, natural language processing, and other fields. In their original paper, Kingma and Welling introduced a specific instantiation of the AEVB algorithm using Gaussian latent variables and a neural network architecture known as the ”reparameterization trick.” They demonstrated the effectiveness of their method on a variety of tasks, including image generation, data compression, and semi-supervised learning. Since its introduction, AEVB has become a widely used tool in the machine learning community, and numerous extensions and variations have been proposed. One notable extension is the use of non-Gaussian latent variables, such as those that follow a mixture of Gaussians or a categorical distribution.

4. Methodology & Implementation

The detailed implementation of GAN and VAE on the MNIST dataset has been explained in this section

4.1. MNIST

MNIST is well-known database ofhandwritten digits, each image can be taken as an array of numbers mentioning the darkness of each pixel, the datapoints are embedded in a 784 dimensional space [6].

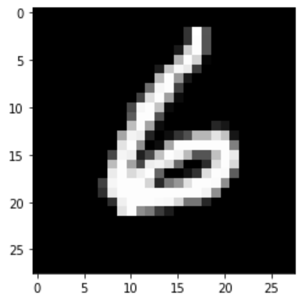

Figure 1. Datapoint visualisation

The dataset is made up of grayscale images of handwritten digits with a resolution of 28x28 pixels, and is commonly used for image classification tasks in machine learning and computer vision research.

It contains 60,000 training images and 10,000 test images, each labeled with a digit from 0 to 9. Due to its high dimensionality, it is frequently utilized for dimensionality reduction techniques such as PCA. Initially, I have explored

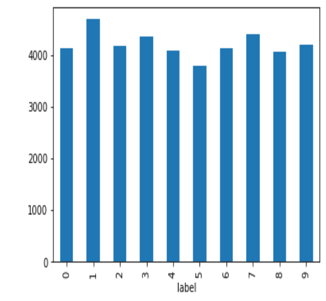

Figure 2. Class frequency graph

the data since it is crucial to comprehend the dataset better. This involved examining the dataset dimensions, class frequencies, and visualizing individual data points.

4.2. Generative Adversarial Networks

As discussed in the GAN, we need to implement both the Generator and the Discrimator part adversially [7]. Implementation was done using PyTorch, loaded the MNIST dataset from the torchVision library. After visualizing the data, I created the discriminator with linear layers including one input layer, one hidden layer, and one output layer. All layers in the network were linear and used LeakyReLU as the activation function with a negative slope of 0.2 in the input layer and hidden layer, and sigmoid activation function was used in the output layer.

The input layer had 784 nodes since it was equal to the size of the input (28x28) image, the hidden layer had 256 nodes, and the output layer had one node to output the probability of the input coming from the data distribution. Next, I created a generator with three linear layers, including one input layer, one hidden layer, and one output layer. ReLU was used as the activation function in all layers except for the output layer, where TanH was used. Since the pixel values were between -1 and +1, TanH was a good activation function in the output layer that squashed the values to stay between -1 and +1.

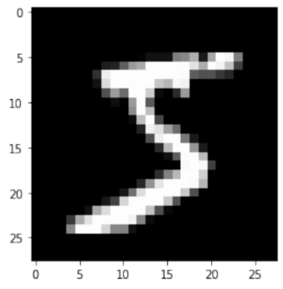

Figure 3. GAN generated output

In the initial execution of the model everything was hazy similar to snowflakes pattern. The loss used for this model was Binary Cross Entropy Loss, and each neural network had a separate ADAM optimizer with a learning rate of 0.002 initially and after trial and error changed it to 0.0002 to prevent the gradients of one network from affecting the other. The number of epochs was initially 100 and then raised to 200. During each epoch, the discriminator was trained for k steps, and then the generator was trained for 1 step to keep the discriminator near optimum and improve the generator. The generator was improved until the output of the discriminator came close to 0.5, which was the best-case scenario. The images generated by the generator during training were saved, and the performance of the model was analyzed.

4.3. Variational Auto Encoder

In this approach I have used PyTorch to implement the VAE architecture with four fully connected layers [8]. The first layer takes the input data (a flattened 28x28 image) and maps it to a hidden layer with 400 nodes. The second layer consists of two branches that output the mean and log variance of the latent variables. The third layer maps the latent variables to another hidden layer with 400 nodes. Finally, the fourth layer maps the hidden layer back to the original input data dimensions. The VAE has three methods. One method to calculate the mean and log variance of the latent variables from the input data. Second method maps the latent variables back to the original input data. The third method to generate a random sample from the latent variable distribution by reparameterizing the mean and log variance.

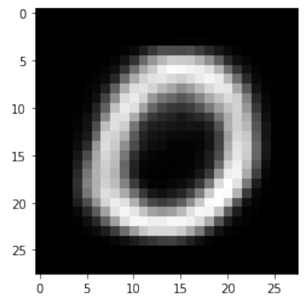

Figure 4. VAE generated output

Finally another method to combine the above three methods and generate output of the VAE from an input data sample. This method also returns the mean and log variance of the latent variables. The encoder network was composed of dense layers and ReLU was used as the activation function, except for the last layer where the latent variable distribution was generated using the sigmoid activation function. To enable the decoder network to perform gradient ascent properly, we applied a reparameterization technique to the output of the encoder network before using it as an input to the decoder network. The decoder network was also made up of dense layers. During the training process, I used the Adam optimizer with its default learning rate of 1e-3 for optimization. The loss function was a combination of KL Divergence and Binary Cross Entropy loss, which acted as the evidence lower bound.

4.4. GAN Intution- Image dataset

The motivation behind this was to establish an understanding of the differences in applying GAN on MNIST dataset vs the image dataset. MNIST dataset is made up of basic grayscale images of handwritten digits that are 28x28 pixels in size. On the other hand, face image datasets contain more complex images with different colors, textures, and lighting conditions[9]. As a result, training a GAN on face image datasets requires a more complex model architecture with more layers than training a GAN on the MNIST dataset. Generating realistic face images is a challenging task because the model must capture subtle features and variations such as facial expressions, hair styles, and skin tones.

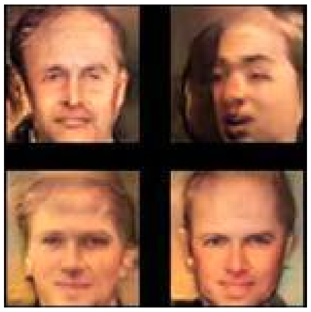

Figure 5. GAN on images

This is a more complex task than generating simple handwritten digits. For experimentation puprose I have used about 1000 images from the CelebA dataset which is a popular image dataset and performed some tweaks on an existing algorithm for better understanding that contains hundreds of thousands of images while the MNIST dataset contains only 60,000 images. As a result, training a GAN on face image datasets can be time-consuming and working on VAE in face image dataset would also need some thought and time which would be my future scope

5. Results

In this section, an attempt to elaborate on the findings is made to understand performance of GAN vs VAE in terms of network size, train time, error handling.

To generate the images the model architecture used for GAN is given below where bias is True. Discriminator(
(layer1): Linear(in_features=784, out_features=128)
(layer2): Linear(in_features=128, out_features=64)
(layer3): Linear(in_features=64, out_features=32)
(layer4): Linear(in_features=32, out_features=1)
(dropout): Dropout(p=0.3, in_place=False)
)

Generator(
(layer1): Linear(in_features=100, out_features=32)
(layer2): Linear(in_features=32, out_features=64)
(layer3): Linear(in_features=64, out_features=128)
(layer4): Linear(in_features=128, out_features=784)
(dropout): Dropout(p=0.3, inplace=False)
)

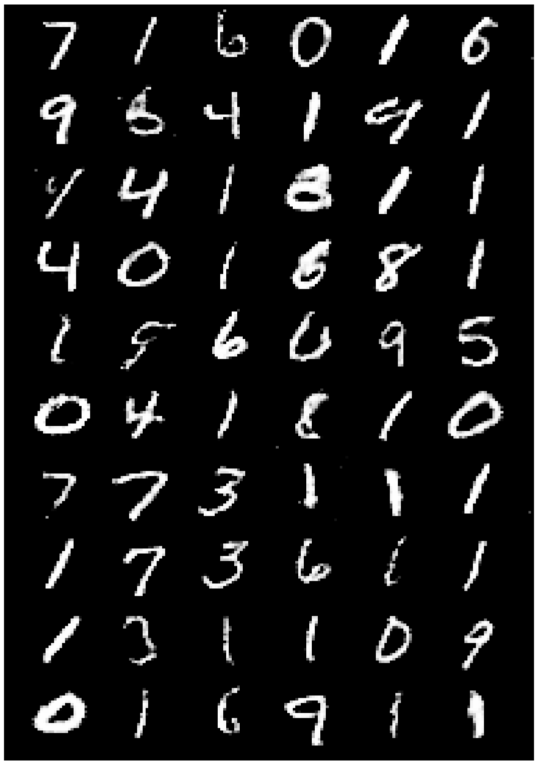

Figure 6. GAN output

The architecture of VAE used is as given below where bias is True VAE(
(layer1): Linear(in_features=784, out_features=512)
(layer2): Linear(in_features=512, out_features=256)
(layer31): Linear(in_features=256, out_features=2)
(layer32): Linear(in_features=256, out_features=2)
(layer4): Linear(in_features=2, out_features=256)
(layer5): Linear(in_features=256, out_features=512)
(layer6): Linear(in_features=512, out_features=784))

The VAE model has a total of 7 linear layers with 784, 512, 256, 2, 2, 256, and 784 output features, respectively. On the other hand, the GAN model has a total of 5 linear layers with 784, 128, 64, 32, and 1 output features, respectively, for the discriminator network.

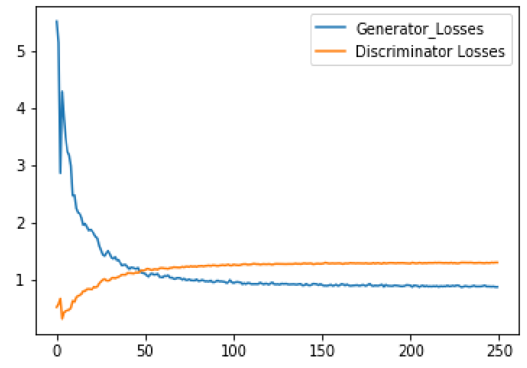

Figure 7. GAN loss

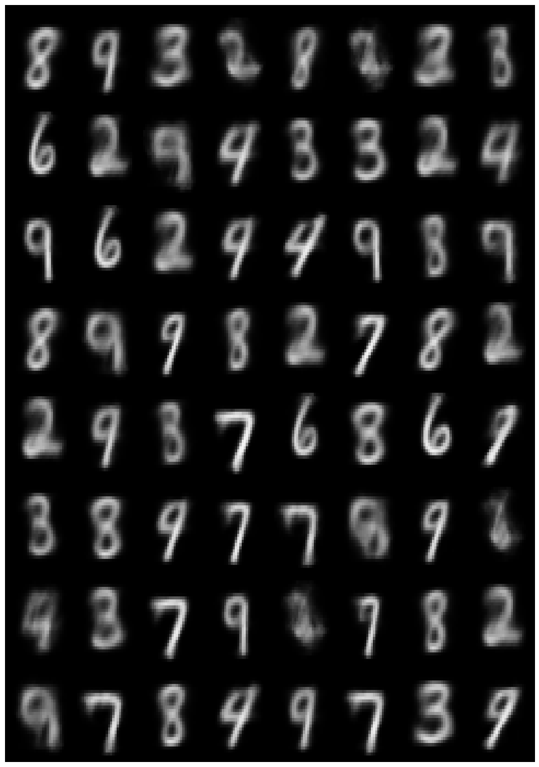

Figure 8. VAE output

The generator network has 4 linear layers with 100, 32, 64, 128, and 784 output features, respectively. On the other hand, the GAN model has a total of 5 linear layers with 784, 128, 64, 32, and 1 output features, respectively, for the discriminator network. The generator network has 4 linear layers with 100, 32, 64, 128, and 784 output features, respectively. The network size if relatively larger with more parameters in the VAE model over GAN. The training time for the GAN model is affected by depth of the network and size on input hence a comparison of these two would not be a great way to evaluate the working but on my device both the algorithms were able to train well and perfomed

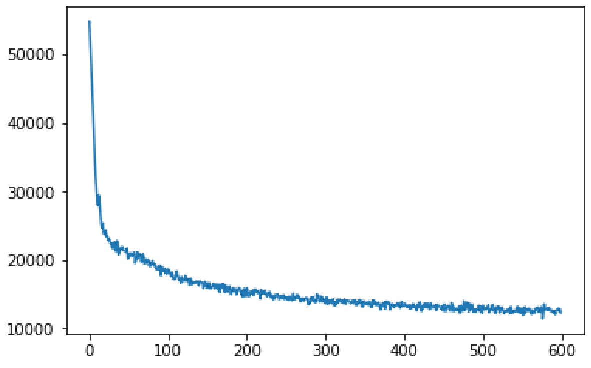

Figure 9. VAE loss

Table 1. GAN vs VAE Comparison.

GAN	VAE
5 LAYERS	7 LAYERS
ADVERSARIAL TRAINING	MAXIMUM LIKELIHOOD ESTIMATION
MIN MAX GAME PRINCIPLE	MAXIMISE ELBO
	MINIMIZE RECONSTRUCTION LOSS & KL DIVERGENCE

well. The Table 1, gives a bried idea about the differences between GAN and VAE.

6. Conclusion

Based on the above work implemented we got a good understanding of GAN and VAE. Both VAE and GAN have a shared component, which is the probability distribution function p(x|z), where z is the latent variable and x is the visible variable. However, what sets them apart is that VAE has a probability distribution function p(z|x), while GAN has a discriminator. Both VAE and GAN use neural networks to implement these components, but they differ in their approach to modeling the probability distribution function and objective functions.

There have been some challenges in this project such as some of the images are not legible as a human written handwriting can cause errors, increasing the number of epochs would increase the quality of the output which we might expect. While training GAN huge computational cost is a set back as it requires training both Generator and Discriminator. Whereas the challenges with VAE is that the outputs are blurred, in cases of medcial image generation this poses a problem. Hyperparameter tuning is also a majot cahllenge as we need to aim for optimization.The future scope includes working on building these models on larger more complicated datasets such as image dataset.

To sum up, we successfully trained and demonstrated the MNIST dataset using GAN and VAE models, producing satisfactory generated images. While GAN generated sharper images, it is more challenging to train than VAE, requiring more computational power. Both GANs and VAEs are powerful generative models that have shown significant success in various applications, such as generating images, text, and music.

References

[1] Goodfellow, Ian, et al. ”Generative adversarial networks.” Communications of the ACM 63.11 (2020): 139-144.

[2] Creswell, Antonia, et al. ”Generative adversarial networks: An overview.” IEEE signal processing magazine 35.1 (2018): 53-65.

[3] Kingma, Diederik P., and Max Welling. ”An introduction to variational autoencoders.” Foundations and Trends® in Machine Learning 12.4 (2019): 307-392

[4] Wang, Yang. ”A mathematical introduction to generative adversarial nets (GAN).” arXiv preprint arXiv:2009.00169 (2020).

[5] Tutorial on variational autoencoders - Doersch, Carl. ”Tutorial on variational autoencoders.” arXiv preprint arXiv:1606.05908 (2016).

[6] Baldominos, Alejandro, Yago Saez, and Pedro Isasi. ”A survey of handwritten character recognition with mnist and emnist.” Applied Sciences 9.15 (2019): 3169.

[7] Metz, Luke, et al. ”Unrolled generative adversarial networks.” arXiv preprint arXiv:1611.02163 (2016).

[8] Doersch, Carl. ”Tutorial on variational autoencoders.” arXiv preprint arXiv:1606.05908 (2016).

[9] Alotaibi, Aziz. ”Deep generative adversarial networks for image-to-image translation: A review.” Symmetry 12.10 (2020): 1705.