Comparative Analysis Between Generative Adversarial Networks and Variational Autoencoders

Rohan H. Wanare
University of Florida
Gainesville, FL 32608
rohanwanare@ufl.edu

Abstract

The project report deals with the comparative analysis between the Generative Adversarial Networks (GAN) and Variational Autoencoders (VAE). These networks are the latest and greatest, belonging to the family of generative models. A comparison between these models makes sense since they have a some similarities, that is to generate images. Even though both these models have their own roles in the Machine Learning regime, a comparison between them provides a good understanding of their roles and their use cases.

1 Introduction

1.1 Generative Adversarial Networks (GAN)

Generative Adversarial Networks have been a huge success since their introduction in 2014 by Ian J. Goodfellow and co. authors in their paper, **Generative Adversarial Networks**.

A GAN consists of two networks called the generator and the discriminator. The generative network generates candidates while the discriminative network evaluates them. This is a contest in terms of data distribution [1]. The generator's job is to learn the mapping from an initial latent space to the existing data distribution of the sample data and the discriminator tries to determine if the data is from the sample or the generator has created the data. So it can be said that the goal of the generator is to fool the discriminator into classifying falsely created data as true data while the discriminator's goal is to classify the true and the fake data correctly. The true data is the training dataset that is passed on as the input to both the discriminator and generator to learn. The figure below should give a clearer view of the roles of the generator and the discriminator in the network.

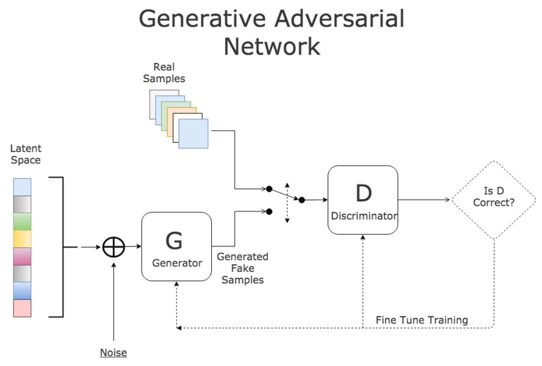


Figure 1: Generative Adversarial Network Model [2]

Application of GANs include but are not limited to,

- Creating photos of imaginary fashion models, with no need to hire a model, photographer or makeup artist, or pay for a studio and transportation
- Up-scaling low resolution 2D images of old games by creating 1080p or higher resolutions
- Deepfake [3], in which a person in an existing image or video is replaced with someone else's likeness to fake content. Deepfake can also create a new person altogether as well.

1.2 Variational Autoencoders (VAE)

Autoencoder is a type of a generative model whose aim is to learn a representation (encoding) for a set of data, typically by dimensionality reduction. An autoencoder consists of an encoder and a decoder. The job of the encoder is to compress the data using dimensionality reduction, that is, finding some sort of structure that exists in the data (i.e. correlations between input features) and learning them. The aim of the decoder is to recreate the the original input in its entirety from the compressed data representation received from the encoder. So in essence, the encoder network outputs a single value for each encoding dimension and the decoder network then subsequently takes these values and attempts to recreate the original input.

On the contrary to autoencoders, the VAE provides a more probabilistic manner for describing an observation in latent space. Therefore, instead of outputting just a single value to describe each latent state as in the vanilla encoder, it builds up a probability distribution for each latent attribute [4].

The motivation behind VAE is simple, if there exists some latent variable Z that creates a data X, we can only see X but it is important to infer the characteristics of Z since it expresses X. Hence, using variational inference to find X is much more helpful. The figure below provides more insight on the same.

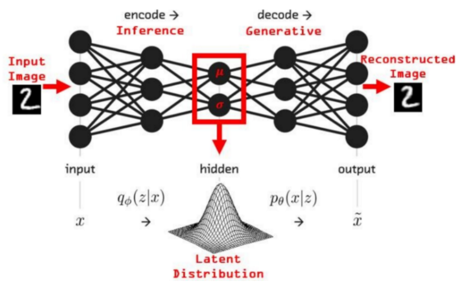


Figure 2: Variational Autoencoder [5]

Application of VAEs include from generating fake human faces, to producing purely synthetic music.

1.3 MNIST dataset

The MNIST dataset is a dataset consisting of 70,000 handwritten digits split into a training set of 60,000 and a test set of 10,000 handwritten digit images. These are grayscale 28x28 pixel images. This dataset can by default be found in the datasets library of PyTorch and Tensorflow and can be imported just by calling torchvision.datasets.MNIST() in PyTorch.

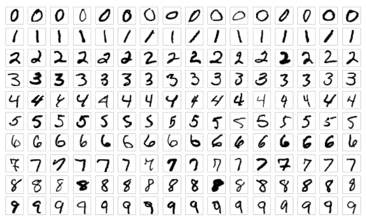


Figure 3: MNIST Dataset

2 Evolution of our Generative Adversarial Network

2.1 Phase 2

A baseline GAN model was created to see how things run in practice. The image below shows the architecture of the baseline model created.

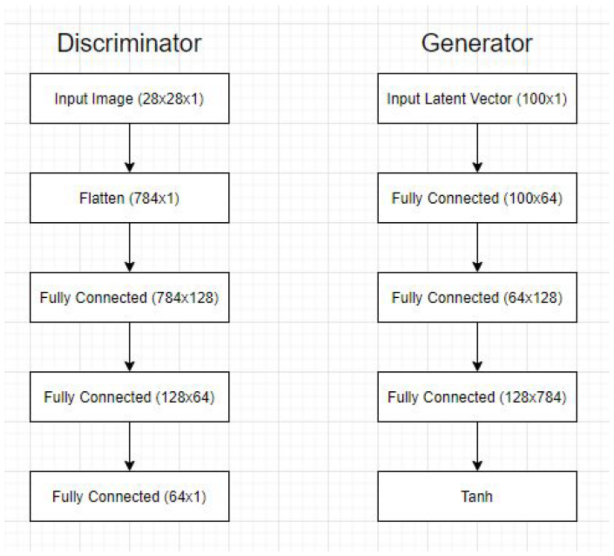


Figure 4: Baseline GAN Model

The discriminator receives input images of 28x28x1. It flattens the image into a vector and passes it through 2 fully connected hidden layers before reaching the output layer which has just 1 neuron which classifies the input image as real or fake. The output label for real images is 1, while the images generated by the generator (aka fake images) has output label 0.

The generator starts off with an initial randomly generated latent vector of uniform distribution. It passes the 100x1 vector through 2 hidden layers and a tanh regularization function and finally ends up with 784x1 vector which given as an input to the discriminator to test whether the input is real or fake.

The main objective of the base model is to get the flow of the discriminator and the generator working. Once this part is accomplished and baseline results are received, tuning the hyperparameters can be started.

Results generated by the generator of the baseline model are shown below,

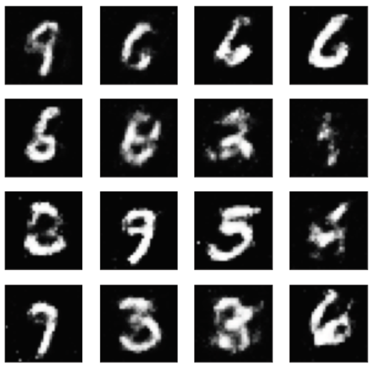


Figure 5: Recreated images by Baseline Generator

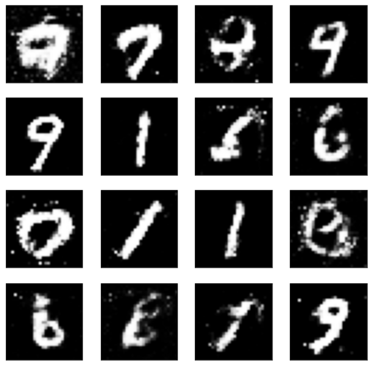


Figure 6: Newly generated fake images by Baseline Generator

Hyperparameters that need tuning to get better results are,

- Number of hidden layers of Generator and Discriminator
- Number of neurons in the hidden layers of the Generator and Discriminator
- Learning rate of Generator and Discriminator
- Number of epochs for training (currently, 50)

2.2 Phase 3

I increased the number of hidden layers from 2 to 3 and thus I also had to increase the number of neurons in each hidden layer (compare current model to previous model) and instantly got better results. I tried a bunch of learning rates like 0.002, 0.0005, 0.0002, 0.0001, 0.00005 and found 0.0002 had slightly better results. To top it all off, I increased the number of epochs from 50 to 100 to see if there's any change in the model and there were significant improvements.

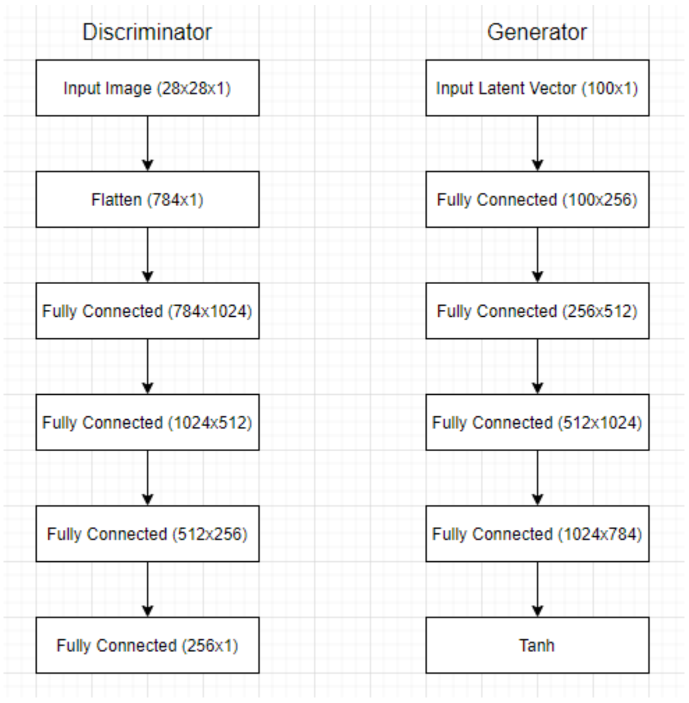

Discriminator
Input Image (28x28x1)
Flatten (784x1)
Fully Connected (784x1024)
Fully Connected (1024x512)
Fully Connected (512x256)
Fully Connected (256x1)
Generator
Input Latent Vector (100x1)
Fully Connected (100x256)
Fully Connected (256x512)
Fully Connected (512x1024)
Fully Connected (1024x784)
Tanh

Figure 7: Final GAN Model

Results generated by the generator of the baseline model are shown below,

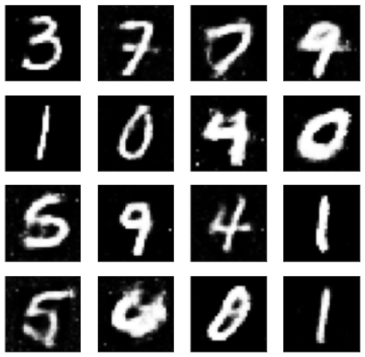


Figure 8: Recreated images by Final Generator

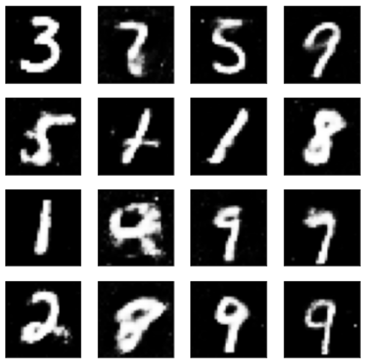


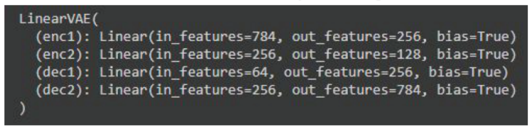
Figure 9: Newly generated fake images by Final Generator

We can see that the recreated images as well as the newly generated images by the generated are far better in the final GAN than in the baseline model.

3 Evolution of our Variational Autoencoder

3.1 Phase 2

We follow the same procedure we did for the GAN model and try to create a baseline model to see how the model runs in practice and in the later phase, try to tune the hyperparameters to squeeze out performance. The image below shows the baseline model.



```
LinearVAE(
  (enc1): Linear(in_features=784, out_features=256, bias=True)
  (enc2): Linear(in_features=256, out_features=128, bias=True)
  (dec1): Linear(in_features=64, out_features=256, bias=True)
  (dec2): Linear(in_features=256, out_features=784, bias=True)
)
```

Figure 10: Baseline VAE Model

Thus, rather than building an encoder which outputs a single value to describe each latent state attribute, we'll formulate our encoder to describe a probability distribution for each latent attribute.

We know that encoders in VAE output a mean and a variance of the distribution for each latent attribute instead of single values, the decoder can pick up randomly from the distribution to create a smooth continuous representation.

For the baseline VAE model I just put 1 hidden layer for each encoder and decoder as it can be seen in the figure to get things going and trained the model for 30 epochs.

Result generated by the VAE is given below. For every column in the figure, the upper image is the input digit while the lower image is the reconstructed digit.

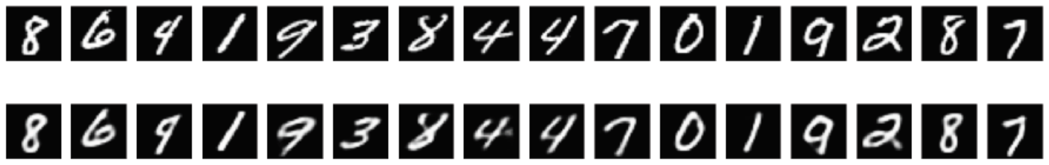


Figure 11: Recreated images by Baseline VAE

It reconstructs images to almost as good as before. I still will play with the hyperparameters (list given below) and try to improve performance.

Hyperparameters that need tuning to get better results are,

- Number of hidden layers of Encoder and Decoder
- Number of neurons in the hidden layers of the Encoder and Decoder
- Learning rate of Encoder and Decoder
- Number of epochs for training (currently, 30)

3.2 Phase 3

To improve the performance of VAE I tried changing the hidden nodes and layers in the network but there was almost no improvement and in some cases the network degraded. So the only change I felt worthwhile was changing the activation function from ReLU to Leaky ReLU and increasing the number of epochs from 30 to 100. These changes haven't shown a drastic increase in performance but there is a little improvement to an already great VAE for MNIST dataset. Below is the result of the final VAE model with all the improvements.

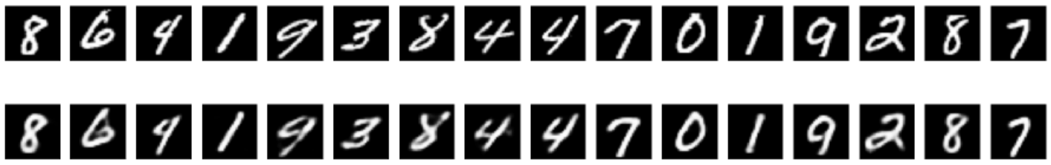


Figure 12: Recreated images by Final VAE

4 Evaluating GAN vs. VAE

From both sets of reconstructed images from the final models we can see that the VAE provides better reconstructions than the GAN consistently. But one thing that the GAN can do that the autoencoder can't is creating a new image from scratch.

I believe the VAE outperforms GAN at reconstructing images is because the encoder gives the decoder a distribution to feed from while the generator has to randomly try and find a good distribution to feed from and hence it doesn't perform as good.

Infact, the use cases of GANs are completely different than the VAEs and hence do things such as creating completely new images which the VAE cannot do.

In conclusion, Variational Autoencoders outshine Generative Adversarial Networks in recreating input images since its the only thing they're good at while the GANs have much more use cases and are primarily used in various other applications as seen in Section 1.1.

The code for both models can be found on GitHub by clicking **here**.

References

[1] https://en.wikipedia.org/wiki/Generative_adversarial_network

[2] https://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html

[3] https://en.wikipedia.org/wiki/Deepfake

[4] https://www.jeremyjordan.me/variational-autoencoders/

[5] https://medium.com/@realityenginesai/understanding-variational-autoencoders-and-their-applicat