Comparative analysis of the performance of a Generative Adversarial Network (GAN) vs a Variational Autoencoder (VAE) on the MNIST Dataset

Sachet Nayyar
Department of Computer and Information Science and Engineering
University of Florida
UFID: 6215-565
Gainesville, FL 32611
`sachet.nayyar@ufl.edu`

Abstract

Generative Adversarial Networks (GAN) and Variational Auto Encoders (VAE) are both machine learning models that utilize unsupervised learning and are generative in nature. A generative model is one which attempts to learn the distribution of data from training examples and subsequently attempts to generate data that is ideally indistinguishable from the training set. A GAN is composed of two primary components - a discriminator and a generator. The objective of the GAN is to generate new data examples which resemble the data it is training on as closely as possible. On the other hand a VAE consists of - an encoder and a decoder. The VAE approaches data generation a little differently, focusing on regularizing the encoding with the objective of ensuring that the latent space good properties to generate new data. The objective is to evaluate how both networks behave when working with images of handwritten digits which is the MNSIT data set.

1 Introduction

1.1 Architecture of a GAN

GAN is a framework used for unsupervised learning. It is a generative model based on deep learning and consists of two major components - the Generator and the Discriminator. Both the components are in themselves neural networks. The generator model, as the name suggests, is responsible for creating brand new data, after being trained on the training data set to understand the characteristics of the data it is attempting to produce. The discriminator model is responsible for essentially evaluating the quality of the data generated by the generator. It discriminates between generated data based on whether it can pass for real data or not. It makes use of the sigmoid function to make this determination with 0 representing a fake image and 1 corresponding to a real image.These two networks in a way compete against each other, resulting in the creation of variations of the training data. After the training phase of the GAN, the generator works in a way so as to attempt to maximize the likelihood of the discriminator making a mistake while the discriminator determines the probability of the generated data being real or not. A high level flow of the GAN is represented below.

Preprint. Under review.

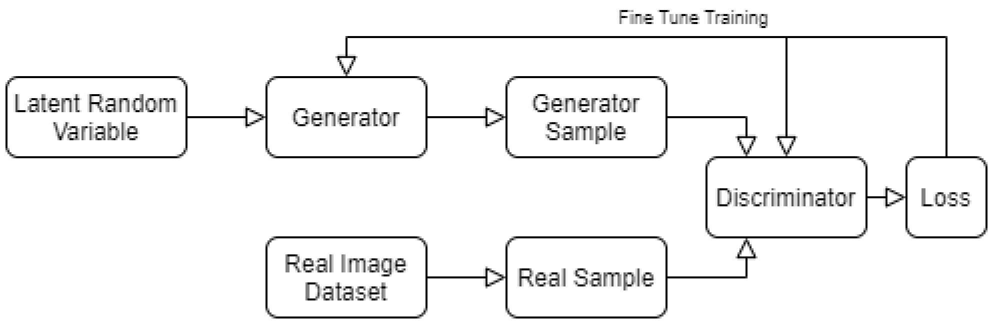

Figure 1: Architecture of GAN

1.2 Architecture of a VAE

VAE is a framework that is used for unsupervised learning, but it has some semi-supervised aspects to it. It is also a generative model and consists of an auto encoder with two major components - the Encoder and the Decoder. The objective of the VAE is also to generate new data, but in this case it is by reconstructing the input data. Ir works towards this objective as follows. The encoder part of the framework is responsible for compressing high dimensional data that is fed to it, converting it into a more manageable representation in fewer dimensions. This is known as the continuous latent space. It achieves this by making use of it's convolutional layers. The latent space representation is stochastic in nature and the vector representation consists of the mean and standard deviation being represented by two different vectors. The role of the decoder is to attempt to re-form the input images by making use of the de-convolutional layers. The quality of the output is dependent upon the constructed latent space. As the latent space increases in size, the optimization of the output also increases, improving the quality of the results. A high level flow of the VAE is represented below.

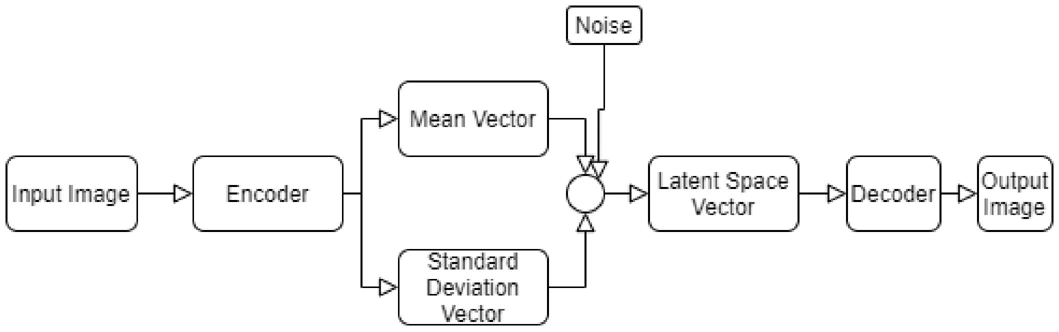

Figure 2: Architecture of VAE

2 Implementation and Challenges

2.1 GAN

2.1.1 Implementation

My implementation of the GAN network in done in Python, utilizing the Pytorch and Torchvision packages.
The discriminator is a multilayer perceptron (MLP) neural network defined sequentially. It consists of fully connected layers with an input layer which accepts inputs with 784 coefficients(MNIST data is 28 x 28 pixels). There are 3 hidden layers consisting of 1024, 512 and 256 nodes respectively with leaky ReLU activation, allowing for small negative slopes in non-positive inputs. The output layer consists of a single node with sigmoidal activation as it represents a probability.
The generator is implemented in a similar manner to the discriminator but the layers are structured

differently. The generator is fed with a 100 dimensional input as it needs to generate more complex data. It also consists of 3 hidden layers, this time consisting of 256, 512 and 1024 nodes respectively with leaky ReLU activation. The output layer consists of 784 nodes which will be arranged to form a 28 x 28 tensor representing the generated image. The activation for the output layer is the hyperbolic tangent function since the output coefficients are expected to be in the interval -1 to 1. The optimizer utilized for minimising loss is ADAM. The dropout layer is set to 0.3. Below is a basic flowchart describing the implemented network

Generator

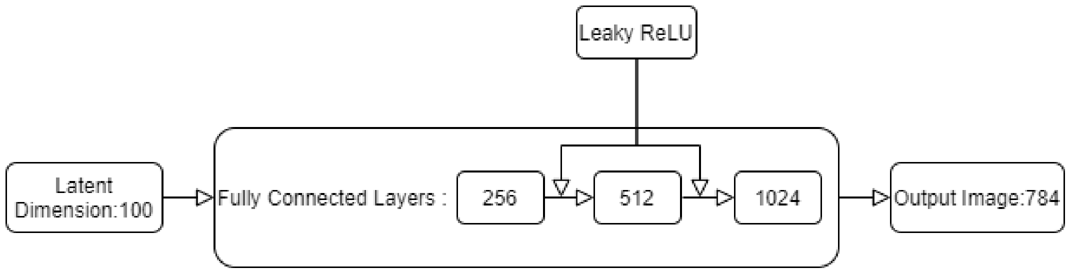

Figure 3: Flowchart for Generator Implementation

Discriminator

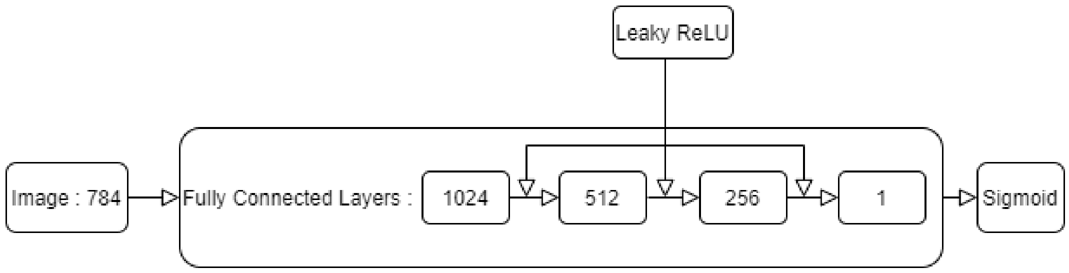

Figure 4: Flowchart for Discriminator Implementation

The training of the network is divided into two phases. An initial phase consisting of only a forward pass with no back propagation. In this pass the generator is not fed any training data while the discriminator is trained on real data in order for it to be able to distinguish between real and fake data accurately. The second phase of the training involves training of the generator in order for it to be able to produce samples that resemble the real data as closely as possible so that it can generate data samples that are as identical as possible to the real data. This phase involves back propagation of the loss as more data is fed in in order to improve the performance of the generator.

2.1.2 Challenges

A few challenges are observed with training a GAN network - Achieving a stable balance between the two modules, the generator and the discriminator can be a bit tricky. We need to ensure that mis-classification of data does not occur and an over tuned or under tuned discriminator can easily result in this happening. As such it is important to have a balanced set up for the best results. They must also have a similar learning rate to each other so there is minimal wait time in the working of the network. Another challenge was deciding the best structure for the various layers of both the generator and discriminator in order to achieve the best results along with the choice of activation function. The choices finalized for these have been stated above. The GAN network was also a lot slower to train in comparison to the VAE network for the same number of epochs

2.2 VAE

2.2.1 Implementation

My implementation of a VAE leveraged Pytorch. The network consists of fully connected linear layers.The network size here is much smaller than the GAN for achieving a good performance level. The encoder has three layers. The first encoder layer has 784 input features, corresponding to the total number of pixels in each image of the MNIST dataset (Each image is 28 x 28 pixels) and has 512 output features. The second encoder layer has 512 input features while having 256 output features. The third layer has 256 input features and 32 output features.
The decoder also consists of three layers which essentially are the reverse of the encoder layers. The first having 16 and 256 input and output features respectively while the second layer has 256 input and 512 output features and third having 512 input and 784 output features respectively. These will be presented in a 28 x 28 pixel pattern to obtain the image. Layers in this case have ReLU activation. Since the posterior here is modelled as Gaussian distribution, we get the mean and covariance as output parameters. Below is a basic flowchart describing the implemented network

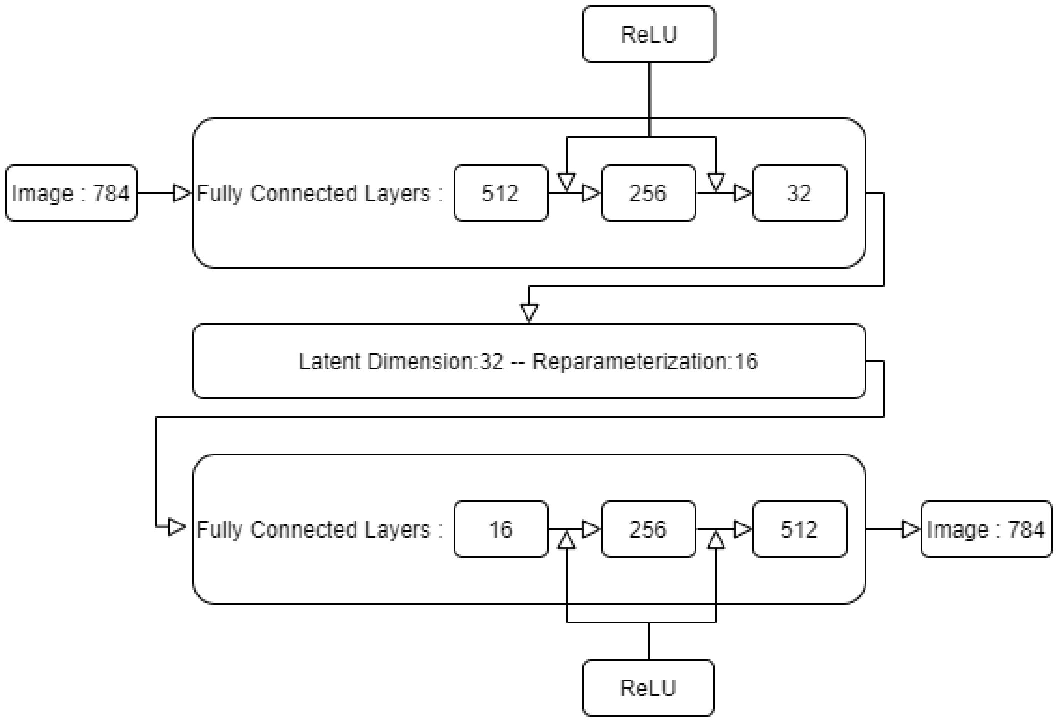

Figure 5: Flowchart for VAE Implementation

In training the network, we sample the training data into the latent space using the network specified above. The generated latent space is then leveraged to generate data with the objective of resembling the training data as much as possible.

$$-D_{KL}[q_\phi(z|x)||p_\theta(\dot{z})] + E_{q_\phi(z|x)}[log(p_\theta(x|z)]$$

Figure 6: VAE Loss Function

$$D_{KL}[q_\phi(z|x)||p_\theta(z)]$$

Figure 7: Reconstruction Loss

$$q_\phi(z|x)$$

Figure 8: Encoder Output

$$p_\theta(z)$$

Figure 9: Mean and Standard Deviation

$$E_{q_\phi(z|x)}[log(p_\theta(x|z)]$$

Figure 10: KL Divergence Loss

$$log(p_\theta(x|z)$$

Figure 11: Decoder Output

2.2.2 Challenges

Some of the challenges observed in training the VAE network include - Manual calculation of the loss function due to a lack of pre-existing libraries that support the calculation of metrics like the KL divergence loss. Additionally, due to the nature of the architecture of the VAE, the output of the encoder serves as the input for the decoder and as such any changes in the first require updating the second as well. From an available resource perspective, information and articles regarding GAN networks are plentiful while they are more sparse when it comes to VAE networks in the domain of image generation.

3 Observations and Results

3.1 GAN

The training was done on the training data set of the MNIST images available through torchvision. We transform these images to a PyTorch tensor and normalize it to the range of -1 to 1. Training was conducted for 10 epochs, then 50 and finally 200 with the loss values(BCE loss) for both discriminator and generator starting around the 0.5 mark. The discriminator loss converges to zero fairly quickly. The generator loss increases and fluctuates at first and over epochs settles around the 27 mark. When testing image generation, after each training epoch we observe the generated images improving in quality and resembling handwritten MNIST images more and more as epochs progress, reducing the noise in the images with each iteration. As we increases the number of epochs over which we train, we observe a cleat improvement in the generated images. There is a significant reduction in the noise of the image and a slow transition to images that start resembling the actual MNIST images very closely, being almost indistinguishable in some cases. The results are not perfect though. The noise levels is still noticeable and some generated images can still be easily distinguished from actual data. Below are sample output images from all 3 stages of training the models

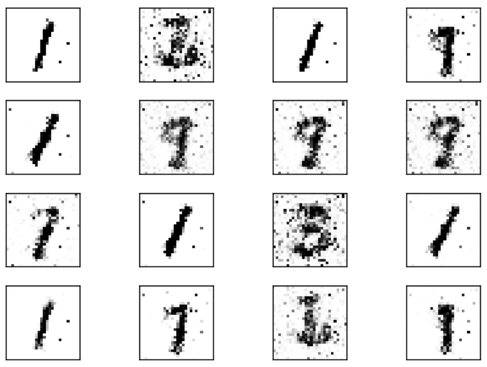

Figure 12: GAN output for 10 epochs

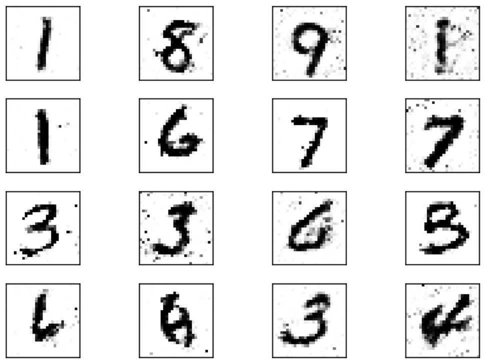

Figure 13: GAN output for 50 epochs

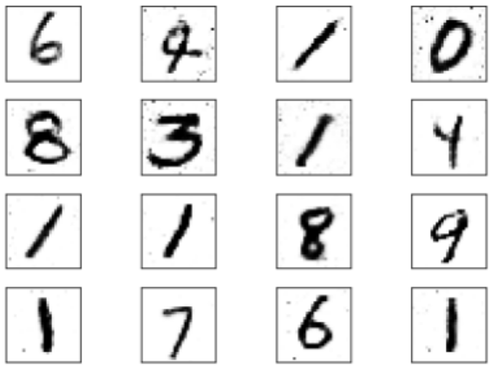

Figure 14: GAN output for 200 epochs

3.2 VAE

The training was done on the training data set of the MNIST images available through torchvision. We transform these images to a PyTorch tensor. Just like with the GAN, training was conducted first for 10 epochs, followed by 50 epochs and finally for 200. The results overall seem successful. The loss value (BCE Loss) starts at around 178 and we see a consistent reduction in the loss as we go through epochs. After 50 epochs, the loss value is observed to be down to around the 135 mark. Similar to GAN, the output images obtained after each epoch also show improvement in achieving results closer to the training images from the MNIST dataset. VAE images show a much lesser amount of noise in them when compared to the GAN however, the primary drawback for this network seems to be the blurry nature of the generated digits in comparison to GAN. We do see a significant improvement in the blurriness with more training. The results of a 10 epoch execution being very difficult to make out while by the time we train over 200, the quality of a lot of the images approaches desirable levels. Below are sample output images from all 3 stages of training the models

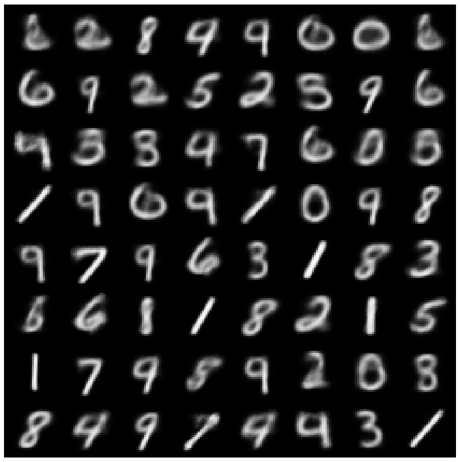
Figure 15: VAE output for 10 epochs

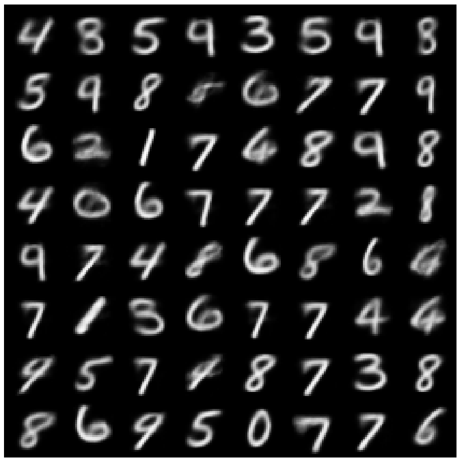
Figure 16: VAE output for 50 epochs

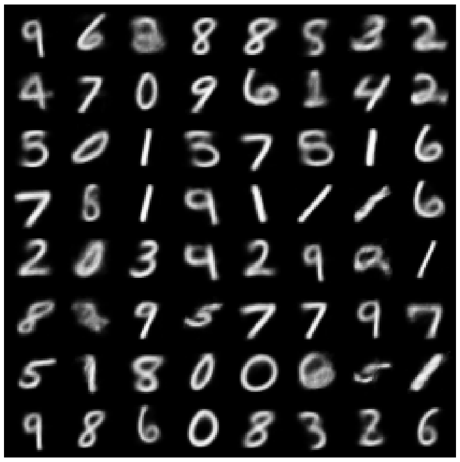
Figure 17: VAE output for 200 epochs

4 Conclusions

From my observations, I fell that both sets of outputs generate good results overall. The majority of the generated images closely resemble the images of the hand-written digits found in the MNIST data set and could easily be fit in with images of actual hand-written digits. However, neither of the two models generates perfect results and both have their own unique drawbacks.
In the case of the GAN, we observe that some of the generated images display a noticeable level of noise surrounding the base digit. This noise level is high at lower training epochs and improves as we train our model further, but even substantial training is unable to completely eliminate the noise. In the case of the VAE, we observe that while noise in the images is minimal, the digits themselves can be blurry at times. This is also something that improves with more training but that improvement stagnates beyond a threshold number of epochs.
However, if we were to evaluate the results of the two models againast each other, there are a few things to note. In case of the VAE network, we make an assumption of optimizing the lower variational bounds. This is not something we do in the case of the GAN. The GAN also poses a difficult challenge in the form of achieving convergence of the algorithm in a manner where the competing sub-modules of the generator and discriminator remain stable. We can observe this from the convergence graph as well.
In conclusion, based on the above factors and the observations with the training process and the generated results, the VAE does a better job of overcoming its shortcomings with time in comparison to the GAN. The image quality of the digits is better on average with the same number of epochs

trained and they are easier to identify on average than the GAN. The GAN is capable of producing individual images that may be higher in quality than those produced by the VAE, but the average image quality is lower.

5 Project Code

The code for the implementation of the GAN and the VAE networks can be found at - https://github.com/Sachet19/ML-Project

References

[1] https://developers.google.com/machine-learning/gan/gan_structure

[2] https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

[5] https://towardsdatascience.com/build-a-super-simple-gan-in-pytorch-54ba349920e4

[3] https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

[4] https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73