Evaluation and Comparison of VAE and GAN networks

Munish Tanwar [1]

Abstract

This paper discusses the Generator Adversarial Network (GAN) and Variational Autoencoders(VAE) are two deep generative models. Both networks perform the common task of generating new data from a distribution. Both networks also contain two components that get connected to form the complete model. This paper aims to study these networks, fine-tune them, and compare both networks to see which one performs better. First, we will discuss the background of each network and its basic structure. Then, we will construct the experiment where we define the dataset that will be used for this study. We will be using MNIST Handwritten Digit Dataset which will be preprocessed for them to be used with the networks. We will discuss the base architecture that will follow the standard implementation for each model and act as a control group. Then each model will make modifications in order to fine-tune their performance. Next, for both models will we discuss which modification works best of one. Lastly, we will compare the best iteration of each model and decide which model is better.

1. Introduction

The Generator Adversarial Network (GAN) and Variational Autoencoders (VAE) are two popular deep generative models used for generating new data from a distribution. Each network consists of two components that are connected to form the complete model. GAN consists of a generator network and a discriminator network. The generator model takes in latent data and generates new, fake image data. The discriminator is a standard classification network that classifies if the input data is fake or real. These networks fight against each other during training and update their weights. VAE consists of an encoder and decoder network. The encoder takes an input image and encodes it into a lower dimensional latent space. The decoder takes a sample from that latent space and tries to reconstruct the image. Both GAN and VAE provide a way to generate new fake data by capturing the input image data into latent distribution.

In this paper, we aim to study and compare the performance of these networks by fine-tuning them. First, we provide a background of each network and its basic structure. Then, we describe our experiment using the MNIST Handwritten Digit Dataset, which has been preprocessed for use with these networks.

For each model, we will construct a base architecture that follows the standard implementation and it will serve as a control group. Then fine-tune modifications will be made to improve their performance. For GAN, batch-normalization and tanh activation functions will be used to fine-tune the model. For VAE, regularization terms and higher latent dimensions with regularization will be used to fine-tune the model. For GAN, the discriminator loss and generator loss will be used to evaluate which modification performed better. For VAE, total loss, reconstruction loss, KL loss, and validation loss will be used to evaluate which model performed better. Once the best configuration for each model is chosen, then we will compare GAN and VAE to identify which generative model is better. The evaluation for this comparison will be based on the computation time and complexity of the network.

2. Background

2.1. GAN

The Generator Adversarial Network was first introduced in a 2014 paper by Goodfellow [1]. It consists of two components: a generator model and a discriminator model. The generator network performs the task of taking data from latent space or noise data and generating an image from it by a series of upscaling. The discriminator network is a standard classifier that takes in an image and classifies it as either real or fake. Both networks work against each other and train through this tug-of-war, the generator network creates a fake image and tries to trick the discriminator into labeling it as real. The discriminator aims to get better at classifying those fake images. Both networks are connected to make GAN. To train the GAN, the discriminator is the first train on the dataset that consists of both fake and real data. Then, GAN feeds data from latent space through the generator part of the network, the generator then outputs its generated image into the discriminator part of the network. During this part of the training, the GAN trains the generator and updates its weights based on how well it tricks the discriminator, while

the discriminator is frozen and untrained.

2.2. VAE

The Variational Autoencoder was introduced in a 2013 paper by Kingma and Welling [2]. VAEs also consist of two components: encoder and decoder. The encoder takes the input image as input and tries to encode it into a lower dimensional latent space as a distribution while maintaining maximum information about the data. The encoder in VAE outputs the mean and log variance which represent the distribution over the latent space. The decoder then samples data from that latent distribution using the mean and log variance and tries to reconstruct the encoder's input image. The VAE is constructed by combining these two components where the output of the encoder is fed into the decoder. Unlike, the GAN where the generator is trained separately from the discriminator, the VAE trains the encoder and decoder together. The VAE uses a custom loss function that combines reconstruction loss which is the error between the image generated by the decoder and the input image, and KL divergence which represents the loss of sampling from the latent space. The loss function which is called the Evidence Lower Bound (ELBO) express as follows [2]:

$$L(q,p) := E_{q(z|x)}[\log p_\theta(x|z)] - KL(q(z|x)|p_\theta(z)) \quad [2]$$

3. Experiment

3.1. Data set

The MNIST Handwritten Digit data set was used for conducting the experiment for both GAN and VAE networks. This data set consists of 70,000 grayscale images of size 28x28 for handwritten digits from 0 to 9. The datasets were loaded using the Keras library which provided training and testing datasets with labels (0-9) for each image. The training set consists of 60,000 data and the testing set consists of 10,000 data.

The images are initially 2D arrays of pixels and for GAN and VAE which primarily used convolution layers, the images were reshaped into 3D arrays with the extra dimension being the grayscale channel. The images were also initially between 0 and 255, and they were normalized to be between 0 and 1.

3.2. GAN

3.2.1. Base Architecture

The base architecture of the GAN discriminator consists of two convolutional layers. The first layer has 64 filters with a kernel size of 5 and stride of 2, while the second layer has 128 filters with a kernel size of 5 and stride of 2. The output of the last convolutional layer is then flattened and fed to the output layer, which uses the sigmoid activation function. Additionally, the discriminator uses LeakyReLU as its activation function, employs dropout regularization, and uses Adam optimization with a learning rate of 0.0002. The model is trained using binary cross-entropy as a loss function, and it outputs a binary classification indicating whether the input image is real or fake.

The generator network is responsible for generating a new and realistic handwritten image. It achieves this by first taking input from a 100-dimensional latent space drawn from a Gaussian distribution. The first layer is a dense layer with 256 nodes that takes 7x7 images, each with 256 channels. The output of this layer is then reshaped into 256 7x7 images. These 256 images are then upscaled using the Conv2DTranspose layer to obtain 128 7x7 images, which are then further upscaled to obtain 64 14x14 images. Finally, the 64 14x14 images are upscaled again to produce a single 28x28 image as output. The activation function used in the hidden layers is LeakyReLU.

The GAN is constructed by combining the generator and discriminator model in a way that the generator takes the input of a latent vector and outputs a fake image, then the discriminator takes the fake image and classifies it as either fake or real. The GAN trains the generator model using a learning rate of 0.0002 with Adam optimization and binary cross entropy loss function. The stochastic gradient descent is used to train GAN with a mini-batch size of 260 for 100 epochs. During the training phase, the discriminator is trained separately first and then the generator is trained while the discriminator's weights are frozen/untrainable.

3.2.2. Batch Normalization

The first change that I experimented with was adding batch normalization to both generator and discriminator models. The paper by Rasford, Metz, and Chintala suggested using batch normalization when building GAN using convolution layers. Based on their study, batch normalization improves training stability by normalizing the input to each unit to have zero mean and unit variance [3]. The study suggested that batch normalization can help prevent the generator from collapsing all the samples to a single point. Batch normalization also prevents internal covariate shift which can make it difficult for GAN to converge to a stable solution [3].

Batch normalization was added, to both generator and discriminator, after every hidden layer and before the LeakyReLU activation function.

3.2.3. Tanh Activation Function

Next, as part of the experiment, I made a modification to the base architecture by replacing the sigmoid activation function in the output layer of the generator model with the

tanh activation function. By doing so, the generator was able to produce images with pixel values ranging from -1 to 1, resulting in improved image quality. To ensure that the input values of the generator are within the range supported by the tanh function, I first preprocessed the MNIST dataset by normalizing the pixel values from 0 to 255 to -1 to 1. This change was also suggested by Radford and Metz in their paper [3]. In theory, the image produced by the generator will be more sharper and detailed.

3.3. VAE

3.3.1. Base Architecture

The first component of the Variational Autoecndoer is the encoder network. The encoder network in the Variational Autoencoder (VAE) consists of an input layer that takes in an image of size 28x28x1. This input is then passed through two convolutional layers. The first layer has 32 filters with a kernel size of 3 and a stride of 2, and the second layer has 64 filters with a kernel size of 3 and a stride of 2 as well. Both convolutional layers use the rectified linear unit (ReLU) activation function. The output of the second convolutional layer is a data with dimensions 7x7x64. This data is then flattened into a vector of length 3136. In VAE, unlike in a standard autoencoder, this vector is used to produce the distribution for the latent space from which the decoder can sample. The dimensionality of the latent space is set to 2 for the base architecture. To obtain the distribution, the encoder has two output layers, one for the mean and another for the variance. These output layers use the linear activation function to output the mean and log-variance of the distribution, respectively.

The parameters of the latent distribution are used to define a sampling function which takes the mean and variance with added randomness to get z, $z = \mu + \sigma\epsilon$ [2]. After obtaining a sample z from the latent distribution using the mean and log-variance parameters output by the encoder, the sample is passed to the decoder component of the VAE. The decoder's objective is to reconstruct the original input image from the sample. The decoder begins with a hidden dense layer that takes the sample as input and expands and reshapes it to a tensor of size 7x7x64. This tensor is then passed through two transposed convolutional layers with 64 and 32 filters, respectively, to upscale into a single 28x28 image. Both transposed convolutional layers use the ReLU activation function. The output layer returns a reconstructed image and uses the sigmoid activation function.

The VAE model consists of an encoder and a decoder. The encoder produces a sample from the input image, which is then fed into the decoder to reconstruct the image. The VAE loss function comprises reconstruction loss and KL divergence. To train the model, a custom loss function was created and added, and Adam optimization with a learning rate of 0.0002 and mini-batch size of 256 was used for 100 epochs.

3.3.2. Regularization Term

The first change that was made to the base architecture was adding an additional hyperparameter to the loss function with reconstruction loss and KL divergence. The regularization term was set 0.001. This change was inspired by the paper from Kumar and Poole on beta VAE [4]. This change enables better control of the trade-off between reconstruction and KL divergence, which encourages the VAE to learn a smoother and more disentangled latent space. The standard VAE loss function includes a KL divergence term that encourages the learned latent distribution to match a predefined prior distribution.

$$L(q,p) := E_{q(z|x)}[\log p_\theta(x|z)] - KL(q(z|x)|p_\theta(z)) \quad [2]$$

However, by scaling this term using the additional hyperparameter, the VAE can prioritize disentanglement and smooth latent space over the accuracy of the reconstruction.

$$L(q,p) := E_{q(z|x)}[\log p_\theta(x|z)] - \beta KL(q(z|x)|p_\theta(z)) \quad [4]$$

3.3.3. Higher Latent Dimension

The next improvement I made to the base architecture was to increase the latent space dimension from 2 to 4. This change allows the encoder to encode the input data into a higher-dimensional latent space, which provides VAE with a greater capacity to capture complex and high-dimensional patterns in the data. However, increasing the latent space dimension can also lead to overfitting To address this issue, I used the additional hyperparameter, which is set to 0.001, introduced in the previous step to regularize the KL divergence loss in the VAE's loss function. This helps to prevent the model from overfitting to the training data by balancing the trade-off between reconstruction and regularization.

4. Evalution and Results

4.1. GAN

To evaluate and compare the performance of GAN across iterations, we can analyze the discriminator through its accuracy in detecting real and fake images. The generator will be evaluated based on the images it generates after training.

Based on the discriminator classification accuracy, we can observe that the batch normalization model performs at detecting real images throughout the training phase and it also performs well at detecting fake images. The generator for batch normalization produces the sharpest and clear image of all the models. Based on both discriminator and generator performance, we can conclude that the batch normalization model performed the best.

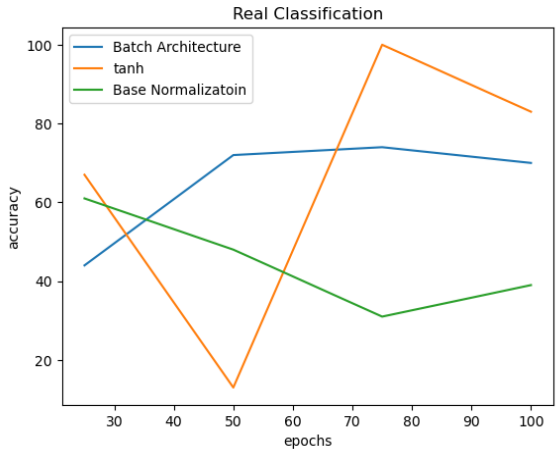

Figure 1. GAN Discriminator Real Accuracy

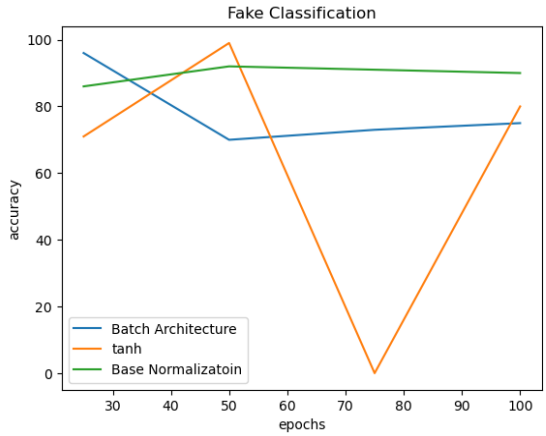

Figure 2. GAN Discriminator Fake Accuracy

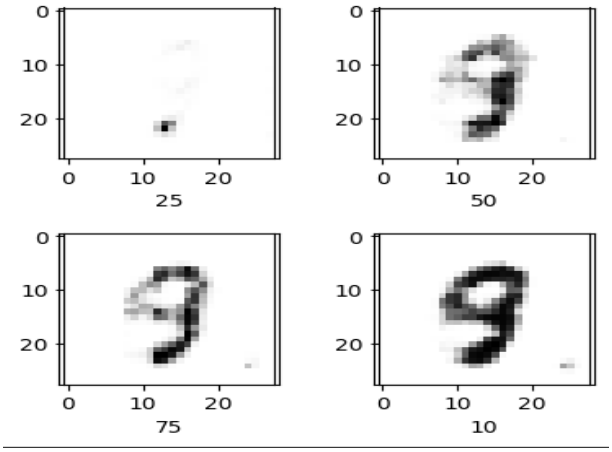

Figure 3. Image Generated by Base Arch. at each epoch

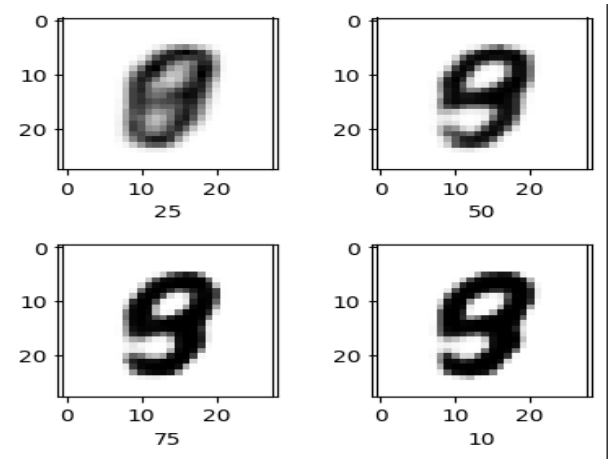

Figure 4. Image Generated by the Batch norm. model at each epoch

4.2. VAE

To evaluate and compare the performance of VAE models across iterations, we can analyze several metrics such as the total loss, reconstruction loss, KL divergence loss, and validation loss. By comparing these metrics across iterations, we can identify the model that was able to minimize the loss most effectively during the training process. This approach allows us to select the best VAE model that generates the most accurate and representative output while also ensuring that the model generalizes well to new data [figure 1 to figure 4].

Based on these metrics, we can observe that the base model and the model with the regularization term added to KL loss had similar performance. However, the model with a high latent space dimension of 4 outperformed the other two models on all metrics. A higher latent space dimension allows the VAE to capture more complex and high-dimensional patterns in the data, as it provides more capacity for encoding and decoding. However, increasing the latent space dimension also increases the complexity of the model and may lead to overfitting. Therefore, it is important to balance the dimensionality of the latent space with the regularization hyperparameter that was used to prevent overfitting.

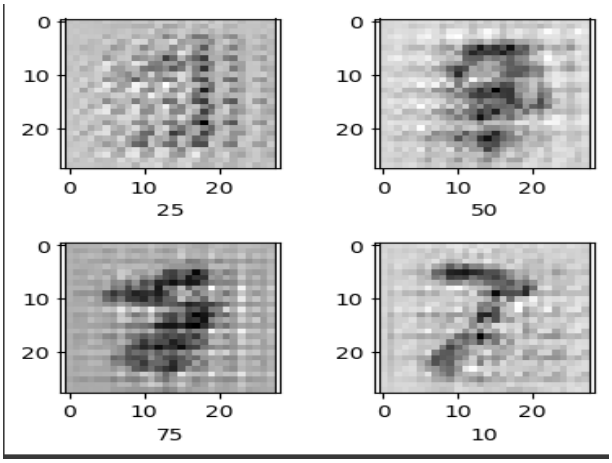

Figure 5. Image Generated by tahn model at each epoch

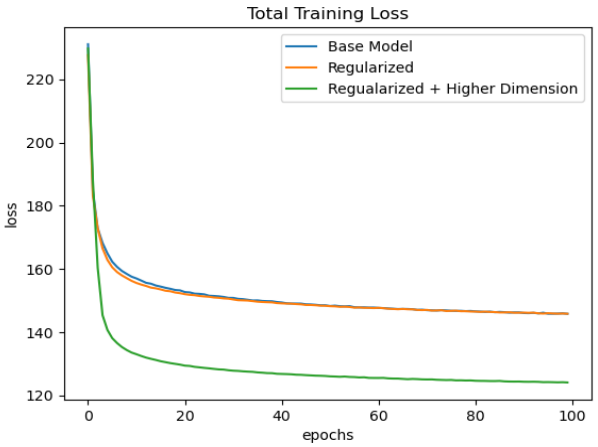

Figure 6. VAE Total Loss

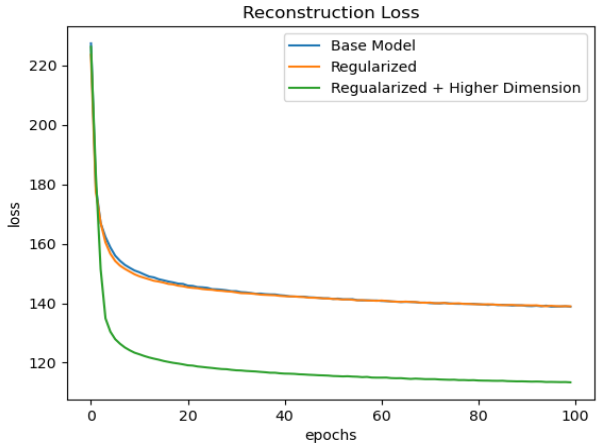

Figure 7. VAE Reconstruction Loss

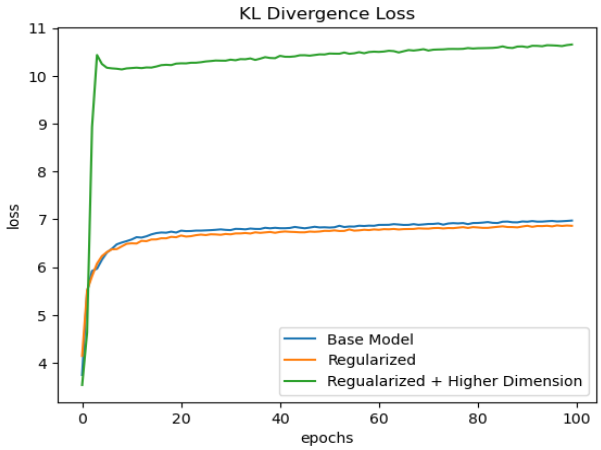

Figure 8. VAE KL Divergence Loss

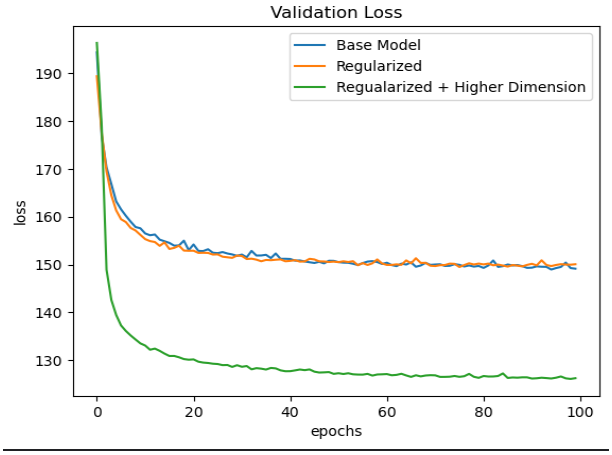

Figure 9. VAE Validation Loss

5. Comparison and Discussion

To evaluate and compare which generative model performs better in this study, we need to compare their training time and network size, and complexity. The best model for VAE consisted of 69,144 trainable parameters for the encoder and 71,361 trainable parameters for the decoder. While the best model for GAN consisted of 213,633 trainable parameters for the discriminator and 2,293,505 parameters for the generator. Furthermore, the training time for VAE was around 15 minutes with a batch size of 256 for 100 epochs. While the training time for GAN was around 57 minutes with a batch size of 256 for 100 epochs. Based on this, we can conclude for the given MNIST dataset, the VAE model with a latent space dimension of 4 and regularization term performs better than GAN.

6. Conclusion

In this paper, we discuss the basic background and structure of GAN and VAE. We constructed the base architecture for both networks and used MNIST handwritten digit dataset. For GAN, we experimented with batch normalization and tanh activation function as a means to fine-tune the model for better performance. For VAE, we experiment with regularization terms similar to beta VAE and higher latent dimension with beta regularization. We concluded that, for GAN, the batch normalization model performed the best and, for VAE, the higher latent dimensions ith regularization performed the best. Lastly, we compare both VAE and GAN and concluded that VAE's best model had lower training time and less complexity than GAN's best model.

References

[1] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., amp; Bengio, Y. (2014, June 10). Generative Adversarial Networks. arXiv.org. Retrieved April 20, 2023, from https://arxiv.org/abs/1406.2661

[2] Kingma, D. P., amp; Welling, M. (2022, December 10). Auto-encoding variational Bayes. arXiv.org. Retrieved April 20, 2023, from https://arxiv.org/abs/1312.6114

[3] Radford, A., Metz, L., amp; Chintala, S. (2016, January 7). Unsupervised representation learning with deep convolutional generative Adversarial Networks. arXiv.org. Retrieved April 20, 2023, from https://arxiv.org/abs/1511.06434

[4] Kumar, A., Poole, B. (2020). On Implicit Regularization in -VAE. Proceedings of the 37th International Conference on Machine Learning (ICML), 2020. Retrieved from https://arxiv.org/abs/2002.00041