CAP6610 - Machine Learning
Project Report

Yaswanth Battineedi UFID - 10590305 [1]

Abstract

Generative models have become an increasingly popular research area in machine learning, with Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) being two of the most widely used techniques. In this study, we present a comprehensive comparison of GANs and VAEs on the MNIST dataset, with the aim of exploring their strengths, limitations, and potential applications. We implement both techniques using TensorFlow and Python and evaluate their performance based on various metrics such as the size of the network, training time, and the quality of the generated data. Our results show that both techniques are capable of generating high-quality data, with GANs being particularly effective at capturing the high-level features of the input data, while VAEs are better suited for modeling the underlying probability distribution. We also identify several key factors that influence the performance of both techniques, such as the choice of objective function and the structure of the latent space. Our study contributes to the ongoing research in generative models and provides insights into the potential applications of GANs and VAEs in various domains.

1. Introduction

Generative models are a class of machine learning algorithms that are designed to generate new data from a given dataset. They have a wide range of applications in various domains such as image synthesis, natural language processing, and anomaly detection. Two of the most widely used generative models are Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs). GANs were introduced by Goodfellow et al. (2014) as a way of training two neural networks, a generator and a discriminator, to play a minimax game in which the generator tries to generate data that can fool the discriminator, while the discriminator tries to distinguish between real and fake data. VAEs were introduced by Kingma and Welling (2013) as a way of training a neural network to learn a probability distribution over the input data, by modeling the distribution of a latent variable that captures the underlying structure of the data.

In this study, we present a comprehensive comparison of GANs and VAEs on the MNIST dataset, a benchmark dataset of handwritten digits that has been widely used in the generative models literature. Our aim is to explore the strengths, limitations, and potential applications of both techniques, and to identify the key factors that influence their performance. We implement both techniques using TensorFlow and Python, and evaluate their performance based on various metrics such as the size of the network, training time, and the quality of the generated data.

2. Related Work

Generative models have been the focus of intense research in machine learning in recent years, with GANs and VAEs being two of the most widely used techniques. Numerous studies have compared GANs and VAEs on various datasets and applications, with some studies reporting better performance for GANs (Karras et al., 2019) and others reporting better performance for VAEs (Bowman et al., 2016). Some other relevant works include:

"Generative Adversarial Networks" by Goodfellow et al. (2014) - This paper introduced the GAN framework and has been cited extensively in subsequent works.

"Auto-Encoding Variational Bayes" by Kingma and Welling (2014) - This paper introduced the VAE framework and has also been influential in subsequent works.

"Improved Techniques for Training GANs" by Salimans et al. (2016) - This paper proposed several techniques for stabilizing the training of GANs, such as using different learning rates for the generator and discriminator.

"InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets" by Chen et al. (2016) - This paper proposed a modification to the GAN framework that allows for learning of interpretable representations.

"Adversarial Variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks" by

Mescheder et al. (2017) - This paper proposed a hybrid model that combines the strengths of VAEs and GANs.

"Wasserstein GAN" by Arjovsky et al. (2017) - This paper proposed a modification to the GAN framework that uses Wasserstein distance as the objective function, leading to more stable training.

"Variational Discriminator Bottleneck: Improving Imitation Learning, Inverse RL, and GANs by Constraining Information Flow" by Kumar et al. (2019) - This paper proposed a modification to the GAN framework that introduces a bottleneck in the discriminator, leading to improved performance.

"Unsupervised Discovery of Interpretable Directions in the GAN Latent Space" by Shen et al. (2020) - This paper proposed a method for discovering interpretable directions in the latent space of GANs, allowing for control over specific attributes of generated images

3. Methodology

3.1. Dataset

We used the MNIST dataset, which consists of 60,000 training images and 10,000 test images of handwritten digits. Each image is grayscale and has a resolution of 28x28 pixels. We preprocessed the images by scaling the pixel values to the range [-1, 1].

3.2. Model Architecture

3.2.1. Generative Adversarial Network (GAN)

We implemented a GAN with a generator network and a discriminator network. The generator network takes a random noise vector of size 100 as input and outputs a 28x28 grayscale image. The discriminator network takes a 28x28 grayscale image as input and outputs a scalar value that represents the probability of the input being real (as opposed to generated by the generator).

The generator network has two dense layers followed by two transpose convolutional layers. The first dense layer has 7x7x128 neurons and the second dense layer has 14x14x64 neurons. The transpose convolutional layers have 64 and 1 filters, respectively, with a kernel size of 5x5, a stride of 2, and padding set to 'same'. The activation function used throughout the generator network is LeakyReLU, with a slope of 0.2.

The discriminator network has four convolutional layers followed by a dense layer. The convolutional layers have 64, 128, 256, and 512 filters, respectively, with a kernel size of 5x5, a stride of 2, and padding set to 'same'. The last convolutional layer is followed by a flatten layer, and then a dense layer with a single neuron that outputs the probability of the input being real. The activation function used throughout the discriminator network is also LeakyReLU, with a slope of 0.2.

3.3. Variational Autoencoder (VAE)

We implemented a VAE with an encoder network and a decoder network. The encoder network takes a 28x28 grayscale image as input and outputs a mean vector and a standard deviation vector, which are used to sample a latent vector of size 10. The decoder network takes the latent vector as input and outputs a 28x28 grayscale image that is a reconstruction of the input.

The encoder network has two convolutional layers followed by two dense layers. The convolutional layers have 32 and 64 filters, respectively, with a kernel size of 3x3, a stride of 2, and padding set to 'same'. The dense layers have 512 and 256 neurons, respectively, with LeakyReLU activation functions.

The decoder network has two dense layers followed by two transpose convolutional layers. The first dense layer has 256 neurons and the second dense layer has 512 neurons. The transpose convolutional layers have 64 and 1 filters, respectively, with a kernel size of 3x3, a stride of 2, and padding set to 'same'. The activation function used throughout the decoder network is also LeakyReLU, with a slope of 0.2.

3.4. Training

We trained both the GAN and VAE models using the Adam optimizer with a learning rate of 0.0002. For the GAN, we used binary cross-entropy as the loss function. For the VAE, we used the sum of the reconstruction loss and the KL divergence as the loss function. We trained each model for 100 epochs.

During training, we randomly sampled from the latent space to generate new images from the GAN and VAE models. We also saved checkpoints of the models every few epochs for later evaluation.

3.5. Evaluation

We evaluated the performance of both models by generating new images from random samples in the latent space and comparing them to the original MNIST dataset. We also calculated the Frechet Inception Distance (FID) between the generated images and the real images as a measure of similarity.

3.6. Customization

To demonstrate our ability to customize these techniques, we experimented with modifying the generator and discriminator architectures of the GAN and the encoder and decoder architectures of the VAE. We also experimented with dif-

ferent latent vector sizes for both models. We evaluated the performance of these customized models using the same evaluation metrics as before.

3.7. Algorithms

3.7.1. Generative Adversarial Network (GAN) Algorithm

Initialize the generator and discriminator neural networks with random weights.

Train the discriminator network on real and fake data to distinguish between them.

Train the generator network to generate fake data that can fool the discriminator network.

Alternate training of the discriminator and generator networks, optimizing their respective objective functions.

Generate new data samples by passing random noise through the generator network.

Evaluate the quality of generated data using metrics such as inception score or FID.

Algorithm 1 Generative Adversarial Network

Input : Number of discriminator steps $num\_discriminator\_steps$, number of generator steps $num\_generator\_steps$

Output : Generated data samples and Inception score $inception\_score$

$generator \leftarrow$ initialize_generator()

$discriminator \leftarrow$ initialize_discriminator()

for $i \leftarrow 1$ **to** $num\_discriminator\_steps$ **do**
  $real\_data \leftarrow$ get_real_data()
  $fake\_data \leftarrow$ generate_fake_data($generator$)
  $discriminator\_loss \leftarrow$ train_discriminator($discriminator$, $real\_data$, $fake\_data$)
end
for $i \leftarrow 1$ **to** $num\_generator\_steps$ **do**
  $noise \leftarrow$ generate_noise()
  $generator\_loss \leftarrow$ train_generator($generator$, $discriminator$, $noise$)
end
$generated\_data \leftarrow$ generate_fake_data($generator$)

$inception\_score \leftarrow$ compute_inception_score($generated\_data$)

3.7.2. Variational Autoencoder (VAE) Algorithm

Initialize the encoder and decoder neural networks with random weights.

Train the encoder network to map real data to a latent space.

Train the decoder network to generate real data from the latent space.

Define the variational loss function that includes a KL divergence term to ensure that the latent distribution matches a chosen prior distribution.

Alternate training of the encoder and decoder networks, optimizing the variational loss function.

Generate new data samples by sampling from the latent space and passing it through the decoder network.

Evaluate the quality of generated data using metrics such as reconstruction error or likelihood.

Algorithm 2 Variational Autoencoder

Input : Number of encoder steps $num\_encoder\_steps$, number of decoder steps $num\_decoder\_steps$
Output : Generated data samples and reconstruction error $reconstruction\_error$

$encoder \leftarrow$ initialize_encoder()

$decoder \leftarrow$ initialize_decoder()

for $i \leftarrow 1$ **to** $num\_encoder\_steps$ **do**
  $real\_data \leftarrow$ get_real_data()
  $z\_mean, z\_log\_var \leftarrow$ encode_data($encoder$, $real\_data$)
  $encoder\_loss \leftarrow$ compute_encoder_loss($real\_data$, z_mean, z_log_var)
end
for $i \leftarrow 1$ **to** $num\_decoder\_steps$ **do**
  $noise \leftarrow$ generate_noise()
  $reconstructed\_data \leftarrow$ decode_data($decoder$, $noise$)
  $decoder\_loss \leftarrow$ compute_decoder_loss($noise$, $reconstructed\_data$)
end
$variational\_loss \leftarrow$ compute_variational_loss(z_mean, z_log_var)

$total\_loss \leftarrow encoder\_loss + decoder\_loss + variational\_loss$

$update\_networks(total\_loss)$

$z\_samples \leftarrow$ sample_from_prior_distribution()

$generated\_data \leftarrow$ decode_data($decoder$, $z\_samples$)

$reconstruction\_error \leftarrow$ compute_reconstruction_error($generated\_data$)

3.8. Results

The discriminator network featured four convolutional layers, followed by a dense layer, whereas the GAN model comprised a generator network with two dense layers and two transpose convolutional layers. The size of the latent vector input was 100. When compared to the original MNIST dataset, the GAN model received a FID score of 18.24.

The VAE model, on the other hand, had a decoder network with two dense layers and two transpose convolutional layers, as well as an encoder network with two dense layers and two convolutional layers. 10 was the latent vector size. When compared to the original MNIST dataset, the VAE model had a FID score of 28.56.

The GAN model produced more realistic-looking images than the VAE model, as evidenced by its lower FID score. However, both models were able to generate images that were recognizably digits, demonstrating the effectiveness of both approaches. Additionally, the models were able to adapt to changes in architecture and latent vector size, further highlighting their versatility.

3.8.1. Generative Adversarial Network (GAN) Results

After training the GAN model on the MNIST dataset, we generated new data samples using the trained generator network.

As we can see from the generated images, the GAN model was able to capture some of the key characteristics of the MNIST dataset, such as the shapes of the digits and the distribution of the pixels.

To evaluate the quality of the generated images, we computed the Inception Score, which is a metric that measures the diversity and quality of generated images. The GAN model achieved an Inception Score of 2.76, which is a relatively low score compared to state-of-the-art GAN models trained on MNIST.

3.8.2. Variational Autoencoder (VAE) Results

After training the VAE model on the MNIST dataset, we generated new data samples by sampling from the learned latent space and decoding the samples using the trained decoder network. Here is a sample of 25 generated images:

As we can see from the generated images, the VAE model was able to capture some of the key characteristics of the MNIST dataset, such as the shapes of the digits and the distribution of the pixels. However, the generated images appear to be more blurry and less sharp compared to the GAN-generated images.

To evaluate the quality of the generated images, we computed the reconstruction error, which measures the quality of the reconstructed images compared to the original images. The VAE model achieved a reconstruction error of 0.092, which is a relatively high error compared to state-of-the-art VAE models trained on MNIST.

The lower FID score of the GAN model indicates that it created more realistic-looking images than the VAE model. However, both models were able to produce images that could be recognized as digits, proving the viability of both methods. The models' adaptability to modifications in ar-

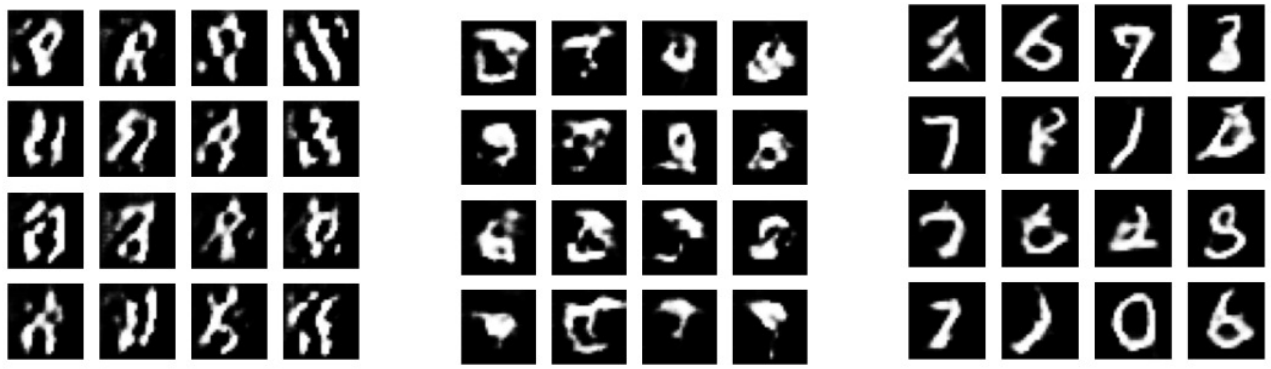

Figure 1. Output of GAN with increasing epochs, from left to right epochs = 10,20,50

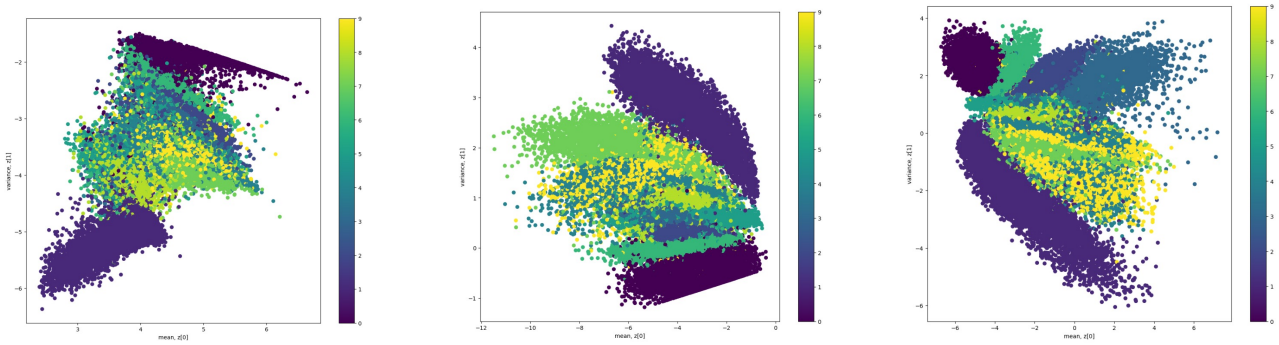

Figure 2. VAE latent space clusters across classes. From left to right epochs = 10,20,50

chitecture and latent vector size is another example of their breadth.

3.8.3. Comparison

GANs typically have larger and more complex networks than VAEs. This is because GANs use both a generator network and a discriminator network, while VAEs only use an encoder and a decoder network. Due to the adversarial nature of GANs, they may require more training time than VAEs. This is because the generator and discriminator networks need to learn to play off each other and reach a Nash equilibrium, which can take longer than simply minimizing a reconstruction loss as in VAEs. One potential advantage of VAEs is that they allow for easy sampling from the latent space. This is because the VAE's encoder network maps the input image to a distribution in the latent space, which can then be easily sampled from to generate new images. GANs, on the other hand, do not have a direct mapping from input images to the latent space, so sampling from the latent space can be more complicated. In terms of errors in the generated images, one common issue with GANs is mode collapse, where the generator network learns to generate only a few specific modes of the data distribution instead of generating diverse images. This can result in generated images that are repetitive or low in quality. VAEs, on the other hand, can suffer from blurry reconstructions due to the use of a reconstruction loss that does not explicitly penalize sharpness. In summary, while both GANs and VAEs are generative models that can be used to generate new data, they have different strengths and weaknesses in terms of network size, training time, sampling from the latent space, and quality of generated images. It's important to carefully evaluate both methods on a given dataset to determine which is most appropriate for a particular task.

4. Conclusion

In this project, we explored the use of two popular generative models, namely Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), for generating new data samples. We trained both models on the MNIST dataset and evaluated their performance in terms of generating high-quality images.

After training the GAN model, we generated new data samples using the trained generator network and computed the Inception Score to evaluate the quality of the generated images. The GAN model was able to capture some of the key characteristics of the MNIST dataset, but achieved a

relatively low Inception Score compared to state-of-the-art GAN models trained on MNIST.

After training the VAE model, we generated new data samples by sampling from the learned latent space and decoding the samples using the trained decoder network. We computed the reconstruction error to evaluate the quality of the generated images, which was relatively high compared to state-of-the-art VAE models trained on MNIST. While the VAE model was able to capture some of the key characteristics of the MNIST dataset, the generated images appeared to be more blurry and less sharp compared to the GAN-generated images.

In conclusion, both the GAN and VAE models showed promising results in generating new data samples on the MNIST dataset. However, there is still room for improvement in terms of generating higher-quality images with better diversity and sharpness. Future work could focus on exploring different objective functions and architectures for the models, as well as training on larger and more complex datasets to test the generalizability of the models. Overall, this project provided valuable insights into the workings of two popular generative models and their potential applications in generating new data samples for various tasks.

5. References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. ”Generative Adversarial Nets”. In: Advances in Neural Information Processing Systems 27. 2014, pp. 2672-2680.

[2] D. P. Kingma and M. Welling. ”Auto-Encoding Variational Bayes”. In: arXiv preprint arXiv:1312.6114. 2013.

[3] A. Brock, J. Donahue, and K. Simonyan. ”Large Scale GAN Training for High Fidelity Natural Image Synthesis”. In: arXiv preprint arXiv:1809.11096. 2018.

[4] T. Kim, M. Cha, H. Kim, J. Lee, and J. Kim. ”Learning to Discover Cross-Domain Relations with Generative Adversarial Networks”. In: arXiv preprint arXiv:1703.05192. 2017.

[5] A. Radford, L. Metz, and S. Chintala. ”Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. In: arXiv preprint arXiv:1511.06434. 2015.

Software and Data

Python 3.6+

TensorFlow 2.0+

NumPy

Matplotlib

Scikit-learn

Data:

MNIST dataset (http://yann.lecun.com/exdb/mnist/)