A study on Synthetic Data Generation

Naman Singh Nayal

April 22, 2023

Indian Institute of Information Technology, Guwahati
Advisor: Dr. Aparjita Dutta

Abstract

In this era dominated by machine learning and deep learning algorithms, one specific limitation that everyone faces is the lack of good data to train and test our models with. This can be due to security or privacy reasons and at times the lack of data can be because of a lack of real-world examples. While there exist several methods which deal with the lack of data, one method that stands out is Synthetic data generation. Synthetic data generation utilizes the existing data and generates a set of synthetic data using generative models, which deals with the issues of privacy and security. This report discusses several different methods of data generation, explaining the maths behind them as well as the implementation along with drawing a comparison among them to conclude which models work best for which situation as well as discussing their scope and limitations.

1 Introduction

While going through several research papers which applied different models on different data sets, a limitation that was common across all of them was the lack of good data. Most of the models, may it be regression, classification or something else require data to train the model with. A good data set isn't biased towards a particular subset of data and can provide a good variety. The data set should also be a reflection of the real world and should represent properly all the intricacies so that it can aid in making a proper model. For example, a human skin disease classification model, which required a lot of data, did not have relevant data because people didn't want to share the images of their skin. This led to a small data set that to having pictures taken at a specific angle. Thus models trained on this data set were not able to capture the actual features of the data set leading to poor performance. To overcome this we look towards synthetic data generation via the use of generative models. Generative models take noise as input and produce desired output which is a reflection of real-world data. They can augment a given image to introduce variance that is required by the data set, eliminating the bias also the images produces are synthetic, thus they don't have privacy-based issues. This report looks towards three individual synthetic data generation models, that being Variational Auto-encoders, Generative Adversarial Networks, and finally the Denoising Diffusion Probabilistic Models and how quickly, and efficiently they generate images as well as the quality and variety of the images produced. We will discuss the mathematics behind the models and the loss of the models over the MNIST data set.

2 Variational Auto-encoders

2.1 Introduction to the Model

The class of generative models known as Variational Auto-encoders (VAEs) offers a logical approach to sampling from the model distribution. It is made up of an encoder and a decoder that are represented by neural networks. The encoder takes a picture as input and encodes the image's mean and variance in the latent space. The decoder then uses this input to create an image by decoding the encoded information in the latent space.

2.2 Working behind VAE

In traditional computer science, we have always searched for the best techniques to reduce the size of a particular file, such as an image or a paper. A unique kind of neural network called an autoencoder has a bottleneck layer for dimensionality reduction called latent representation. A dimensionality reduction method's main objective is to identify the top encoder/decoder pair within a given family. In other words, we are seeking the pair of encoders and decoders that retains the most information during encoding and, thus, have the least amount of reconstruction error during decoding for a given set of feasible encoders and decoders.

An encoder and a decoder are configured as neural networks, and the best encoding-decoding scheme is learned via an iterative optimization process. This is the basic concept of auto-encoders. To update the weights of the networks, we back-propagate the error through the architecture after each iteration, compare the encoded-decoded output with the starting input, and then feed new data into the auto-encoder architecture (the encoder followed by the decoder).

A variational auto-encoder is an architecture made up of both an encoder and a decoder that is trained to reduce the reconstruction error between the encoded-decoded data and the original data, just like a regular auto-encoder. To somewhat alter the encoding-decoding process, we encode an input as a distribution over the latent space rather than a single point to introduce some regularization of the latent space. After that, the model is trained as follows:

1 Distribution over the latent space is used to encode the input.

2 That distribution is sampled from a point in the latent space.

3 The sampled point is decoded, and it is possible to calculate the reconstruction error.

4 The network experiences back-propagation of the reconstruction error

2.2.1 Loss function of VAE

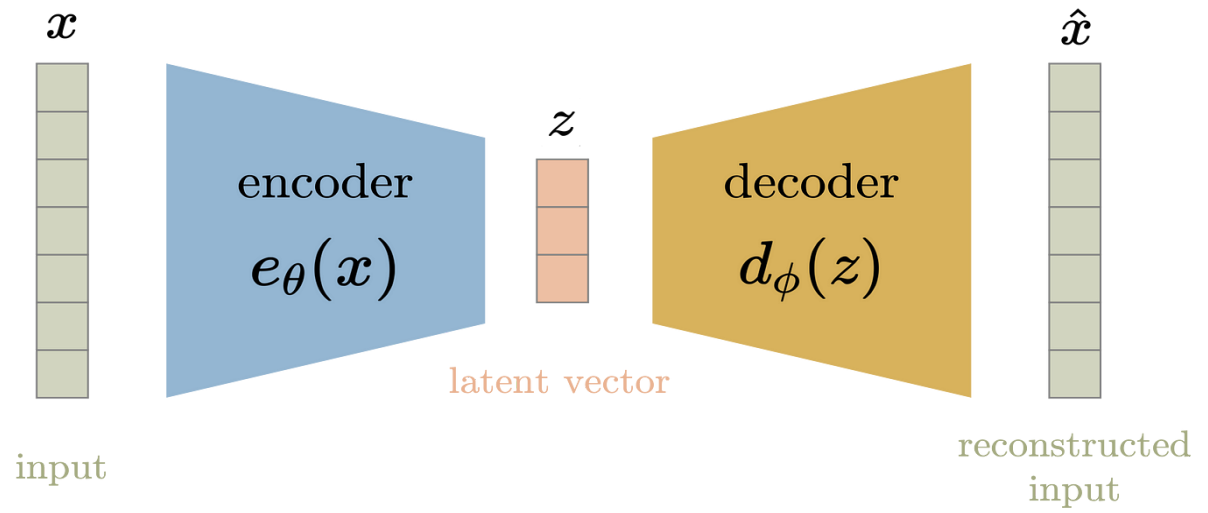
The loss function design of the VAE is one of its fundamental ideas. In other words, we are attempting to build the loss so that it pertains to the full distribution and does not over-fit to just the image itself, while also reconstructing well based on the given images. As a result, the VAE loss is the result of:

1 To maximize the similarity of the reconstruction, Binary Cross Entropy (BCE) Loss measures the pixel-to-pixel difference between the reconstructed and original image. BCE Loss is determined as follows:

$$\Sigma_{i=1}^{n} x_i' \log(x_i) + (1 - x_i') \log(1 - x_i) \tag{1}$$

where x_i and x_i' signify, respectively, the original and rebuilt picture pixels (total n pixels).

2 KL-Divergence Loss — KL divergence calculates how similar two distributions are to one another. Assuming that the distribution is normal in this instance, the loss is created as follows:



(a) VAE Block diagram

$$\Sigma_{i=1}^{m} = \sigma^2 + \mu^2 + \log(\sigma_i) - 1 \tag{2}$$

which is calculated via our predicted mean and sigma of every value in the latent vector (size m)

Consider the scenario where we want to produce the observation x from a distribution z. In order to calculate (p(z—x)), we can use the method shown below:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{3}$$

$$p(x) = \int p(x|z)p(z)dz \tag{4}$$

It is typically an intractable distribution as a result. Therefore, to convert it into a tractable distribution, we must approximate p(z—x) to q(z—x). We will minimize the KL-divergence loss, which determines how similar two distributions are, to more accurately approximate p(z—x) to q(z—x):

$$\min KL(q(z|x)||p(z|x)) \tag{5}$$

By simplifying, the above minimization problem is equivalent to the following maximization problem :

$$E_{q(z|x)} \log p(x|z) - KL(q(z|x)||p(z)) \tag{6}$$

The first term denotes the likelihood of reconstruction, while the second term guarantees that our learned distribution q resembles the real prior distribution p. As a result, our total loss is made up of two terms: reconstruction error and KL-divergence loss:

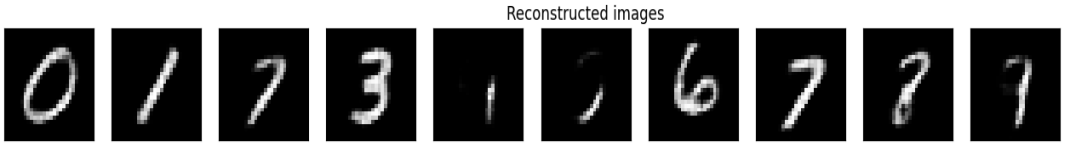
$$Loss = L(x, \hat{x}) + \Sigma_j KL(q_j(z|x)||p(x)) \tag{7}$$

2.3 Implementation on MNIST data set

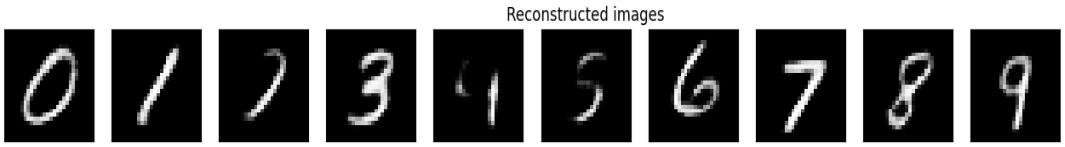
The MNIST data set is a collection of handwritten digits from 0 to 9. Our VAE model will take the images as input and then pass them through an encoder, reducing its dimensions down to the latent space from where it will sample and generate new images via the encoder.

2.3.1 Result

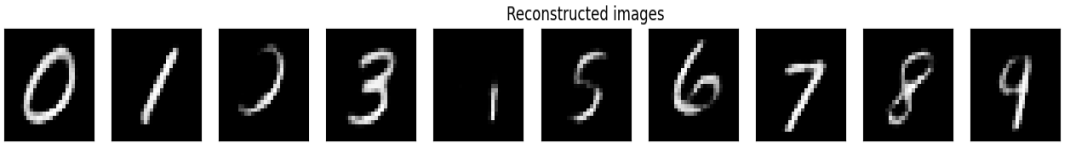
We train the model for 100 epochs and get the images synthesized by the generator for every epoch. Displaying a set of generated images:



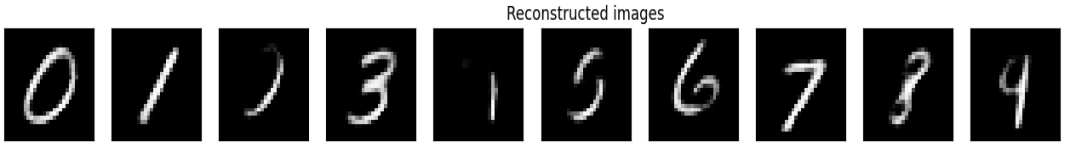
(a) Epoch 1



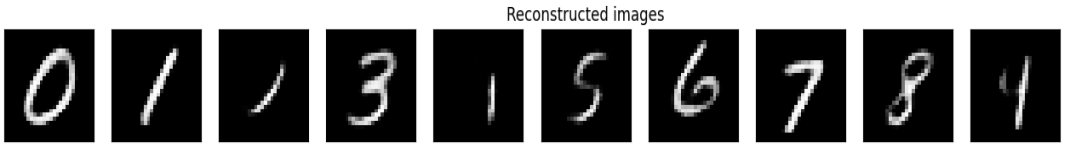
(a) Epoch 20



(a) Epoch 50



(a) Epoch 70



(a) Epoch 99

2.3.2 Graph of Epoch vs Loss

The synthetic images tend to get better and better after every epoch. This can be shown precisely by plotting the loss of VAE on a graph over the 100 epochs on both training and validation data:

(a) VAE Loss vs Epoch

2.4 Limitations of VAE

The biggest limitation of VAE is that the generated output tends to be blurry and lacks sharp contrast between the background and the output. This reduces the quality of the generated image making it less viable to be used for data sets that rely on the high quality of the image to work on. This lack of quality makes the model unusable in a lot of cases.

2.5 Uses of VAE

The best use case for VAE is in text generation as text data does not rely upon quality in regards to pixels. Another use case is when time and variation in data are of the essence as VAE is the fastest model along with being the one that produces varying outputs.

3 Generative Adversarial Networks

3.1 Introduction to the Model

Deep neural net topologies known as "Generative Adversarial Models" (GANs) feature two neural networks that are in direct competition with one another. Generative refers to the process of creating a probability distribution function that closely resembles the distribution function of the original data. The conflict between the discriminator and generator networks is referred to as adversarial. As a benchmark for any new synthetic image-generating model that emerges, GANs have shown to be particularly beneficial in the synthetic data production department.

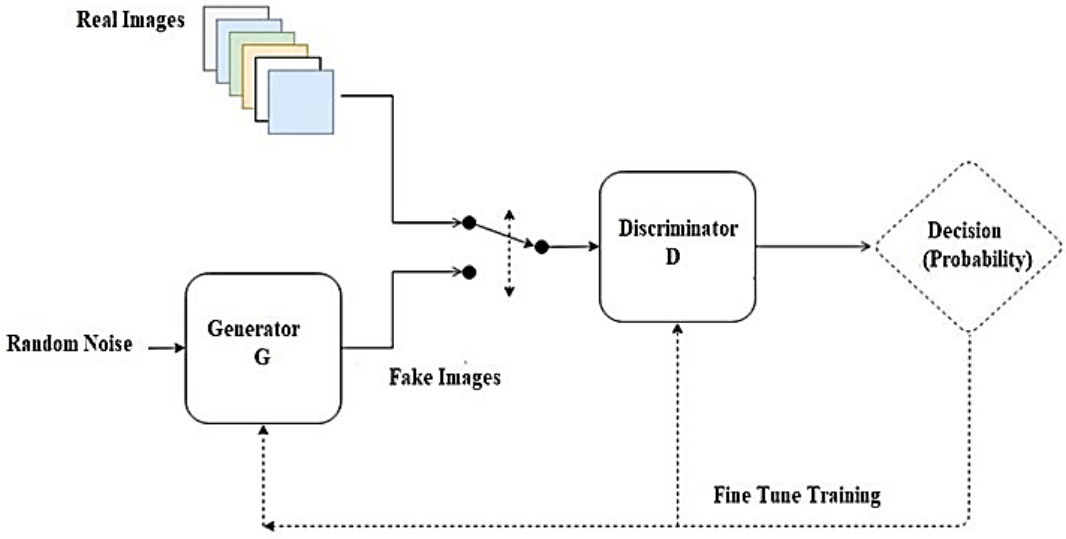
3.2 Working behind GANs

The fundamental premise of GANs is an intriguingly straightforward one: let us pit two neural networks against one another in the hopes that the rivalry would spur them on to domination. The main objective is to produce material, such as photos, that is identical to the content found in the training data. Two distinct models are required to accomplish it:

1 A discriminator's job is to categorize images as real or fake based on their input, which might be actual or generated by a generator.

2 A generator that outputs graphics based on the input of random noise.

The generator's objective is to produce realistic-appearing images that will deceive the discriminator, which is responsible for distinguishing between false and real pictures. Imagine the generator as an art forger and the discriminator as a police detective on a mission to catch the forger. This is a helpful anthropomorphization of the two models. Because the two networks' objectives are at odds with one another, each network should gradually improve during training. The generator should eventually be able to produce visuals that resemble reality.



(a) GAN Block diagram

GANs generate an artificial image by sampling noise and running it through a generator. The discriminator uses this and an actual image to determine if an image is phony or real. The discriminator and generator parameters are learned using a back-propagation method. The binary loss entropy function is the loss function employed by GANs.

3.2.1 Loss function of GAN

Basic conventions for understanding loss function:

1 The set of real data is represented by the probability density function $p_{data}(x)$, where x is a sample from the set of real data.

2 The set of noise data is represented by the probability density function $p_z(z)$, where z is a sample from the set of noise data that will be sampled by the generator to create the set of synthetic data.

3 The generator is represented as G() and discriminator is represented as D()

4 The generator G takes a noise sample z and based on parameters Θ_1, we generate a fake sample x. This set of synthetic/fake data is represented by the probability density function $p_g(x)$, where x is a sample from the set of synthetic/fake data.

$$G(z, \Theta_1) = x \rightarrow p_g(x) \tag{8}$$

$D(x, \Theta_2)$ =Probability that x is coming from original data set

Now we will look at the formulation of the loss function: GANs use the binary loss entropy function $L(\hat{y}, y) = [y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$ as their basis.

The labels for $p_{data}(x)$ is $y = 1$ and $\hat{y} = D(x)$ so putting this:

$$L(D(x), 1) = \log(D(x)) - (1) \tag{9}$$

For data coming from generator, the label is $y = 0$ and $\hat{y} = D(G(z))$ so in that case:

$$L(D(G(z)), 0) = (1 - 0) \log(1 - D(G(z))) = \log(1 - D(G(z))) - (2) \tag{10}$$

The objective of the discriminator is to correctly classify fake vs real. For this $\log(D(x))$ and $\log(D(G(x)))$ should be maximized.

$$\max \log(D(x)) + \log(1 - D(G(z))) \tag{11}$$

Objective of the generator is to produce $D(G(z)) = 1$

$$\min \log(D(x)) + \log(1 - D(G(z))) \tag{12}$$

The task of D is to maximize and the task of G is to minimize. Combining them we get:

$$\min_G \max_D \log(D(x)) + \log(1 - D(G(z))) \tag{13}$$

Considering all instances of x, we take the expectation :

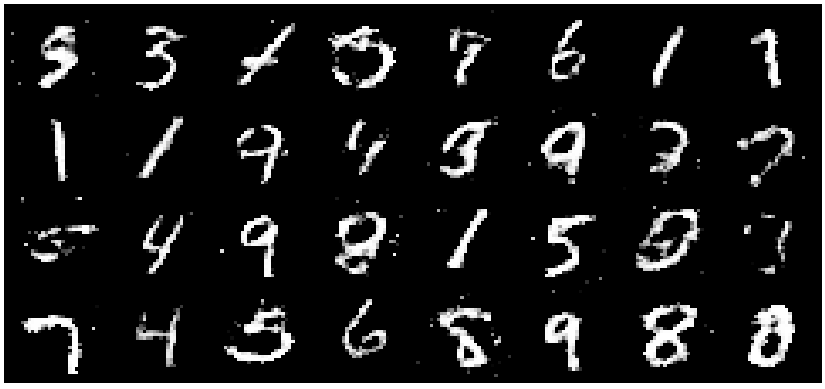
$$\min_G \max_D V(D, G) = E_{x \epsilon p_data(x)}[\log(D(x))] + E_{z \epsilon p_z(z)}[\log(1 - D(G(z)))] \tag{14}$$

3.3 Implementation on MNIST data set

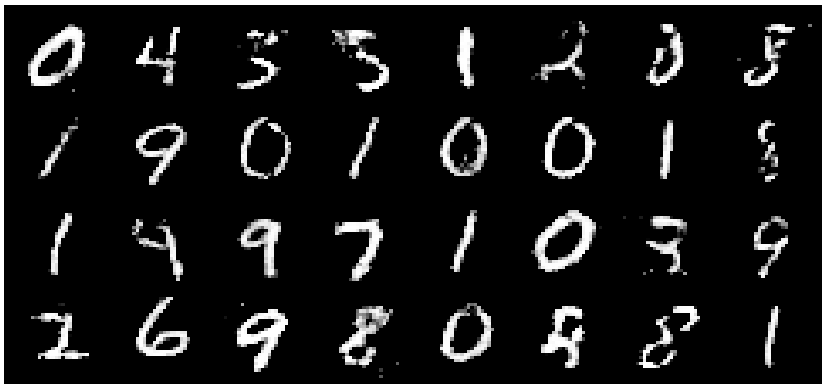
The MNIST data set is a collection of handwritten digits from 0 to 9. Our GAN model will take the images as input and then generate their synthetic handwritten digits. First, we build the neural network for the discriminator and the generator, then using the loss function we train them both to get the optimal results.

3.3.1 Result

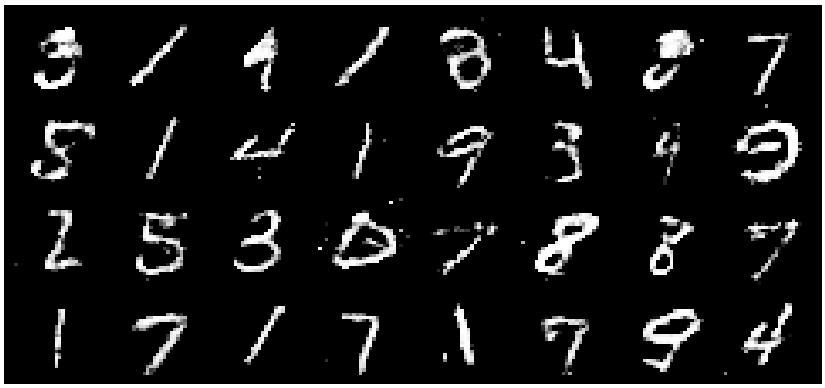
We train the model for 100 epochs and get the images synthesized by the generator for every 10 epochs:



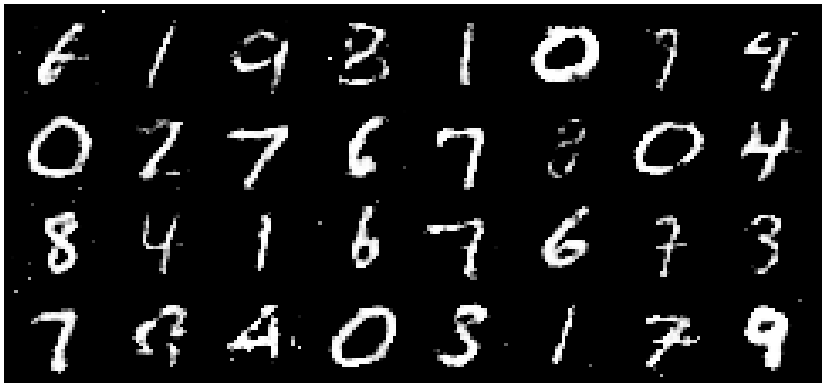
(a) Epoch 1



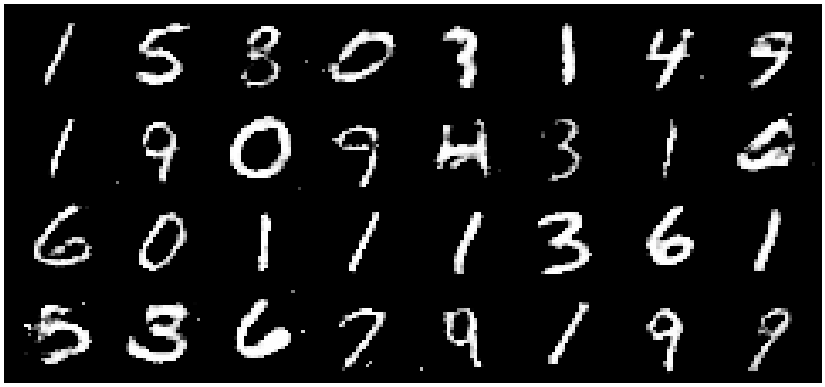
(a) Epoch 21



(a) Epoch 51



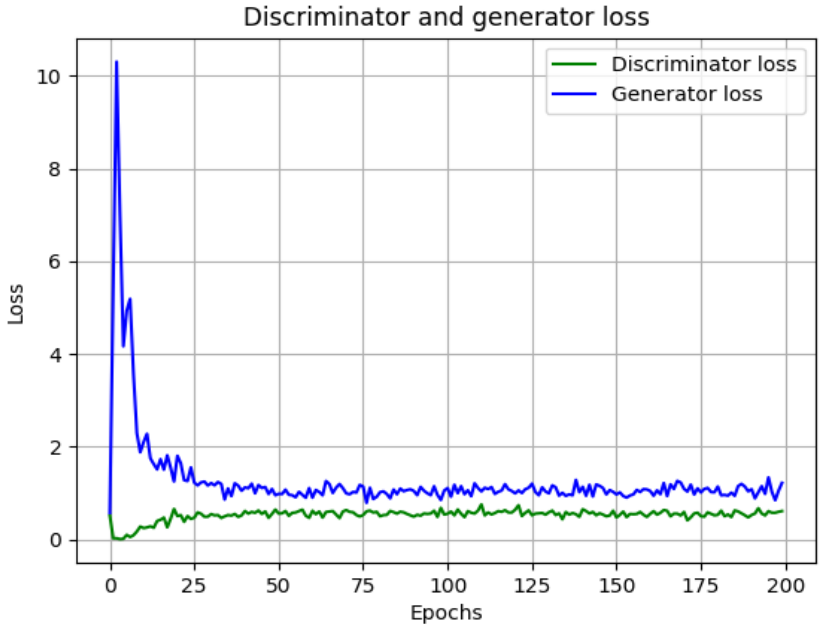
(a) Epoch 81



(a) Epoch 99

3.3.2 Graph of Epoch vs Loss

The synthetic images tend to get better and better after every epoch. This can be shown precisely by plotting the loss of the generator and discriminator on a graph over the 100 epochs:



(a) GAN loss vs epoch

3.4 Limitations of GAN

As of now, GANs are considered the best synthetic data generating algorithm having a high quality of produced images, however, they have certain limitations which hold them back:

1 Mode collapse: During the training, the generator may collapse to a setting where it always produces the same output. This is called mode collapse.

2 Hard to achieve Nash Equilibrium making the model unstable.

3.5 Uses of GAN

The best use case for GANs is when no of classes to be generated by a model is low and there is a requirement for high-quality images. GANs are computationally heavy and it thus takes a considerable amount of time to train the models. Another issue is its instability. Sometimes the discriminator can get so strong that the generator is unable to produce a decent-quality image. Other times, the discriminator can be so weak that it can accept even noise as real data. In both cases, the synthetic data generated is of poor quality.

4 Denoising Diffusion Probabilistic Models

4.1 Introduction to the Model

The way Denoising Diffusion Probabilistic models (DDPMs) function is by first destroying the picture in the forward phase and then creating it in the backward step of denoising. This is also referred to as a Markov Chain. In diffusion models, the latent space is of the same dimensionality as the input. The model's job is to forecast the noise that was introduced to each image. The diffusion model can take a random noise sample and produce an image that is highly similar to the collection of input photos by analyzing the noise added and eliminating it.

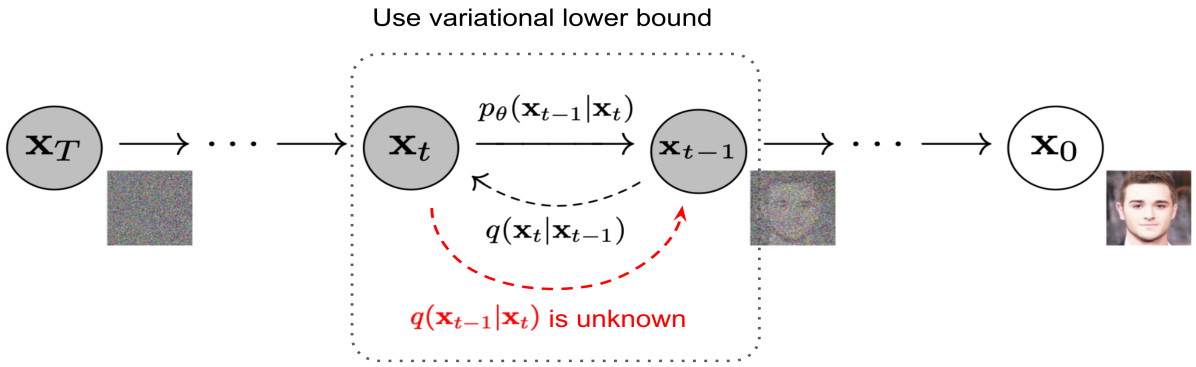
4.2 Working behind DDPM

It is challenging to produce a natural image from noise in a single step. Imagine creating an image in several smaller steps. sort of like how old-style photography would let an image develop on a Kodak film. As a result, each step's input and output should be easier for the neural network to process than going straight from pure noise to a final natural image.

This iterative generation concept has a drawback. What should the visuals be like in between? It should not be the case that throughout this iterative process, an image of a cat initially appears, and then the cat transforms into a human face, according to someone old enough (like me) to have expertise with old vintage photography. It seems sensible to impose the "gradual-ness" constraint on the in-between images. But how should it be expressed mathematically? Two processes are required:

1 A forward process, which turns a natural image into noise.

2 A backward process, which turns noise into a natural image.



(a) Epoch 91

Given a data point x_0 sampled from the real data distribution $q(x)(x_0 \epsilon q(x))$, one can define a forward diffusion process by adding noise. Specifically, at each step of the Markov chain, we add Gaussian noise with variance β_t to x_{t-1}, producing a new latent variable x_t with distribution $q(x_t|x_{t-1})$. This diffusion process can be formulated as follows:

$$q(x_t|x_{t-1}) = (x_t : N_t = \sqrt{1-\beta_t}x_{t-1}, \sigma_t = \beta_t I) \tag{15}$$

Since we are in the multi-dimensional scenario I, is the identity matrix, indicating that each dimension has the same standard deviation β_t. Thus, we can go in a closed form from the input data x_0 to x_t in a tractable way. Mathematically, this is the posterior probability and is defined as:

$$q(x_{1:T}|x_0) = \Pi_{t=1}^{T} q(x_t|x_{t-1}) \tag{16}$$

4.2.1 Reparameterization

If we define $\alpha_t = 1 - \beta_t, \bar{\alpha}_t = \Pi(s = 0)^t \alpha_s$ where $\epsilon_0, ..., \epsilon_{t-2}, \epsilon_{t-1}$ $N(0, I)$, one can use the reparameterization trick, thus to produce a sample x_t we can use the following distribution:

$$x_t \; q(x_t|x_0) = N(x_t; \sqrt{\hat{\alpha}}x_0, (1-\hat{\alpha}_t)I) \tag{17}$$

Since β_t is a hyper-parameter, we can pre-compute α_t and $\hat{\alpha}_t$ for all time-steps. This means that we sample noise at any time-step t and get x_t in one go. Hence, we can sample our latent variable x_t at any arbitrary timestamp.

4.2.2 Approximating the reverse process with a neural network

We approximate $q(x_{t-1}|x_t)$ with a parameterized model p_θ (e.g. a neural network). Since $q(x_{t-1}|x_t)$ will also be Gaussian, for small enough β_t, we can choose p_θ to be Gaussian and just parameterize

the mean and variance:

$$p_\theta(x_{t-1}|x_t) = N(x_{t-1}; \mu_\theta(x_t, t), \sigma_\theta(x_t, t)) \tag{18}$$

If we apply the reverse formula for all time-steps $p_\theta(x_{0:T})$, also called trajectory), we can go from x_T to the data distribution:

$$p_\theta(x_{0:T}) = p_\theta(x_T)\Pi_{t=1}^{T} p_\theta(x_{t-1}|x_t) \tag{19}$$

4.2.3 Loss function

The loss function can be expressed as:

$$L_t = E_{x_0,t,\epsilon}[\frac{1}{2||\sigma_\theta(x_t, t)||_2^2}||\epsilon_t - \epsilon_\theta(\sqrt{\hat{\alpha}_t}x_0 + \sqrt{1-\hat{\alpha}_t}\epsilon, t||^2)] \tag{20}$$

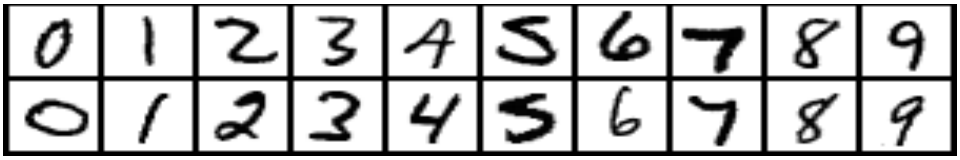
This effectively shows us that instead of predicting the mean of the distribution, the model will predict the noise ϵ at each timestamp t.

4.2.4 Implementation on MNIST data set

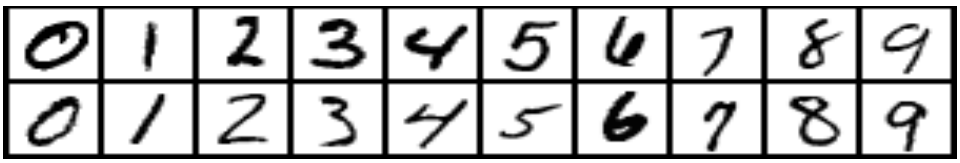
The MNIST data set is a collection of handwritten digits from 0 to 9. Our DDPM model will take the images as input and add noise to them at each step. Afterward, it will learn to identify the added noise and remove it to generate synthetic images.

4.2.5 Result

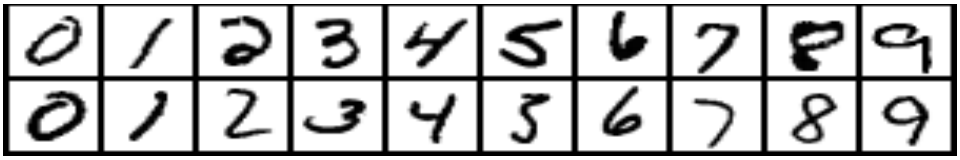
We train the model for 100 epochs and get the images synthesized by the model for every epoch. Here are displayed a few outputs:



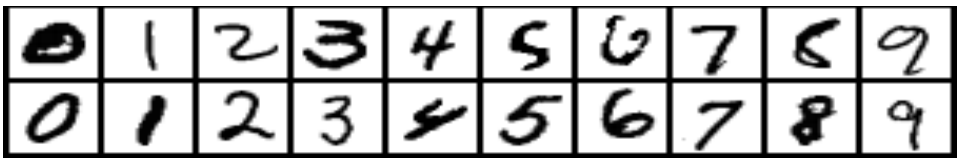
(a) Epoch 1



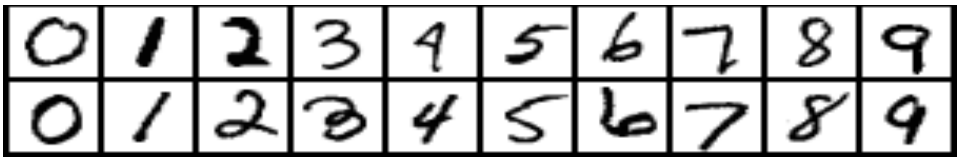
(a) Epoch 21



(a) Epoch 51



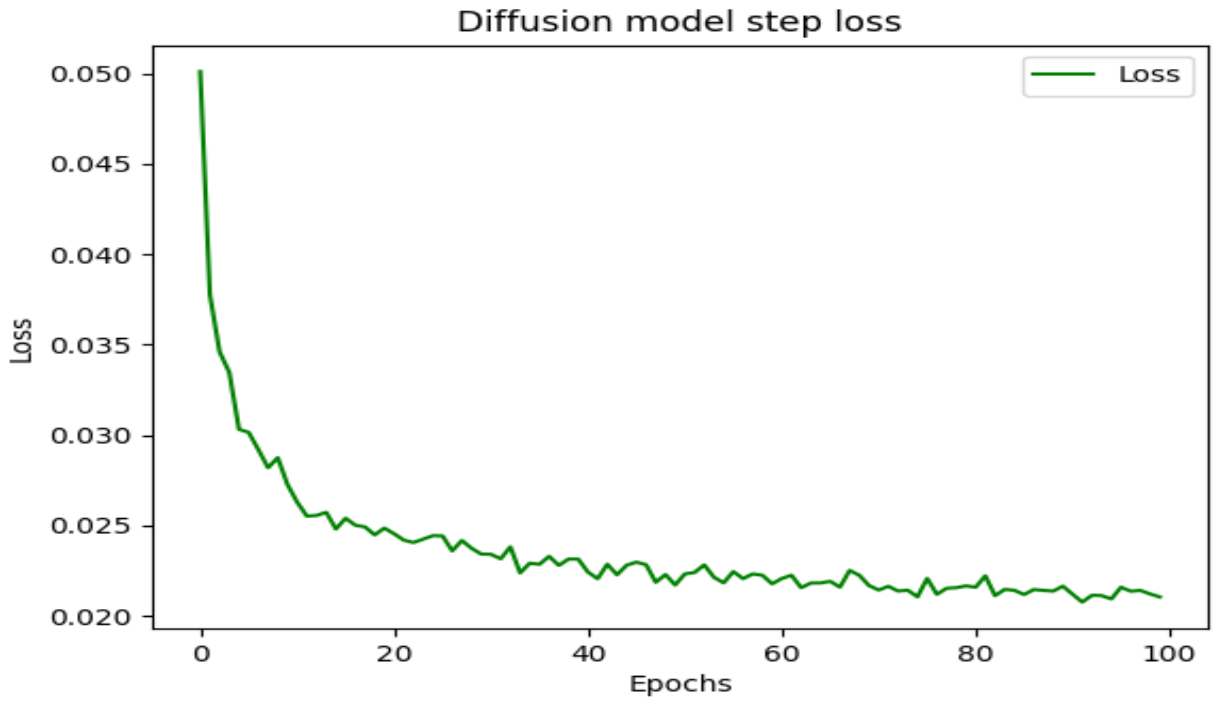
(a) Epoch 81



(a) Epoch 99

4.2.6 Graph of Epoch vs Loss

The synthetic images tend to get better and better after every epoch. This can be shown precisely by plotting the loss of DDPM on a graph over the 100 epochs:



(a) DDPM Loss vs epoch

4.3 Limitations of DDPM

The major limitation of DDPMs is the slow generation process, as at every step it deals with the data at the same size as the original input. Another limitation is the inability for dimension reduction, which both VAE and GAN have shown are capable of.

4.4 Uses of DDPM

DDPM are relatively recent models and have been used extensively for image generation as well as text-to-image generation and so on. They also find their uses in audio generation as audio signals can be noised and denoised in a similar fashion.

5 Discussion

The initial question of how to generate synthetic data has been thoroughly answered through the introduction of models such as these, but the new question which arises is the applications of these models. While GANs have been accepted universally as the best model for generating data based on the top quality of the images that are produced, the limitation of instability and modal

collapse reduces their viability. While several methods have been developed to reduce the chances of modal collapse, the variance in the generated data provided by the other two models outclasses GANs. Coming to VAE, the poor quality of the data produced makes it difficult to be used as a substitute for real-world data as a lot of applications require high quality to capture the essential features of the data. This brings us to DDPMs, which are by far the youngest models and have shown the capability to not only provide the variance to the degree of VAE but also provide the quality of GANs. The biggest issue with them is the amount of training time required, which is much greater when compared to the rest of the models. This trifecta of limitations and advantages between the three models makes it difficult choice to decide which model to choose in which situation.

The understanding of when to apply a given synthetic model and where to use it is crucial in furthering the study of synthetic data generation. Recently multiple websites have gotten popular which utilize these generative models at their core. ChatGPT the one that uses a generative pre-trained transformer who have GANs in its architecture to generate text-based responses while the Hugging Face project uses DDPM for the image-to-text or image-to-image generation. While data generation is one aspect, these models also find themselves useful when it comes to enhancing or altering images. VAE are good examples of compressing an image while DDPM models are sued for the colorization of old black and white photos.

Apart from image bases, text-based and audio-based data have also been generated using these models. A study on synthetic video generation is underway. While these models have been wonderful at producing data, there have been some protests against such models. Some image generation models, having been rained on the paintings or real-life people can generate paintings themselves which humans are unable to distinguish. This has led to questioning if the data generated is legitimate or just theft of existing images. While ethics have been questioned, there is no doubt that the data that is generated has reached the level of quality that it can easily replace real-world data. Synthetic data generators are tools that in the right hands have the potential to help us in dealing with data security and privacy.

6 Conclusion

The above three models shown in the report performed decently when it came to generating samples for the MNIST data sets of just over 100 epochs. While the quality of output and the training time differed, the samples produced were reasonably good. This showcases how these synthetic data generation models can be used to create a good data set. The models aren't limited to generating image-based data only, they can also generate text-based as well as audio-based data. Thus, generative models can provide a solution to the problem of lack of data and synthetic data seems to be the step toward the future development of ML, AI, and data science fields.

7 References

1 Generative Adversarial Nets - Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair†, Aaron Courville, Yoshua Bengio‡
`https://arxiv.org/pdf/1406.2661.pdf`

2 Denoising Diffusion Probabilistic Models - Jonathan Ho, Ajay Jain, Pieter Abbeel
`https://arxiv.org/pdf/2006.11239.pdf`

3 Auto-Encoding Variational Bayes - Diederik P. Kingma, Max Welling
`https://arxiv.org/pdf/2006.11239.pdf`