Comparative study between VAE and GAN

Shantanu Ghosh
4311-4360
University of Florida
shantanughosh@ufl.edu

Abstract

Unsupervised representational learning has been very popular where we represent each sample from a given dataset using it's latent representation. With the recent advancement in deep learning research and the high availability of faster GPUs, deep generative models dominated the domain of unsupervised representational learning. This has happened primarily due to the introduction of Variational autoencoders (VAE) and Generative Adversarial Networks (GAN). While both of these networks aim to learn the true data distribution, their learning algorithms are significantly different. Although significant progress has been made to these models since their introduction, this report compares the architectural difference of vanilla VAE and GAN through various experiments conducted on MNIST dataset.

1 Introduction

The whole machine learning domain is divided into two major categories - 1) Supervised learning and 2) Unsupervised learning. In supervised learning, the dataset is labelled and the goal of the learning algorithm is correctly predict the labels. In unsupervised learning, we don't have the access to the labels and the goal is to the true data distribution. With recent advancement in deep neural networks, we have seen tremendous growth in supervised learning research. With the availability of high computational power through the faster GPU's and deep neural networks, high prediction accuracies have been achieved in various supervised learning tasks such as object classification, object localization, speech recognition etc.

However supervised learning is expensive as labelling the data is time consuming and requires a lot of human effort. Various domains such as biomedical informatics or medical image analysis often suffer from limited labelled training samples. However, with large storage, there is a plethora of unlabelled data available. Using unsupervised representational learning, we can be able to learn the true data distribution by learning meaningful features, represent the data using the latent code to identify the true data generating process. Traditionally clustering techniques such as kmeans clustering or mixture models have been used in unsupervised learning. However with the introduction of deep generative models like Variational autoencoder(VAE)[4] and Generative adversarial networks(GAN) [2], the research domain of unsupervised representational learning has reached greater boundaries. Termed as "the most interesting idea in the last 10 years in Machine Learning", by the Turing award winner Yann LeCun, Since it's inception, Generative adversarial networks have been used in various domains to generate realistic images. In last couple of years, several variations of GANs have been published which improved the quality of generated images than the original vanilla GAN - two such variations are Conditional GAN [5] and InfoGAN [1] which we have been used in this study to augment the dataset and then, compare the accuracy of the classifier. The InfoGAN explicitly helps us to get the disentangled representation of features which helps in producing more realistic images and showed better performance than the original GAN. Similarly a wide range of VAEs have been developed like Conditional VAE [6], InfoVAE [7] etc which enriching this field of research.

The goal of both the VAE and GAN is to learn the true data distribution. In GAN, there are two neural networks, termed as generator and discriminator are trained in an adversarial manner aiming to achieve

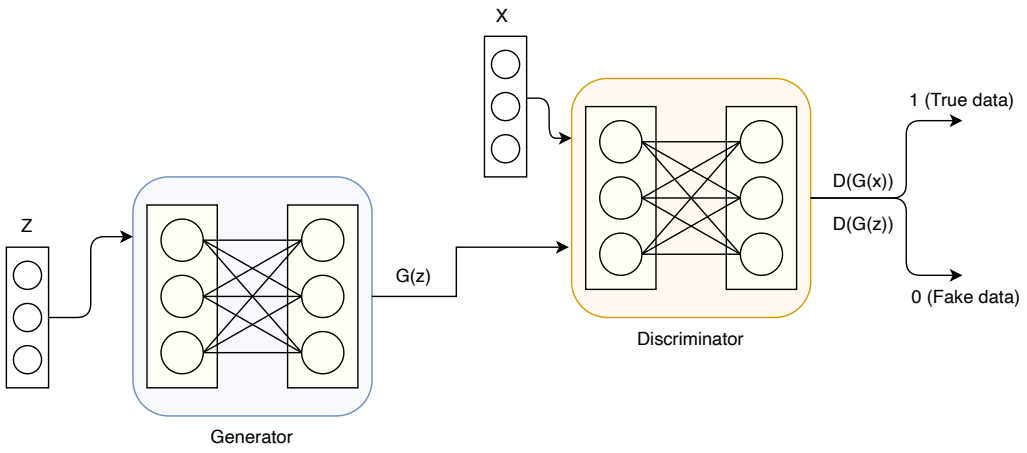

Figure 1: Block diagram of GAN.

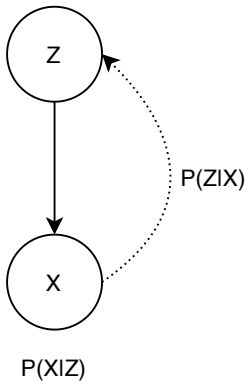

Figure 2: Underlying directed graphical model of VAE.

the Nash equilibrium [3]. Generator generates fake samples from a random noise, sampled from a prior distribution and pass it to the discriminator as a input. The second input to the discriminator is the samples from the true data distribution and the aim of the discriminator is to distinguish the real and the fake samples. The block diagram of GAN is explained in figure 1.

VAE assumes the data generating process as shown in the directed graphical in figure 2. Here z is a latent variable which is known as the data generating distribution sampled from a prior density $p(z)$ and $p(x|z)$ is known as data likelihood distribution and the goal is to calculate the posterior distribution $p(z|x)$. Now from Bayes theorem, we know that $p(z|x) = p(x|z)p(z)/p(x)$. Calculating this posterior $p(z|x)$ is a difficult task as the denominator – the marginal density $p(x) = \int p(x|z)p(z)dz$ is intractable, so we can not use Expectation Maximization (EM) algorithm. In VAE, we will try to approximate the posterior density $p(z|x)$ to a known distribution $q(z|x)$ using variational inference by minimizing the KL divergence between $p(z|x)$ and $q(z|x)$. To learn the $q(z|x)$, we will use a neural network which is the encoder of the VAE , parametrized by ϕ. Then the latent vector z will be sampled from the learned posterior $q(z|x)$ will be passed to another neural network which the decoder of VAE to learn a data likelihood density $p(x|z)$ parametrized by θ to reconstruct the original data sample. The block diagram of VAE is shown in figure 3.

In this study, a GAN and a VAE are designed with vanilla neural network and CNN and then the performance of these two models are compared with each other.

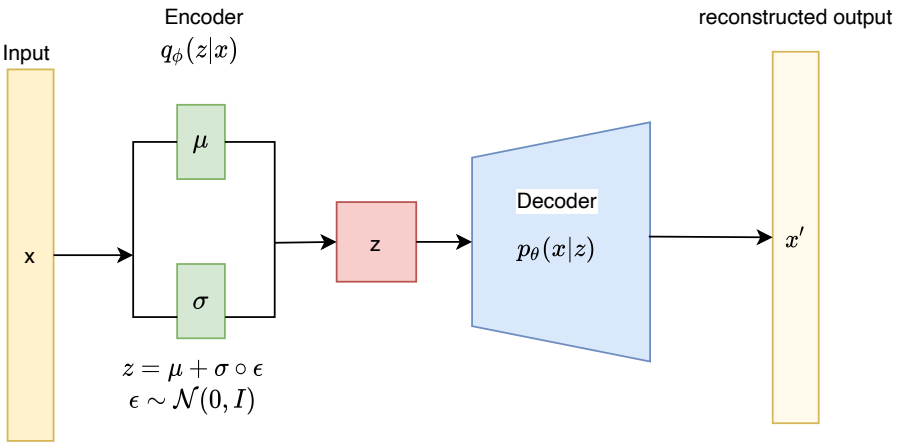

Figure 3: Block diagram of VAE.

2 Model Description

2.1 GAN

In standard GAN [2], there are two neural networks - generator (G) and discriminator (D) - which plays a two player minimax game, trained in the classical adversarial manner, such that the generator learns the true data distribution $\boldsymbol{p(x)}$ and tries to fool the discriminator by producing fake samples. The standard optimization function to train a GAN with data distribution as $\boldsymbol{p_{data}(x)}$ and random noise as $\boldsymbol{p_Z(z)}$ is as follows:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(\boldsymbol{z}))] \tag{1}$$

Here the Generator *G*, independent of the first term, is optimized with m mini-batches as follows

$$\min_G \phi_g(D,G) = \frac{1}{m}\sum_{i=1}^{m}\left[\log\left(1 - D\left(G\left(\boldsymbol{z^{(i)}}\right)\right)\right)\right] \tag{2}$$

The discriminator *D* in standard GAN whose primary objective is to differentiate the real images and fake generated images. The discriminator *D* is optimized with m mini-batches as follows:

$$\max_{D_G} \phi_d(D,G) = \frac{1}{m}\sum_{i=1}^{m}\left[\log D\left(\boldsymbol{x^{(i)}}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z^{(i)}}\right)\right)\right)\right]$$

2.1.1 GAN - loss function analysis

Achieving discriminator loss. Let for the i^{th} sample the true label and the predicted label is $y^{(i)}$ and $\hat{y}^{(i)}$ respectively. The aim of the discriminator is to classify the real samples coming from real distribution and the fake samples coming the generator. So the loss function of the discriminator using standard binary cross entropy loss for the i^{th} sample will be calculated as:

$$L^{(i)}(y,\hat{y}) = y^{(i)}\log\hat{y}^{(i)} + (1 - y^{(i)})\log(1 - \hat{y}^{(i)}) \tag{3}$$

For the real data samples coming from the true data distribution $p_{data}(x)$, $y^{(i)} = 1$ and $\hat{y}(i) = D(x^{(i)})$. Putting these values in the equation 3,

$$L^{(i)}(y,\hat{y}) = D(x^{(i)}) \tag{4}$$

For the fake data samples generated by the generator from the noise $z \sim p_Z(z)$, $y^{(i)} = 0$ and $\hat{y}(i) = D(G(z^{(i)}))$. Putting these values in the equation 3,

$$L^{(i)}(y,\hat{y}) = \log\left(1 - D(G(z^{(i)}))\right) \tag{5}$$

The objective of the discriminator is to maximise the loss in equations 4 and 5. So for all the samples $\{i\}_{k=1}^{n}$, the discriminator loss is

$$\max_{D} V(D,G) = \mathbb{E}_{x\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(\boldsymbol{z}))] \tag{6}$$

Achieving generator loss. For the fake data samples generated by the generator from the noise $z \sim p_Z(z)$, the aim of the generator is to generate samples such that $\hat{y}(i) = D(G(z^{(i)})) = 1$; at this point the value of $\log(1 - D(G(z^{(i)}))$ is $-\infty$, which is the minimum. So the aim of the generator is to minimise the equation 1. So, the optimization function of generator module of the GAN is,

$$\min_{G} V(D,G) = \mathbb{E}_{x\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(\boldsymbol{z}))] \tag{7}$$

Achieving optimal discriminator The generator takes noise samples $z \sim p_Z(z)$, values of taken by the random variable Z and transformed to $G(Z)$. Mathematically it is written as $X = G(Z)$. Now as G is one to one differentiable function, the density of the generated samples can be written as

$$p_g(x) = p_Z(G^{-1}(x)) \mid \frac{\partial G^{-1}(x)}{\partial x} \mid \tag{8}$$

where g denotes the generator.

The value of the optimal discriminator $D_G^* = \max_D V(D,G)$ using equation 6 can be obtained as follows:

$$\begin{aligned}
V(D,G) &= \mathbb{E}_{x\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(\boldsymbol{z}))] \\
&= \int_x p_{data}(x)\log(D(x))dx + \int_z p_Z(z)\log(1-D(G(z)))dz \\
&= \int_x p_{data}(x)\log(D(x))dx + \int_z p_Z(G^{-1}(x))\log(1-D(x)dG^{-1}(x) \\
&= \int_x p_{data}(x)\log(D(x))dx + \int_x p_Z(G^{-1}(x))\log(1-D(x))\frac{dG^{-1}(x)}{dx}dx \\
&= \int_x p_{data}(x)\log(D(x))dx + \int_x \left(p_Z(G^{-1}(x))\frac{dG^{-1}(x)}{dx}\right)\log\Big(1-D(x)\Big)dx \quad (9) \\
&= \int_x p_{data}(x)\log(D(x))dx + \int_x p_g(x)\log\Big(1-D(x)\Big)dx \quad (10) \\
&= \int_x \left[p_{data}(x)\log(D(x))dx + p_g(x)\log\Big(1-D(x)\Big)\right]dx \quad (11)
\end{aligned}$$

Equation 9 follows from equation 8. Now to get the optimal generator we need to maximise equation 11 w.r.t $D(x)$.

$$\frac{\partial V(D,G)}{\partial D(x)} = \frac{p_{data}(x)}{D(x)} - \frac{p_g(x)}{1-D(x)} = 0 \tag{12}$$

Solving equation 12, we get the optimal discriminator as,

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \tag{13}$$

Achieving optimal generator. To get the value of the optimal generator, we need to put $D_G^*(x)$ in equation 11 and minimise $V(D,G)$ w.r.t G,

$$V(D^*,G) = \int_x \left[p_{data}(x)\log\left(\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right)dx + p_g(x)\log\left(\frac{p_g(x)}{p_{data}(x)+p_g(x)}\right)\right]dx \tag{14}$$

Adding and subtracting $\log 2 * p_{data}(x)$ and $\log 2 * p_g(x)$in equation 14,

$$
\begin{aligned}
V(D^*, G) = \int_x \Bigg[& \Bigg\{ -\log 2 \Big(p_{data}(x) + p_g(x) \Big) \Bigg\} + \\
& p_{data}(x) \Bigg\{ \log 2 + \log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \Bigg\} + \\
& p_g(x) \Bigg\{ \log 2 + \log \frac{p_g(x)}{p_{data}(x) + p_g(x)} \Bigg\} \Bigg] dx \\
= & -2 \log 2 + \int_x \Bigg[p_{data}(x) \Bigg\{ \log \frac{p_{data}(x)}{\frac{p_{data}(x)+p_g(x)}{2}} \Bigg\} + p_g(x) \Bigg\{ \log \frac{p_g(x)}{\frac{p_{data}(x)+p_g(x)}{2}} \Bigg\} \Bigg] dx \\
= & -2 \log 2 + \Bigg[KL \Big(p_{data}(x) \Big|\Big| \frac{p_{data}(x) + p_g(x)}{2} \Big) + KL \Big(p_g(x) \Big|\Big| \frac{p_g(x) + p_g(x)}{2} \Big) \Bigg] \\
= & -\log 4 + \Bigg[2JSD \Big(p_{data}(x) \Big|\Big| p_g(x) \Big) \Bigg] \qquad (15)
\end{aligned}
$$

JSD term in equation 15 is the Jensen-Shannon divergence. Now to get the optimal generator, we have to minimise $V(D^*, G)$ w.r.t G and we denote the minimum generator as G^*_{min}. At G^*_{min}, the JSD term in euqation 15 becomes 0 and $G^*_{min} = -log4$ and and $p_{data}(x) = p_g(x)$. This is the global optimality condition discussed in GAN [2].

2.2 VAE

The loss function of VAE described in the figure 3 is as follows,

$$
\min_{\theta,\phi} -\mathbb{E}_{z \sim q_\phi(z|x)} \Big[\log p_\theta(x|z) \Big] + KL\big(q_\phi(z|x) \big|\big| p_\theta(z) \big) \qquad (16)
$$

Here the parameters of the encoder and decoder are represented by θ and ϕ.

2.2.1 VAE - loss function analysis

As discussed in the introduction, the posterior distribution $p(z|x)$ is intractable, so we need to approximate to a know distribution $q_\phi(z|x)$ by minimising the KL divergence between these two.

$$
\begin{aligned}
KL(q_\phi(z|x) \big|\big| p_\theta(z|x)) &= \sum_z q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(z|x)} \\
&= \mathbb{E}_{z \sim q_\phi(z|x)} \log \frac{q_\phi(z|x)}{p_\theta(z|x)} \\
&= \mathbb{E}_{z \sim q_\phi(z|x)} \Big[\log q_\phi(z|x) - \log p_\theta(z|x) \Big] \\
&= \mathbb{E}_{z \sim q_\phi(z|x)} \Big[\log q_\phi(z|x) - \log \frac{p_\theta(x|z) p(z)}{p_\theta(x)} \Big] \\
&= \mathbb{E}_{z \sim q_\phi(z|x)} \Big[\log q_\phi(z|x) - \log p_\theta(x|z) - \log p(z) \Big] + \log p_\theta(x)
\end{aligned} \qquad (17)
$$

Rearranging the terms in equation 17,

$$\begin{aligned}\log p_\theta(x) - KL(q_\phi(z|x)\big|\big|p_\theta(z|x)) &= \mathbb{E}_{z\sim q_\phi(z|x)}\Big[\log p_\theta(x|z)\Big] - \mathbb{E}_{z\sim q_\phi(z|x)}\Big[\log q_\phi(z|x) - \log p_\theta(z)\Big] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}\Big[\log p_\theta(x|z)\Big] - \sum_z \Big[q_\phi(z|x)\log\frac{q_\phi(z|x)}{p_\theta(z)}\Big] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}\Big[\log p_\theta(x|z)\Big] - KL\Big(q_\phi(z|x)\big|\big|p_\theta(z)\Big)\end{aligned} \tag{18}$$

Rearranging the terms in equation 18,

$$\log p_\theta(x) = KL(q_\phi(z|x)\big|\big|p_\theta(z|x)) + \mathbb{E}_{z\sim q_\phi(z|x)}\Big[\log p_\theta(x|z)\Big] - KL\Big(q_\phi(z|x)\big|\big|p_\theta(z)\Big) \tag{19}$$

As $KL(q_\phi(z|x)||p_\theta(z|x)) \geq 0$, $\log p_\theta(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - KL(q_\phi(z|x)||p_\theta(z))$.

The term $\mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - KL(q_\phi(z|x)\big|\big|p_\theta(z))$ is known as Evidence lower bound or ELBO loss. The objective of VAE is to minimise to approximate to a known to a posterior distribution $q_\phi(z|x)$ by minimising $KL(q_\phi(z|x)\big|\big|p_\theta(z|x))$. As the LHS of the equation 19 is independent of ϕ, we can achieve that by maximising the ELBO loss or minimising the -ELBO loss yeilding equation 16. The first term of the ELBO loss is known as the reconstruction loss between the original input and the reconstructed output. The second term of the ELBO loss, i.e the KL divergence is the regularization term which forces the approximated posterior $q_\phi(z|x)$ to be as close the the prior of the latent density $p_Z(z)$

Optimization of the VAE loss. The VAE loss function 16 can be rewritten as

$$\min_{\theta,\phi} L(\theta,\phi) \tag{20}$$

where $L(\theta,\phi) = -\mathbb{E}_{z\sim q_\phi(z|x)}\Big[\log p_\theta(x|z)\Big] + KL\big(q_\phi(z|x)\big|\big|p_\theta(z)\big)$.

To get the optimal θ and ϕ, for all the samples, we need to minimise L w.r.t θ and ϕ in alternate fashion. Getting the optimal θ^*, for i^{th} example,

$$\begin{aligned}\theta_i^* &= \nabla_\theta(L(\theta,\phi)) \\ &= \nabla_\theta(-\mathbb{E}_{z\sim q_\phi(z|x)}\big[\log p_\theta(x|z)\big]) \\ &= \nabla_\theta\Big(-\sum_{l=1}^{L}\log p_\theta(x|z^{(l)})\Big) \qquad (21) \\ &= -\frac{1}{L}\sum_{l=1}^{L}\nabla_\theta \log p_\theta(x|z^{(l)}) \qquad (22)\end{aligned}$$

Equation 21 follows from the Monte Carlo approximation of the Expectation operator and l is the dimension of $z^{(l)} \sim q_\phi(z|x)$

Getting the optimal ϕ^*, for i^{th} example,

$$\begin{aligned}\phi_i^* &= \nabla_\phi(L(\theta,\phi)) \\ &= \nabla_\phi(-\mathbb{E}_{z\sim q_\phi(z|x)}\big[\log p_\theta(x|z)\big]) + KL\big(q_\phi(z|x)\big|\big|p_\theta(z)\big) \qquad (23)\end{aligned}$$

Here we can not take the derivative w.r.t ϕ $z^l \sim q_\phi(z|x)$. Here we have to use re-parametrization trick. Let $\epsilon \sim p(\epsilon)$, where $p(\epsilon) = \mathcal{N}(0,1)$ and $z = g_\phi(\epsilon,x)$. So z can be written as $z = \mu_\phi(x) + \epsilon \odot \sum^{1/2}(x)$

Table 1: Configuration of generator used in GAN

Layer	Layer type	Depth	Kernel	Stride	Padding
1	Transposed Convolution	100	3*3	1 -	
2	Transposed Convolution	512	3*3	2	0
3	Transposed Convolution	256	5*5	1	1
4	Transposed Convolution	128	4*4	2	1
5	Transposed Convolution	64	4*4	2	1

$$
\begin{aligned}
\phi_i^* &= \nabla_\phi(L(\theta,\phi)) \\
&= \nabla_\phi(-\mathbb{E}_{z\sim q_\phi(z|x)}\big[\log p_\theta(x|z)\big]) + KL\big(q_\phi(z|x)\big|\big|p_\theta(z)\big) \\
&= \nabla_\phi(-\mathbb{E}_{z^{(l)}\sim p(\epsilon)}\big[\log p_\theta(x|z^{(l)})\big]) + KL\big(q_\phi(z|x)\big|\big|p_\theta(z)\big) \\
&= -\frac{1}{L}\Big[\sum_{l=1}^{L}\log p_\theta(x|z^{(l)}) + KL\big(q_\phi(z|x)\big|\big|p_\theta(z)\big)\Big]
\end{aligned} \tag{24}
$$

3 Experimental setup

3.1 Dataset

We have used the MNIST dataset to perform our experiments where each image is of 28x28 pixels. The dataset consists of hand written digits from 0 to 9, uniformly distributed. The training dataset consists of 60000 images and the test dataset has 10000 images. This project has been developed in MacBook Pro i7, 16GB memory mounting Mac OS Big Sur.

3.2 Deep learning modules' settings

3.2.1 GAN

3.2.2 Generator

In GAN, the generator generates realistic images to fool the discriminator. To accomplish this, a random noise vector(z) of size 100 sampled from $\mathcal{N}(0,1)$ has been passed as input to the Generator. We used 5 **transposed convolution** layers detailed in table 1 as the configuration of the generator. The Generator generated an MNIST image of size 28 x 28. We have used **Relu** as the activation function used for the Generator for each of all the transposed convolution hidden layer. For the output layer of the Generator, we have use **tanh** activation function. Between the convolution layers, except the last one, a batch normalization layer has been used.

3.2.3 Discriminator

The role of the discriminator is to detect the real images from the MNIST dataset and fake images generated by the Generator. We used couple of convolution layers in the discriminator to learn the features from the images. The input of the discriminator is an image from the MNIST dataset, so the input layer has a dimension of 784. We have used 2 **convolution** layers and 3 **fully-connected** hidden layers as the configuration of the discriminator detailed in table 2. Also after the convolution layers, a maxpool layer has been used. We have used **Relu** as the activation function used for the discriminator for each of the fully connected layers and the convolution layers. As we have developed the GAN in Pytorch, we did not use the sigmoid activation in the output layer of the discriminator as the BCE(binary-cross-entropy) loss explicitly use normalise the output between 0 and 1 to give us the probability of the 2 classes - real and fake.

3.2.4 VAE

In VAE, the dimension of the image is reduced to it's latent representations. The details of the network architecture of the VAE is specified in the table 3. Here all the layers are fully connected layers and

Table 2: Configuration of discriminator used in GAN

Layer	Layer type	Depth	Kernel	Stride
1	Convolution	1	5*5	1
3	Max Pool	-	2*2	2
4	Convolution	6	5*5	1
5	Max Pool	-	2*2	2
6	Fully Connected1 + RELU	-	-	-
7	Fully Connected2 + RELU	-	-	-
8	Fully Connected3 + RELU	-	-	-

Table 3: Configuration of VAE

Layer	Network	Layer type	Depth	Kernel	Stride	Padding
1	Encoder	Convolution1	1	4*4	2	1
3	Encoder	Convolution2	64	4*4	2	1
4	Encoder	Linear + RELU	-	-	-	-
5	Encoder	Linear + RELU	-	-	-	-
6	Decoder	Linear + RELU	-	-	-	-
7	Decoder	Convolution1	128	4*4	2	1
8	Decoder	Convolution2	64	4*4	2	1

we used **ReLU** as the activation function for each of the layers of the encoder and decoder in the VAE.

3.3 Training the GAN and VAE

To train, the GAN and VAE, the different hyperparamters are specified in table 4 and 5 respectively. At the start of this project, I have used only fully connected layers in GAN and VAE to generate images, later I used CNN layers to generate more relistic images. The images generated after training the GAN and VAE using vanilla network figure 5 and figure 4. The images generated after training the GAN and VAE using deep CNN network figure 6. The loss vs iteration plot for GAN model is given in figure 7

4 Code

The code is attached in the Final.zip file. In-order to run the code, the **VAE_main.py** inside VAE folder and **GAN_Manager.py** inside GAN folder files should be executed separately to see the outputs of VAE and GAN respectively. The code will also be uploaded shortly at the ***Github repository***.

4.1 GAN

python3 GAN_Manager.py

4.2 VAE

python3 VAE_main.py

5 Analysis and Conclusion

Last time using vanilla neural network, the images generated by VAE was more realistic than GAN. However, this time using CNN, the images generated by GAN are more realistic and crisp. From the reconstructed output of the VAE, we can infer that the generated images of the VAE are similar to the original digits, but the quality of the images need to be improved. I have trained the GAN and

Table 4: Hyperparameters used in training the GAN

Hyperparameters	**Value**
Epochs	1000
Optimizer	Adam with default setting
Learning rate	0.002
Loss	BCE loss
Batch size	128

Table 5: Hyperparameters used in training the VAE

Hyperparameters	**Value**
Epochs	1000
Optimizer	Adam with default setting
Learning rate	1e-3
Loss	BCE loss + KL Divergence
Batch size	128

VAE for only 100 and 1000 epochs in my local laptop which is also another reseaon that the images generated like that. While I tried for longer epochs, but somehow python in my laptop got crashed. However, I got the access to the hi-pergator because of another course recently, so before the final write-up, I will run these models to the hi-pergator using the GPU's for longer epoch. My current code-base is already configured to use GPU. To develop this study, I take help explicitly from the following two resources:
1)***VAE***
2) ***GAN*** Theoretically, GAN gives superior performance thn VAE as at the start the noise distribution is very different from the true data distribution, so there is hardly any overlap between this two. As VAE uses KL divergence in it's loss, the KL divergence between two distribution without any overlap is ∞. However, GAN uses JSD, so at the very beginning, the JSD between the noise distribution and the data distribution is finite, in-fact the value is $\log 2$. Still, GAN often suffers from vanishing gradient and mode collapse problem and research is still going on the rectify these issues.

References

[1] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[3] Charles A Holt and Alvin E Roth. The nash equilibrium: A perspective. *Proceedings of the National Academy of Sciences*, 101(12):3999–4002, 2004.

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[5] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[6] Artidoro Pagnoni, Kevin Liu, and Shangyan Li. Conditional variational autoencoder for neural machine translation. *arXiv preprint arXiv:1812.04405*, 2018.

[7] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Infovae: Information maximizing variational autoencoders. *arXiv preprint arXiv:1706.02262*, 2017.

Figure 4: Generated images after training the VAE using vanilla neural network. From left to right, images generated after 10 and 20 epochs.

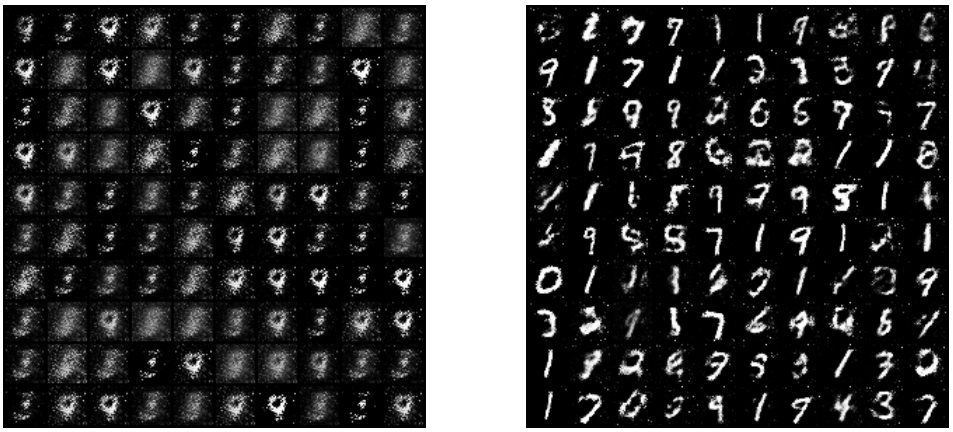

Figure 5: Original (left) vs reconstructed (right) images after training vanilla GAN

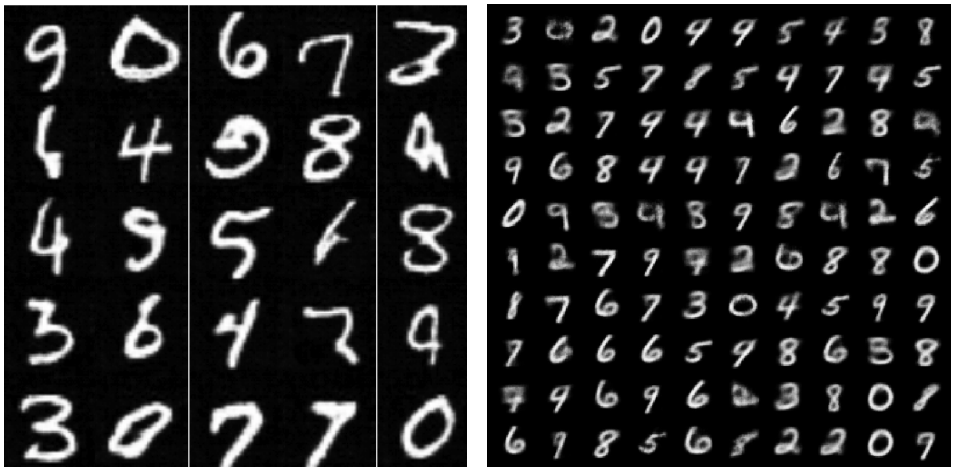

Figure 6: Generated images after training the GAN (left) and VAE (right) using Deep CNN.

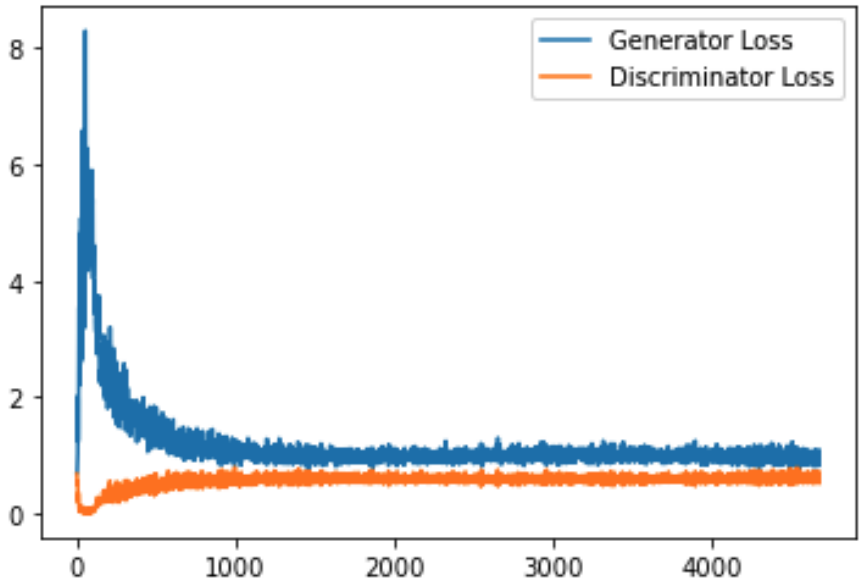

Figure 7: loss vs iteration plot for GAN.