Efficient Learning of GAN

ARNAB SAHA

M.Tech CS

Supervisor: Dr. N.R. Pal

A thesis submitted in fulfilment of
the requirements for the degree of
M.Tech

ECSU
Indian Statistical Institute
Kolkata

10 July 2020

M.Tech(CS) DISSERTATION THESIS COMPLETION CERTIFICATE

Student : Arnab Saha

Topic : Efficient learning of GAN

Supervisor : Dr. N.R. Pal

This is to certify that the thesis titled "***Efficient learning of GAN***" submitted by **Arnab Saha** in partial fulfillment for the award of the degree of **Master of Technology** is a bonafide record of work carried out by him under my supervision. The thesis has fulfilled all the requirements as per the regulations of this Institute and, in my opinion, has reached the standard needed for submission. The results contained in this thesis have not been submitted to any other university for the award of any degree or diploma.

_____

Prof. N.R. Pal

Supervisor

Dedication

To my parents and close ones, without your help and encouragement it wouldn't have been possible.

Acknowledgements

I would like to thank my dissertation supervisor Dr. N.R. PAL for agreeing to guide me and for helping me to undertake work in the topic. Without his continuous guide and support this wouldn't have been possible.

I am also very much thankful to Suvro Da, Suchismita Di, Manish, Laltu and all my lab mates and seniors of The CI lab of Indian Statistical Institute , Kolkata for helping me though out the project with their valuable time and suggestions.

Abstract

GAN or Generative Adversarial Network is a combination of two deep Neural Networks in which one network acts as a generator where the other acts as a discriminator which differentiate between real and generated fake samples. There are different variants of GAN. For every variant of GAN we have to train two deep neural networks simultaneously and the hardest part about GAN is it's training. During training many GAN models suffer various major problems like non-convergence, mode collapse, high sensitivity to the selection of hyper-parameters and vanishing gradient. In this project we tried to address the problem **Mode-collapse**. Where the generator generates only one or limited variants of samples irrespective of the inputs.

Contents

CHAPTER 1

Introduction

Generative Adversarial Networks (GANs) are a powerful class of neural networks that are used for unsupervised learning, which was developed and introduced by *Ian J. Goodfellow in 2014* [**1**] .GAN is for creating, like drawing a portrait or composing a symphony. GAN consists of *3 parts* [**2**].
Generative : To learn a generative model, which describes how data is generated in terms of a probabilistic model.

Adversarial : Training of model is done in an adversarial setting.

Network : Use of deep neural networks to train the model.
From the structural point of view GAN consist of 2 parts .

Generator : *Generates fake samples from noise data. The outputs of generator are fed to the discriminator. It tries to fool the discriminator*[**3**].

Discriminator : *The input of the discriminator is either from generator's output or from the real data. It's job is to discriminate between real and fake data*[**3**].
A generator alone will just create random noise. Conceptually, the discriminator in GAN provides guidance to the generator on what images to create.

As we can see in the figure 1.1 in next page, input or random noise are being fed to generator's input and it creates an image which is known as a fake sample or synthetic image. Then this fake sample and training sample from real data both are being fed to discriminator and the discriminator produce an output $D(X)$. The discriminator tries to make this $D(x)$ as close as

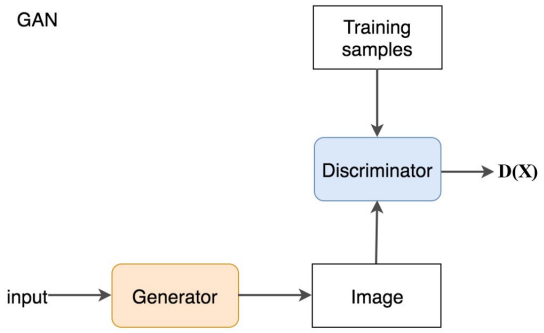


FIGURE 1.1. Schematic diagram of GAN [**5**]

possible to 1 for the real data and as close as possible to 0 for the fake data. And depending upon this discriminator's output GAN trains itself through back-propagation. This is the simplest explanation of how the training of GAN is done. But the training process of GAN is really not that simple.

Now to develop a clear idea of the training process we will go through some simple equations .The discriminator outputs a value $D(x)$ indicating the chance that x is a real image. Our objective is to maximize the chance to recognize real images as real and generated images as fake. i.e. *the maximum likelihood of the observed data* [**3**]. To measure the loss, we use cross-entropy as in most Deep Learning: $plog(q)$. For real image, p (the true label for real images) equals to 1. For generated images, we reverse the label (i.e. one minus label). So the objective becomes:

$$\max_D V(D) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1 - D(G(Z)))] \text{ ... (1)}$$

On the generator side, its objective function wants the model to generate images with the highest possible value of $D(x)$ to fool the discriminator.

$$\min_G V(G) = E_{z\sim p_z(z)}[log(1 - D(G(Z)))] \text{ ... (2)}$$

And combining both the equations 1 and 2 we get the objective function of GAN. Which is illustrated below,

$$\min_G \max_D V(G,D) = E_{x\sim p_{data}(x)}[logD(x)]+E_{z\sim p_z(z)}[log(1-D(G(Z)))]...(3)$$

As we can see in the above objective function the discriminator tries to produce $D(x) = 1$ where x is taken from real data distribution and $D(G(Z) = 0$ where Z is sampled from normal or uniform distribution, So that the value of $V(G, D)$ can be 0, which is the maximum value of $V(G, D)$. So we can see that the discriminator tries to maximize the objective function $V(G, D)$ by only controlling it's parameter D.

On the other hand the generator tries to fool the discriminator by making $D(G(Z) = 1$. So that the value of $V(G, D)$ can be $-\infty$, which is the minimum value of the objective function $V(G, D)$. So we can see that the generator tries to minimize the objective function $V(G, D)$ by only controlling it's parameter G. That's why we often called GAN a *min-max game* [**4**].

CHAPTER 2

Training of GAN Model

After the objective function of GAN is defined the model is trained by alternating gradient descent [**3**]. We fix the generator model's parameters and iterate once to perform gradient descent on the discriminator parameters using the real and generated images. Then we fix the discriminator's parameters and train the generator for another single iteration. We continue this alternate gradient descent until the generator produces good quality images. The below figure 2.1 shows the data flow and gradients in backpropagation.

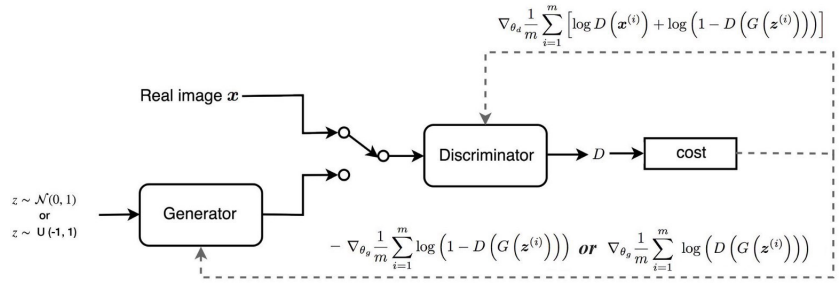


FIGURE 2.1. Back propagation in GAN [**3**]

The pseudo code below summarizes everything and we can see how a GAN is trained.

Algorithm

for number of training iteration **do**

- Sample minibatch of m noise samples $\{z_1, ..., z_m\}$ from noise prior $p_g(z)$.
- Sample minibatch of m examples $\{x_1, ..., x_m\}$ from data generating distribution $p_{data}(x)$.
- Update the discriminator by ascending stochastic gradient :

$$\nabla_{\theta d} \frac{1}{m} \sum_{i=1}^{m} [logD(x^{(i)}) + log(1 - D(G(z^{(i)})))]$$

- Sample minibatch of m noise samples $\{z_1, ..., z_m\}$ from noise prior $p_g(z)$.
- Update the generator by descending stochastic gradient:

$$\nabla_{\theta g} [\frac{1}{m} \sum_{i=1}^{m} [log(1 - D(G(z^{(i)})))].$$

end for

CHAPTER 3

Mode Collapse

Mode collapse is said to happen when the generator generates a limited diversity of samples, or even the same sample, regardless of the input.A complete collapse is not common but a partial collapse happens often [**5**]. For example consider the MNIST dataset. When the dataset is feed to GAN model it can be seen that after some training it for some considerable iterations the generator starts to produce some particular digits with very high quality instead of producing all the variety or mode of the digits i.e. from 0 to 9.

Now the question is why does this Mode Collapse occur in the first place? To understand the reason behind it we have to recall the objective of GAN generator (eq 2).

$$\min_G V(G) = E_{z\sim p_z(z)}[log(1 - D(G(Z)))]$$

The stochastic gradient of the generator is given by,

$$\nabla_{\theta g} \frac{1}{m} \sum_{i=1}^{m} log(1 - D(G(z^{(i)})))]$$

The objective of generator is to create images that can fool the discriminator most. Now we consider an extreme case where we train the generator extensively and update it parameters without updating of discriminator parameter(i.e. we stopped the discriminator training). Then the generator training will converge to find the optimal image x^* which is most realistic from discriminator perspective. In this time the generator's output will be independent of generator's input. Whatever z fed to generator's input it will always create the x^*. So, x^* can be defined as,

$$x^* = argmax_x D(x)$$

And this way the mode collapses in a single point x^*. The gradient associated with z approaches to zero. And there is nothing in the objective function that explicitly forces the generator to generate different samples given the input.

When we restart the training in the discriminator, the most effective way to detect generated images is to detect this single mode. Since the generator desensitizes the impact of z already, the gradient from the discriminator is likely push the single point around for the next most vulnerable mode. This is not hard to find. The generator produces such an imbalance of modes in training that it deteriorates its capability to detect others. Now, both networks are over fitted to exploit short-term opponent weakness . *This turns into a cat-and-mouse game and the model will not converge*[**5**].

In the diagram below, in figure 3.1 the *Unroll GAN* [**6**] manages to produce all 8 expected modes of data. The second row shows another GAN for which the mode collapses and rotates to another mode when the discriminator catches up.

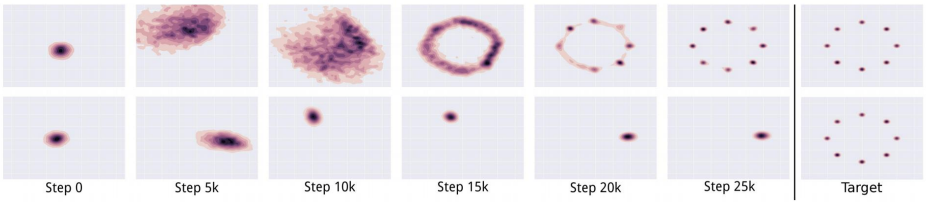


FIGURE 3.1. Mode Collapse example [**6**]

CHAPTER 4

Our Approach to solve Mode Collapse

As we have seen earlier that we don't use any particular method or objective that explicitly forces the generator to generate different samples or more diverse samples in case of traditional GAN models. So in our work, we use an explicit method to force the generator to generate more diverse samples.
In our work, we tried to minimize the distance between the real data and generated data using Maximum Mean Discrepancy(MMD) method.

4.1 Theory

Assume we are given data $\{x_i\}_{i=1}^n$, where $x_i \in X$ and $x_i \sim P_X$. If we are interested in sampling from P_X , it is not necessary to estimate the density of P_X . Instead, Generative Adversarial Network (GAN) trains a generator $g_{\theta g}$ parameterized by θ_g to transform samples $z \sim P_Z$, where $z \in Z$, into $g_{\theta g}(z) \sim P_\theta$ such that $P_\theta \approx P_X$. To measure the similarity between P_X and P_θ via their samples $\{x\}_{i=1}^n$ and $\{g_\theta(z_j)\}_{j=1}^n$ during the training, we train the discriminator D_{θ_d} parameterized by θ_d for help. The learning is done by playing a two-player game, where D_{θ_d} tries to distinguish x_i and $g_{\theta g}(z_j)$ while $g_{\theta g}$ aims to confuse $D_{\theta d}$ by generating $g_{\theta g}(z_j)$ similar to x_i.
On the other hand, distinguishing two distributions by finite samples is known as Two-Sample Test in statistics. One way to conduct two-sample test is via kernel maximum mean discrepancy (MMD) [7]. In general, MMD

is defined by the idea of representing distances between distributions as distances between mean embedding of features. Given two distributions P and Q, and a kernel k, the square of MMD distance is defined as,

$$M_k(P,Q) = \|\mu_P - \mu_Q\|^2_{\mathcal{H}} = \mathbb{E}_P[k(x,x')] - 2\,\mathbb{E}_{P,Q}[k(x,y)] + \mathbb{E}_Q[k(y,y')].$$

where $\mathcal{H}$ is a Reproducing Kernel Hilbert Space.
and k is a characteristic kernel, such as Gaussian kernel.
In practice we use finite samples from distributions to estimate MMD distance [7]. Given $X = \{x_1, ..., x_n\} \sim P$ and $Y = \{y_1, ..., y_n\}$, one estimator of $M_k(P,Q)$ is,

$$\hat{M}_k(X,Y) = \frac{1}{\binom{n}{2}} \sum_{i \neq i'} k(x_i, x'_i) - \frac{2}{\binom{n}{2}} \sum_{i \neq j} k(x_i, y_j) + \frac{1}{\binom{n}{2}} \sum_{j \neq j'} k(y_j, y'_j).$$

4.2 Algorithm

So, we can see from theory section, that MMD is a measure of distance between two distributions. The main idea of our approach is to minimize the distance between the real data and fake or generated data, so that our generator can learn all the diversities embedded in the real data and therefore can produce all the modes of the input dataset.
So we minimize the MMD of two distributions, but instead of calculating the MMD of real data and fake data we extract our real and fake data from our discriminator's second last layer. After that we calculate their MMD and used this measure as a regularizer to train our generator.
So basically, our algorithm is almost same as the conventional algorithm of training a GAN but with the addition of this new regularizer in generator's objective function. So, our new algorithm looks like below,

Algorithm

for number of training iteration **do**

- Sample minibatch of m noise samples $\{z_1, ..., z_m\}$ from noise prior $p_g(z)$.
- Sample minibatch of m examples $\{x_1, ..., x_m\}$ from data generating distribution $p_{data}(x)$.
- Update the discriminator by ascending stochastic gradient :

$$\nabla_{\theta d} \frac{1}{m} \sum_{i=1}^{m} [logD(x^{(i)}) + log(1 - D(G(z^{(i)})))]$$

- Extract the latent space representation of real data samples and fake data samples from discriminator's second last hidden layer. Say, $\hat{X} \implies$ real data's latent space representation and $\hat{Z} \implies$ fake data's latent space representation.

- Calculate MMD of two distribution $\hat{Z}$ and $\hat{X}$, say $M_k(\hat{Z}, \hat{X})$.
- Sample minibatch of m noise samples $\{z_1, ..., z_m\}$ from noise prior $p_g(z)$.
- Update the generator by descending stochastic gradient with added regularizer term:

$$\nabla_{\theta g} [\frac{1}{m} \sum_{i=1}^{m} [log(1 - D(G(z^{(i)})))] + \eta M_k(\hat{Z}, \hat{X})].$$

end for

4.3 Experiment

We train our regularized GAN to generate images on MNIST dataset which contains 28*28, single channel, 60K training images. All the images are generated from a random normal noise with $\mu = 0$ and $\sigma = 1$.

Network architecture : We used the conventional architecture of Deep Convolution GAN (DCGAN) in our experiment. The network architecture is shown below in figure 4.1,

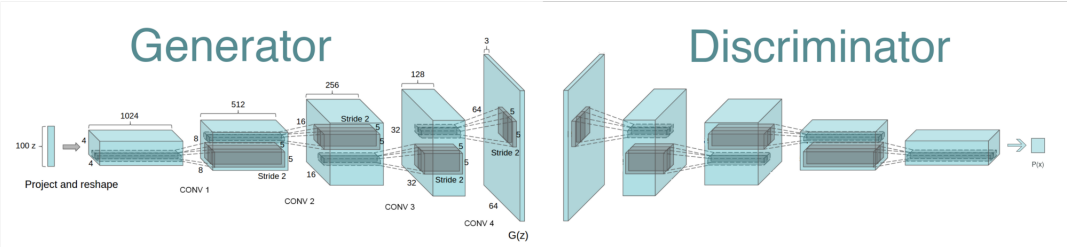


FIGURE 4.1. DCGAN Architecture [**8**]

Kernel : In this experiment we use Gaussian kernel as our characteristics kernel. So in our case, $k(x, x') = exp(\|x - x'\|^2)$. We used a mixture of Gaussian kernels with σ = [1,2,4,8,16] and calculate MMD based on these kernels [**7**].

Hyper-parameters : We used adam optimizer with $\alpha = 0.0002$ and $\beta = 0.5$ to optimize the generator and discriminator. We checked for regularizer co-efficient = 5, 10 and 15. We also tuned the latent space dimension between 10 , 20 and 30. The minibatch size is set to 100 for all the cases.

4.4 Result

In Figure 4.2, 4.3, 4.4 and 4.5 we show our experiment result for regularizer co-efficient $\eta = 0, 15, 10, 15$. Here, $\eta = 0$ implies we train the network without any regularizer. The reason we showed our result from epoch 25 is because, we observed that the DCGAN model without the regularizer tends to generate mode collapsed images continuously after epoch 23 or 24 in most of the cases. We stopped our training after 30 epochs. So we compare that result with our regularized GANs using different regularizer co-efficient and one can observe that the results are better when we use the MMD as regularizer.

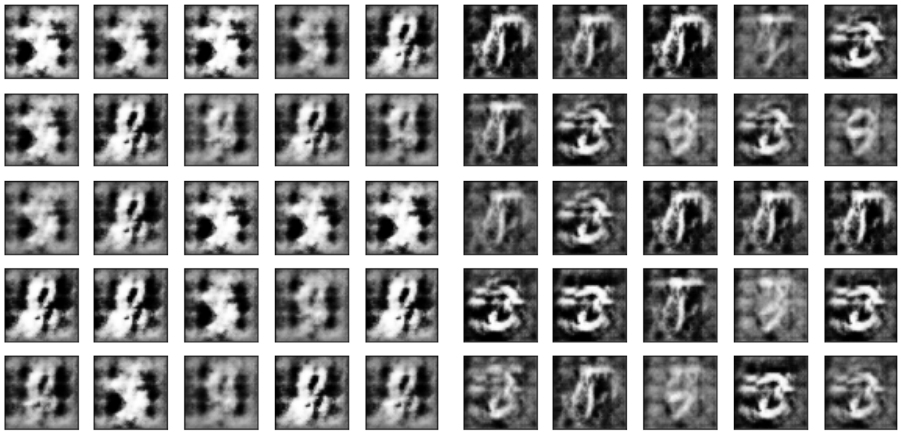
To make the comparison fair and transparent we use same initial condition for each of the regularizer co-efficient value. Here same initial conditions refers to the fact that, we use same random noise to train the generator as well as to produce fake samples from the generator in each of the cases.

In MNIST dataset there are total 60K training samples and in each iteration we train mini batches of real data and fake data of size 100. So the number of mini batches = 60000/100 = 600.

Now for each mini batch we need random noise (a 4D tensor) of dimension [(mini batch size),1,1,100] to feed the generator. So we produce a list of random numbers of size 600 and then used each element of that list as a seed value to generate a noise vector for each 600 mini batches. In this way, we ensure that the set of noise vectors for each regularizer co-efficient value is exactly the same.

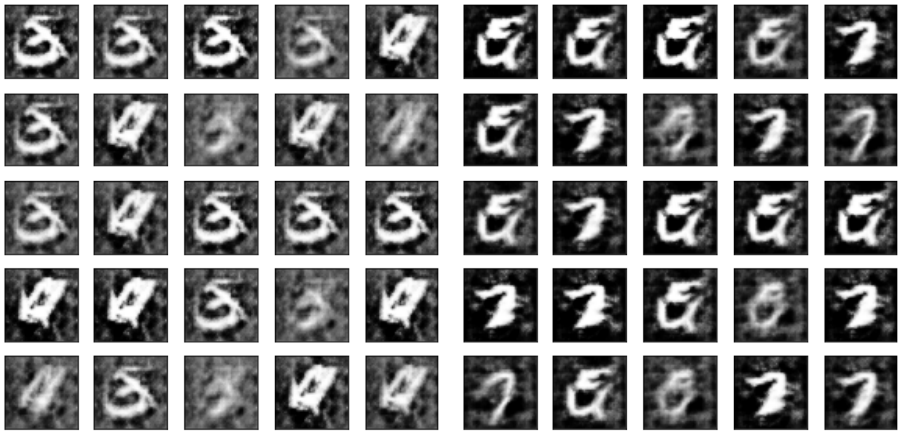
During the generation process we fixed a constant seed value and then generates random noise of dimension [(mini batch size),1,1,100]. This way, we ensure that after being trained for 1 epoch our generator used the exactly same random noise to generate fake samples. And this random noise is also identical for each regularizer co-efficient value.

FIGURE 4.2. Regularizer co-efficient $\eta = 0$



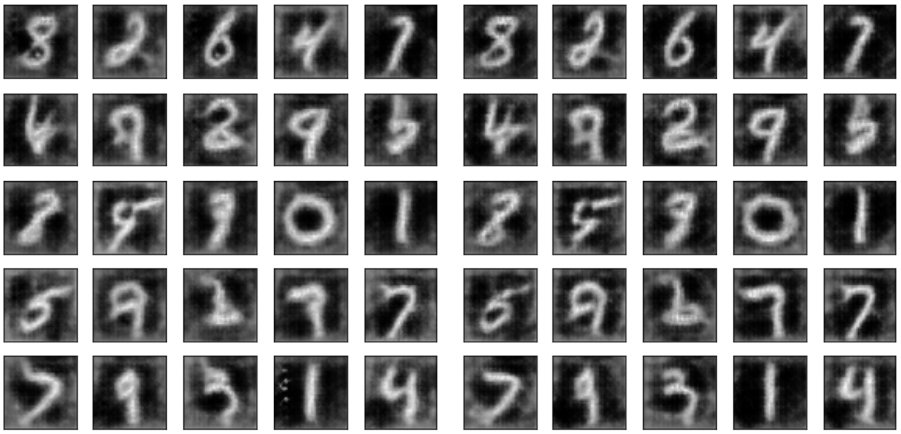
Epoch 25 Epoch 26

Epoch 27 Epoch 28

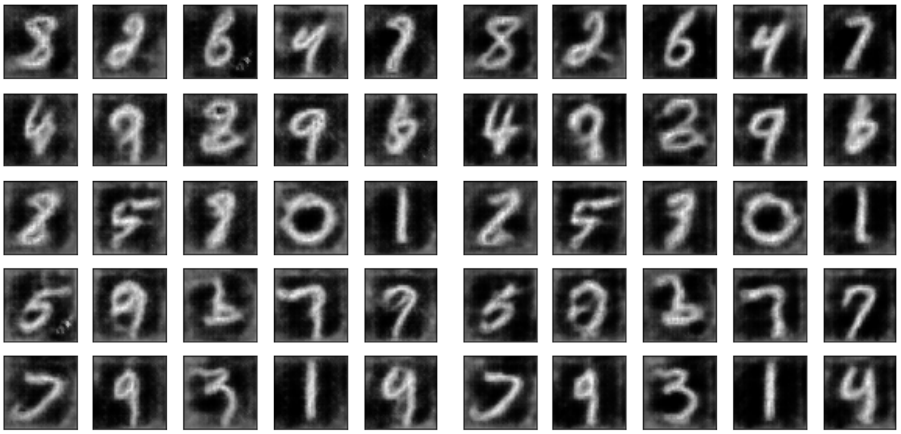


Epoch 29 Epoch 30

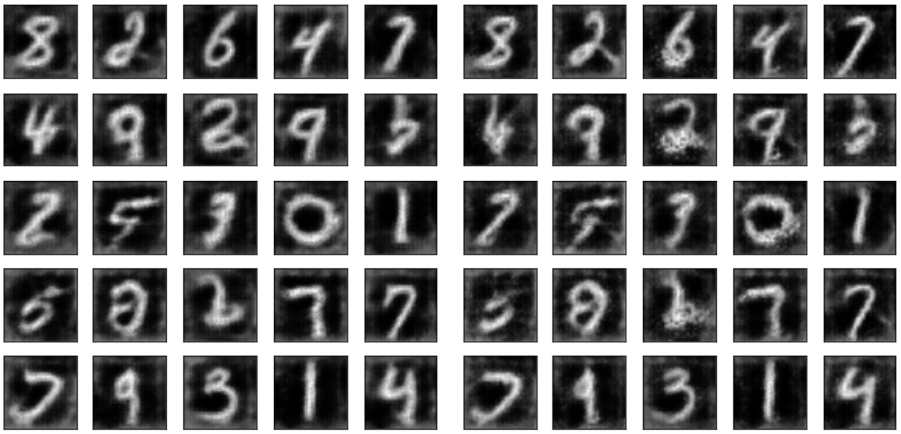
FIGURE 4.3. Regularizer co-efficient $\eta = 5$



Epoch 25 Epoch 26

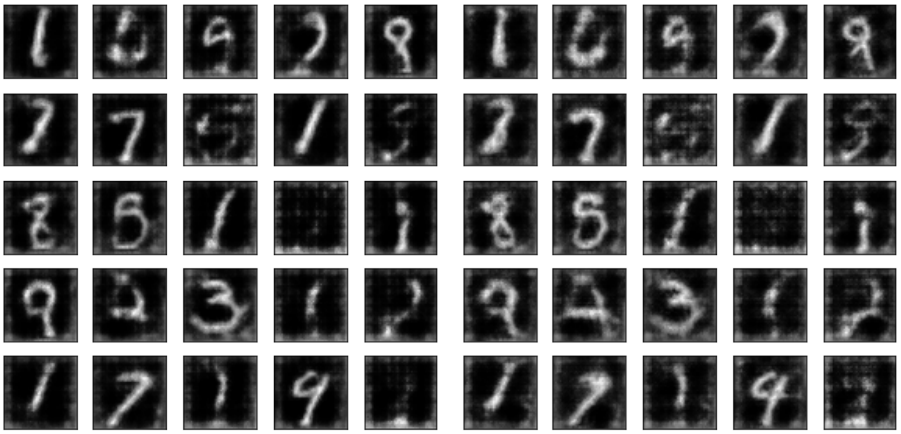


Epoch 27 Epoch 28

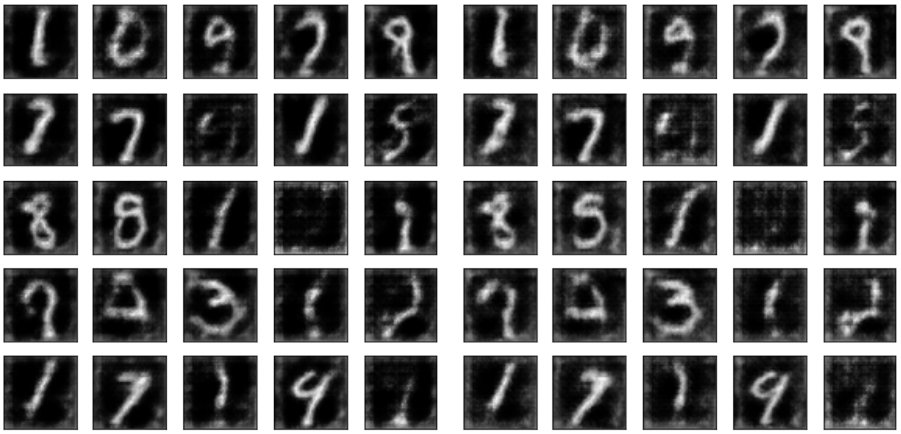


Epoch 29 Epoch 30

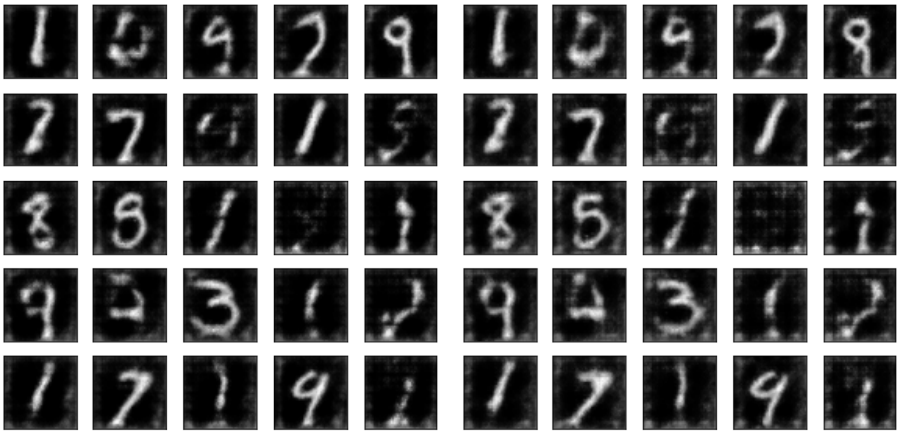
FIGURE 4.4. Regularizer co-efficient $\eta = 10$



Epoch 25 Epoch 26



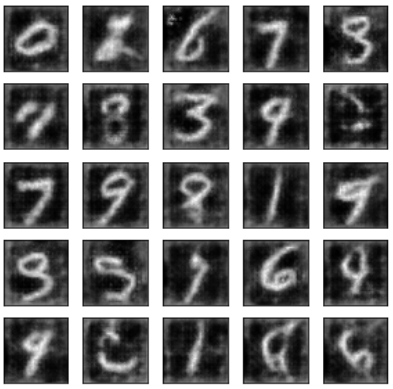
Epoch 27 Epoch 28



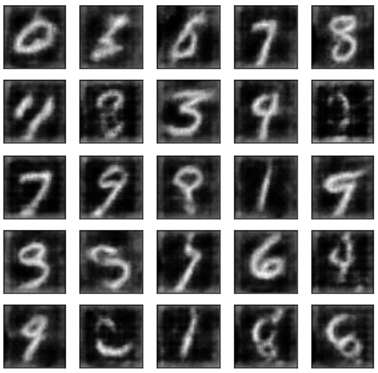
Epoch 29 Epoch 30

FIGURE 4.5. Regularizer co-efficient $\eta = 15$

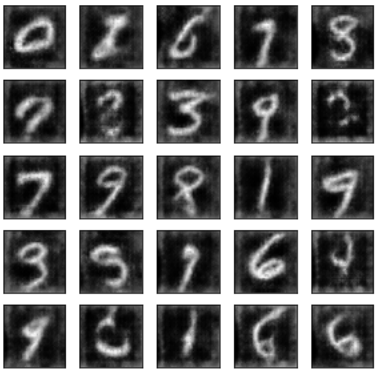
Epoch 25



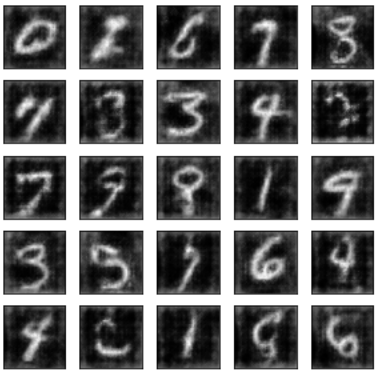
Epoch 26



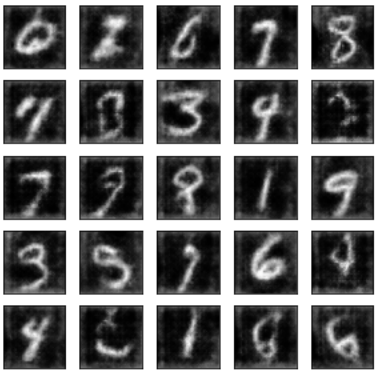
Epoch 27



Epoch 28



Epoch 29



Epoch 30

4.5 Result Analysis

As we can see from the result, for $\eta = 0$ (i.e. when we didn't use MMD as a regularizer in generator's cost function) from epoch 25 mode collapse is occurring again and again up to the last epoch of our training.
Now to match the scale of MMD with generator's loss we used 3 regularizer co-efficient 5, 10 and 15.
We observed that, when we used $\eta = 5$, then the generation quality is somewhat good compare to $\eta = 10, 15$. But if we use this value multiple times to train our generator, in some runs we observed that, mode collapse occurred and stayed up to 2 to 3 epochs and then the generator again started to produce mode-collapse free images.
When we used $\eta = 15$, then the generation quality dropped in some cases.
In case of $\eta = 10$ the mode collapse didn't occur and the generation quality is also better than the case where $\eta = 15$.
From the result we can conclude that, if we use $\eta = 10$ then the value of MMD can be brought to the same scale as generator's loss and we observed there is a fair trade off between the generation quality and mode collapsed free images. In case of $\eta = 5$ the generation is biased towards the generation quality and for $\eta = 15$ the generation is biased towards mode collapsed free images.

4.6 Conclusion

Mode collapse is an architectural problem of GAN. During the training process of the generator if the generator's parameter is updated extensively the generator tries to find the optimal image which is most realistic from discriminator's perspective and it starts to produce that image irrespective of the input and collapsed on that single mode. Now when the discriminator detects this single mode the generator tries to find the next most vulnerable mode from discriminator's perspective. And this way, it turns in to a cat and

mouse game and the process repeats again and again.

Now to overcome this problem we introduce MMD in our work. MMD or Maximum Mean Discrepancy is a measure between two probability distributions. In the conventional cost function of GAN there is no explicit term that force the generator to product more diverse samples. We used MMD as a regularizer in our generator's cost function and minimized it with the generator's cost using Adam optimizer so that the generator can produce more diverse samples and can avoid mode collapse. And our results are reflecting our goal of this work. Though due to this pandemic situation it was not possible to access GPU or high speed internet connectivity. So we only ran our test on MNIST data and couldn't run it on large datasets like CelebA dataset.

In future this solution can further be extended to produce more diverse samples with very high quality generation and this can lead to the solution of GAN's convergence problem which is another major problem of GAN.

References

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

[2] Rahul Roy *Generative Adversarial Network (GAN)* GFG Article link

[3] Jonathan Hui *GAN — What is Generative Adversarial Networks GAN?* Medium Article link

[4] Ian J. Goodfellow *NIPS 2016 Tutorial: Generative Adversarial Networks* https://arxiv.org/abs/1701.00160

[5] Jonathan Hui *GAN — Why it is so hard to train Generative Adversarial Networks!* Medium Article link

[6] Luke Metz, Ben Poole, David Pfau, Jascha Sohl-Dickstein *Unrolled Generative Adversarial Networks* https://arxiv.org/abs/1611.02163v4

[7] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, Barnabás Póczos *MMD GAN: Towards Deeper Understanding of Moment Matching Network*. In the Proceedings of *Thirty-first Annual Conference on Neural Information Processing Systems (NIPS 2017)* https://arxiv.org/abs/1705.08584v3

[8] Alec Radford, Luke Metz, Soumith Chintala *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks* Under review as a *conference paper at ICLR 2016* https://arxiv.org/abs/1511.06434