CS 5100 FINAL REPORT

TEXT TO IMAGE SYNTHESIS USING GENERATIVE ADVERSARIAL NETWORKS

ANANT VASANT SHANBHAG, POOJA KRISHNANATH SHANBHAG

DEC 2019

1. INTRODUCTION

The motivation behind this project is to understand automatic synthesis of images from a text description. Through this project, we wanted to explore architectures that could help us achieve our task of generating images. In recent years, powerful neural network architectures like Generative Adversarial networks (proposed by Ian Goodfellow et al) are generating highly compelling images. Meanwhile, recurrent neural network architecture has been developed to learn discriminative text feature representations. In this project, we have developed a similar architecture and GAN formulation to generate images. We are trying to implement the paper Generative Adversarial Text to Image Synthesis [1] by using a simpler data set (MNIST dataset).

2. PROBLEM STATEMENT

The exact problem this project intends to solve is to generate images of handwritten digits given a textual description. The image generated will be in MNIST format. This can be seen as a reverse Optical Character Recognition (OCR). Given a text description “one” it should be able to generate an image of handwritten digit “1” in MNIST format.

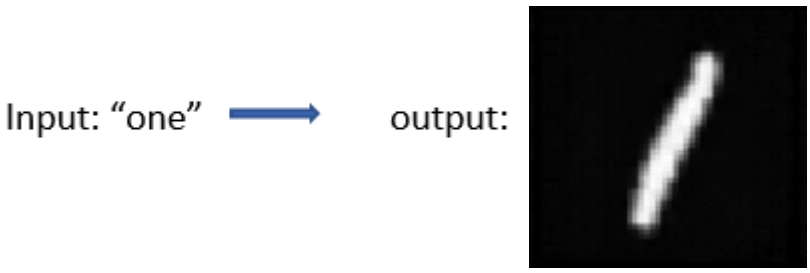


Figure 1: Problem Statement

The main goal is to train a generator model to learn the true distribution of the real data so that it generates a new unique unseen image every time. This generated image should look like it has come from the distribution of real training data. Similarly, it should generate images for texts from “zero” to “nine”.

In future, this problem statement can be extended to support complex text descriptions. We can try to generate image of handwritten number which will be the result of a mathematical expression. For example: Given a textual description like “2 * 5”, it should generate an image of handwritten number “10”.

3. DATASET

We are using LeCunn’s MNIST (Modified National Institute of Standards and Technology database) dataset. This consists of 60,000 black and white images of handwritten digits, each with size 28x28 pixels^2.

4. METHODS

Our approach was to learn the concept of Generative Adversarial Networks (GAN) and then implement a Vanilla GAN to generate images from random noise. With success in basic image generation we intend to iteratively enhance this implementation to achieve text to Image synthesis.

4.1. GENERATIVE ADVERSARIAL NETWORKS

The basic idea of GANs is to set up a minimax game between two players. One of them is called the generator G. The generator creates samples that are intended to come from the same distribution as the training data. The other player is the discriminator D. The discriminator tries to distinguish real training data from synthetic images, and the generator tries to fool the discriminator [2]. Concretely, D and G play the following game on V (D, G):

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

The two players in the game are represented by two functions, each of which is differentiable both with respect to its inputs and with respect to its parameters. The discriminator is a function D that takes 'x' as input and uses $\theta^{(D)}$ as parameters. The generator is defined by a function G that takes 'z' as input and uses $\theta^{(G)}$ as parameters. $J^{(D)}$ and $J^{(G)}$ are cost functions of discriminator and generator respectively.[2]

Both players have cost functions that are defined in terms of both players' parameters. The discriminator wishes to minimize $J^{(D)}(\theta^{(D)}, \theta^{(G)})$ by controlling only $\theta^{(D)}$. The generator wishes to minimize $J^{(G)}(\theta^{(D)}, \theta^{(G)})$ by controlling only $\theta^{(G)}$. Because each player's cost depends on the other player's parameters, but each player cannot control the other player's parameters, this scenario is most straightforward to describe as a game rather than as an optimization problem. The solution to an optimization problem is a (local) minimum, a point in parameter space where all neighboring points have greater or equal cost. The solution to a game is a Nash equilibrium.[2]

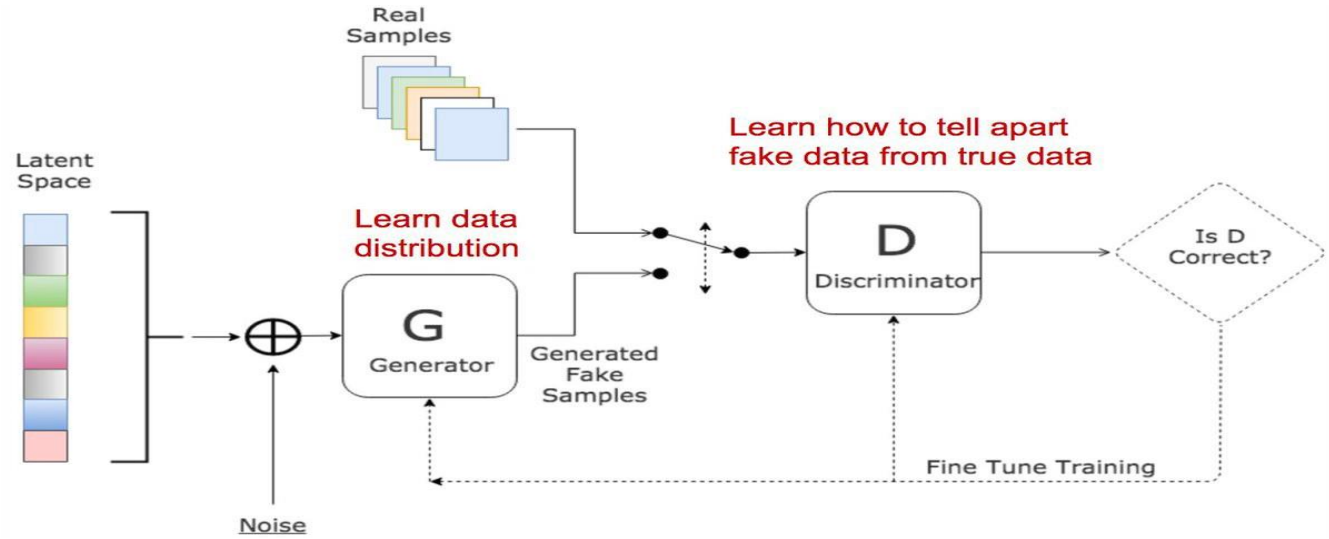


Figure 2: GAN Architecture

4.1.1. DISCRIMINATOR

The discriminator learns using traditional supervised learning techniques. It is trained on real and fake images with target as true and false respectively. The goal of the discriminator is to output the probability that its input is real rather than fake, under the assumption that half of the inputs it is ever shown are real and half are fake. Here discriminator tries to make D(x) to be near 1(i.e. probability of real image) and D(G(z)) to be near 0(i.e. probability of fake image).

4.1.2. GENERATOR

The inputs z to the generator are randomly sampled from the model's prior over the latent variables. The discriminator then receives input G(z), a fake sample created by the generator. The discriminator strives to make D(G(z)) approach 0 while the generator strives to make the same quantity approach 1. If both models have enough capacity, then the Nash equilibrium of this game corresponds to the G(z) being drawn from the same distribution as the training data, and D(x) = 1/2 for all x. [3]

4.2. IMPLEMENTATIONS

4.2.1. VANILLA GAN

This is the simplest type of GAN. Here the Generator and Discriminator are simple fully connected neural networks.

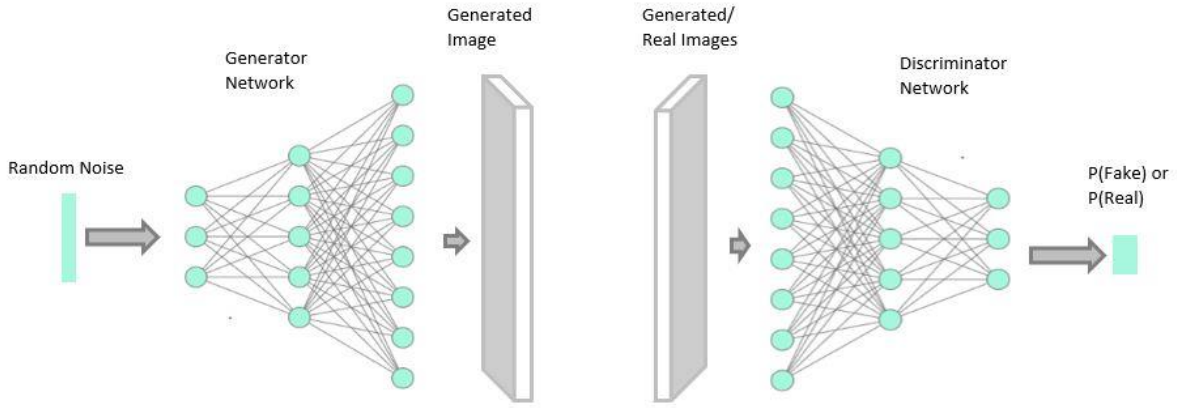


Figure 3: Vanilla GAN Architecture

Steps:

- We started with preprocessing the MNIST data by normalizing it from [0,255] to [-1,1]. We used this step to improve the training dynamics. This is a trick while training a GAN.
- We picked 'n' pre-processed images via random sampling for training the discriminator. The generator generated images by applying weights on the noise samples.
- Discriminator: It is a fully connected neural network with 3 hidden layers. Each hidden layer is followed by LeakyReLU for non-linearity, and a dropout layer to prevent overfitting. Then sigmoid function is added to the output of the last layer. This gives the value in the open-range (0, 1).
  Input to the discriminator: A flatten image 28*28 = 784 pixels
  Output from the discriminator: Probability of image belonging to synthetic or real data distribution.
- Generator: It is a fully connected neural network with 3 hidden layers. Each hidden layer is followed by LeakyReLU for non-linearity. TanH activation function is added to the output layer. This maps the output value to (-1,1) range which represents a generated image.
- Hyperparameter used for discriminator and generator:

Optimizer = ADAM optimizer. Learning rate = 0.0002.
Loss = Binary Cross entropy loss (BCE loss).

- Training process: The training process consists of simultaneous SGD. On each step, two minibatches are sampled: a minibatch of x values from the dataset and a minibatch of z values drawn from the model's prior over latent variables. Then two gradient steps are made simultaneously: one updating $\theta^{(D)}$ to reduce $J^{(D)}$ and one updating $\theta^{(G)}$ to reduce $J^{(G)}$.
- Training Discriminator: The discriminator takes a real data and fake data, and trains itself with labels true (1) and false (0) respectively. Discriminator tries to minimize the loss(D(z)) and predict the fake as accurate as possible.
- Training Generator: The generator takes fake data (noisy data generated using random pixel values) as input and takes label as true (1). It gets the prediction for fake data from the discriminator. The generator now tries to learn new weights by minimizing the error of generator i.e. train to prove the fake data is real. This in turn tries to maximize D(G(z)).

The fully connected neural networks are not capable of capturing correlation between different object parts in a scene. This entails the need for a deep neural net architecture which will solve this problem.

4.2.2. DEEP CONVOLUTIONAL GAN (DCGAN)

This is a successful implementation of GAN where generator and discriminators are deep convolutional neural networks. They generate highly compelling images. The adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminators.[4] The correlation between different object parts in a scene is captured by CNN whereas fully connected layers failed at this.

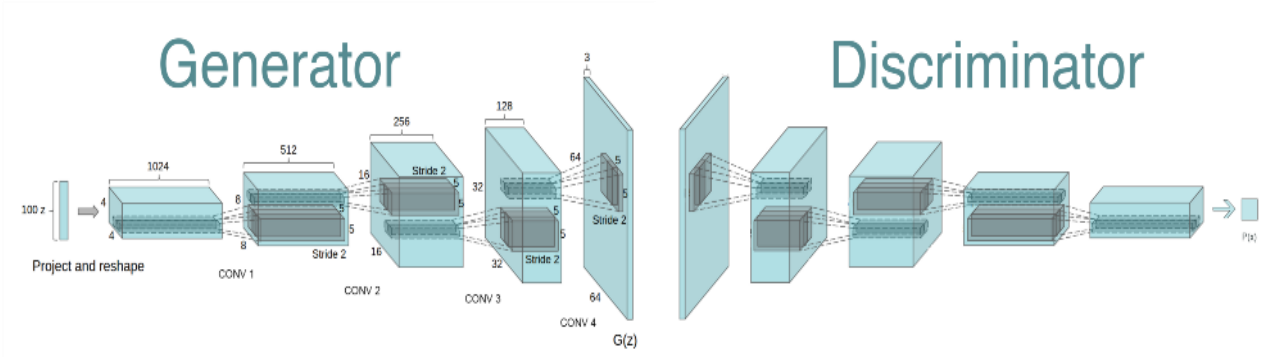


Figure 4: DCGAN Architecture

Steps:

- From the implementation of Vanilla GAN, we removed the fully connected layers and replaced those with Convolutional Neural Networks.
- Discriminator: It is composed of 4 convolutional layers as shown in the figure above. It uses batch normalization in most of the layers. The first layer of the discriminator is not batch normalized, so that the model can learn the correct mean and scale of the data distribution.
- Generator: It is composed of 4 convolutional layers as shown in the figure above. It uses batch normalization in most of the layers. The last layer of the generator is not batch normalized, so that the model can learn the correct mean and scale of the data distribution.
- This architecture contains neither pooling nor "unpooling" layers. When the generator needs to increase the spatial dimension of the representation it uses transposed convolution with a stride greater than 1.
- Training: The training process of generator and discriminators remain same as that of Vanilla GAN.

4.2.3. TEXT TO IMAGE SYNTHESIS (GAN-CLS ALGORITHM) USING DCGAN

This implementation tries to achieve image generation for given text description. Text description is represented as embeddings and adversarial pair is trained with these embeddings along with image vectors. In this implementation, we are trying to generate handwritten numbers given a single word description of the number. For example, a "nine" would generate a handwritten number 9 in MNIST image format.

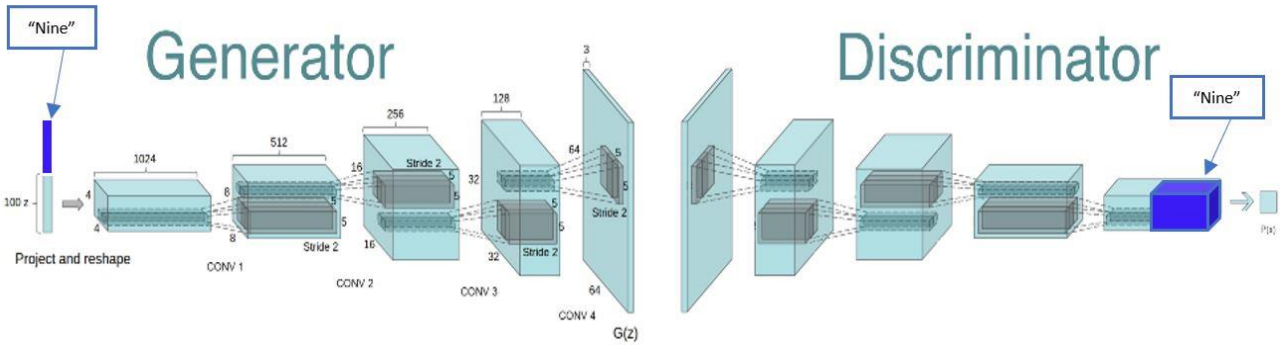


Figure 4: Text to Image Synthesis Architecture

The generator network is denoted G: $R^Z \times R^T \rightarrow R^D$, the discriminator as D: $R^D \times R^T \rightarrow \{0, 1\}$, where T is the dimension of the text description embedding, D is the dimension of the image, and Z is the dimension of the noise input to G. In this implementation, the architecture is designed using GAN-CLS algorithm and DCGANs. [1]

Matching-aware discriminator (GAN-CLS):

Algorithm 1 GAN-CLS training algorithm with step size α, using minibatch SGD for simplicity.

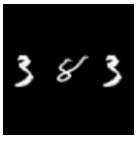
Input: minibatch images x, matching text t, mismatching $\hat{t}$, number of training batch steps S
for $n = 1$ **to** S **do**
  $h \leftarrow \varphi(t)$ {Encode matching text description}
  $\hat{h} \leftarrow \varphi(\hat{t})$ {Encode mis-matching text description}
  $z \sim \mathcal{N}(0, 1)^Z$ {Draw sample of random noise}
  $\hat{x} \leftarrow G(z, h)$ {Forward through generator}
  $s_r \leftarrow D(x, h)$ {real image, right text}
  $s_w \leftarrow D(x, \hat{h})$ {real image, wrong text}
  $s_f \leftarrow D(\hat{x}, h)$ {fake image, right text}
  $\mathcal{L}_D \leftarrow \log(s_r) + (\log(1 - s_w) + \log(1 - s_f))/2$
  $D \leftarrow D - \alpha \partial \mathcal{L}_D / \partial D$ {Update discriminator}
  $\mathcal{L}_G \leftarrow \log(s_f)$
  $G \leftarrow G - \alpha \partial \mathcal{L}_G / \partial G$ {Update generator}
end for

Steps:

- We convert the text description to one-hot embedding. We have done this because the text descriptions supported in this implementation were very less.
- In the generator G, first we sample from the noise prior $z \in R^Z \sim N(0, 1)$. We encode the embedding of the text description using fully connected layer. The encoded text(φ(t)) description is concatenated to the image vector z(noise). The first step in figure 4 describes the concatenation. Following this, inference proceeds as in a normal deconvolutional network: we feed-forward it through the generator G; a synthetic image $x\hat{}$ is generated via $x\hat{} \leftarrow G(z, \varphi(t))$.
- In the discriminator D, we perform several layers of stride2 convolution with spatial batch normalization followed by leaky ReLU. We encode the embedding of the text description using fully connected layer. The encoded text(φ(t)) description is deep concatenated with the penultimate convolutional layer. The penultimate step in figure 4 describes the concatenation.
- Batch normalization is performed on all convolutional layers.
- The Discriminator is trained with three types of data samples. First type: Real images are paired with correct text descriptions and labelled as true (1). Second type: Real images are paired with incorrect text descriptions and labelled as false (0). Third type: Fake images are paired with correct text descriptions and labelled as false (0). Such a training will enforce the discriminator to learn which image-description pair is real, and the description is matching the appropriate image.
- The Generator takes random noise and the correct text describing the image. For this image-description pair it gets the prediction from the discriminator. The generator now tries to learn new weights by minimizing the error of generator.

5. LIMITATIONS AND CHALLENGES

- Vanilla GAN was difficult to train and get reasonable results. Several times discriminator becomes too good at its job, and the generator gradient diminishes to a point that it vanishes, and generator learns nothing in between iterations. This is the problem of the "vanishing gradients". Due to this, it was taking a lot of iterations to improve even a bit in generating an image.
- In comparison, even though DCGAN produced better looking images and were observed to be more stable, it did not capture noise in the data like the Vanilla GAN did on several occasions.
- Enhancing the text to image synthesis model for complex text descriptions to support arithmetic operations:
  - MNIST dataset has only images of 0-9 digits. To support arithmetic operations, we created a data set with arithmetic operations as text descriptions and images representing the results of arithmetic expressions.
  - With MNIST data we could only have those arithmetic expressions which gives result in the range 0-9. This dataset had very less examples to train GAN.
  - To resolve this, we need data set of images with multiple digits in an image and the description of arithmetic operations.
    Example:

Text Description	Image
"338 + 45"	

  - We are currently building such a data set.

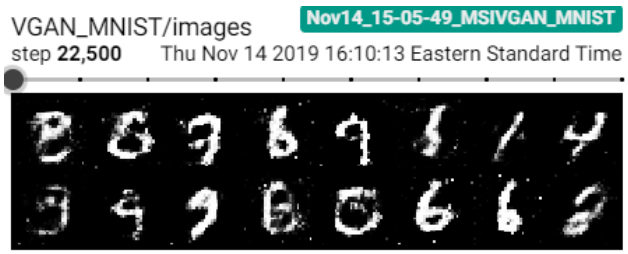
6. EMPIRICAL RESULTS

6.1. VANILLA GAN

Vanilla GAN takes huge number of iterations to generate distinguishable images which in turn consumes hours of training time. We had to run at least 200 iterations to generate the images as shown in the second figure below. We might achieve Nash equilibrium but with exhaustive number of iterations. We could not confirm an exact Nash equilibrium is achieved. But we reached almost near to the equilibrium at epoch 200 with D(x) = 0.5411, D(G(z)) = 0.4193.

Here are some images generated by generator at multiple iteration steps:

Iteration 10:



Iteration 200:

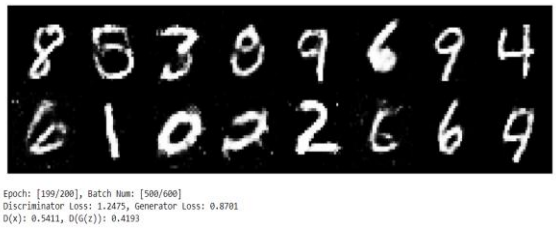


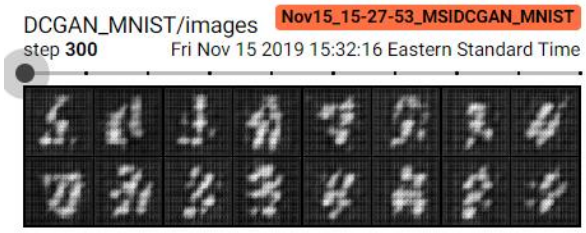
Figure 5: Images generated by Vanilla GAN

6.2. DCGAN

DCGAN captures the correlation between different object parts in an image, hence better accurate image generation results. We could achieve high quality image generation with only 10 iterations consuming 1/10th the training time of vanilla GAN.

Here are some images generated by generator:

Iteration 1:



Iteration 10:

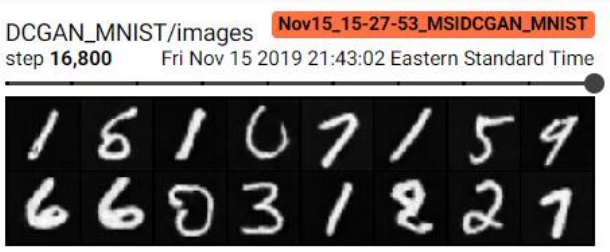


Figure 6: Images generated by DCGAN

6.3. TEXT TO IMAGE SYNTHESIS

As this implementation uses deep convolutional networks for generator and discriminators, we achieved similar results as that of DCGAN with respect to image generation. But in this case, we had overhead of generating images as per the text description. The model learnt to generate same quality images with text embeddings too. It has learnt the true distribution and is able to generate new unseen images every time. These generated images seem to have come from the true distribution.

Here are some images generated by generator at iteration 10 for the text descriptions mentioned on top of each image respectively. We can see that all the four generated 9's are different from each other and look like they have come from the true distribution.

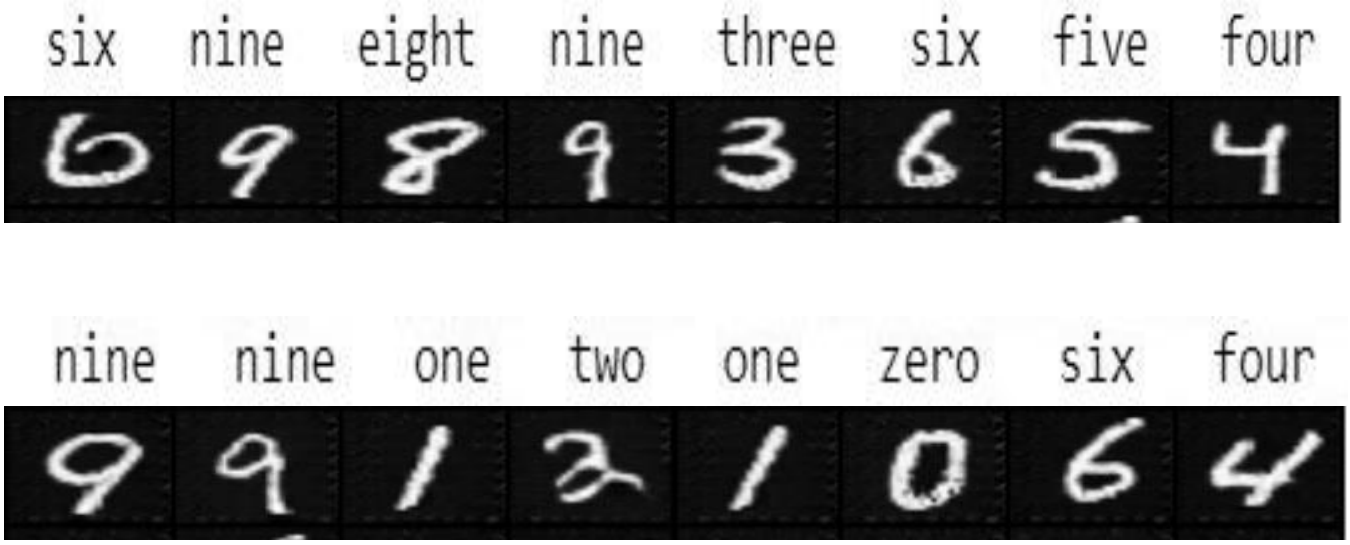


Figure 7: Text to Image Synthesis results

7. ANALYSIS

7.1. VANILLA GAN

The discriminator error is very high in the beginning. As time passes, discriminator gets better and its error decreases till it reaches 0.5. The generator error increases, which shows that the discriminator can correctly classify the fake samples.

With time, the generator gets better i.e. its error decreases. This results in increase of error of discriminator which means that the generator is generating better images with time and discriminator is failing to identify them as fake images.

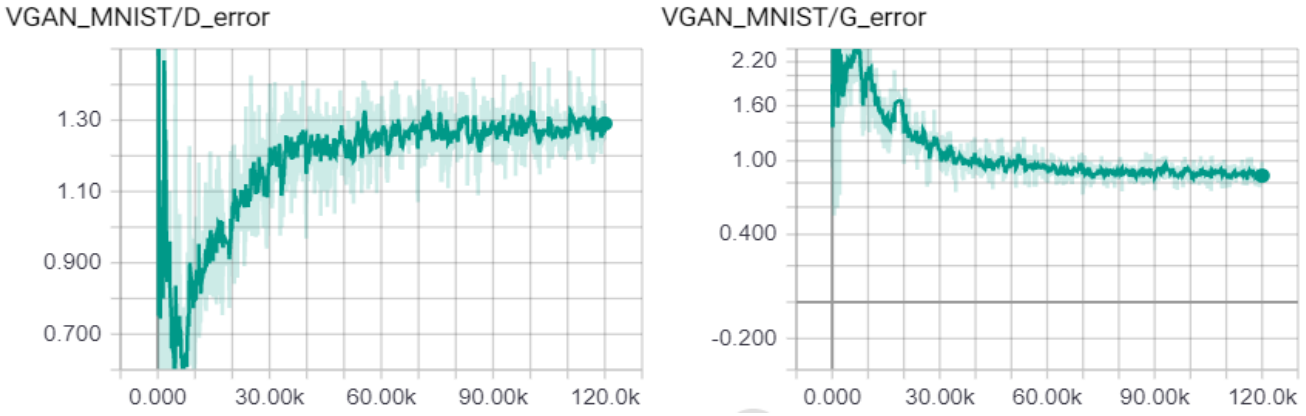


Figure 8: Discriminator and Generator error of Vanilla GAN (x-axis: no of steps, y-axis: error)

7.2. DCGAN

The discriminator and generator both have fluctuating errors. Even in the earliest stages, the discriminator loss is fluctuating rather than increasing steadily. Over the period, the generated images are getting better but its not capturing the noise as in the real data. That's why Generator error is stagnant. With more and more epochs this implementaion might achieve equilibrium. But in our case as we have noise in the real data(MNIST), the discriminator is still able to identify some fake, due to lack of noise in generated images. When generator captures the noise information and generates images with anticipated noise, discriminator and generator will achive nash equillibrium. But this might take a long time. But our goal of generating better images is achieved before the equilibrium is even reached.

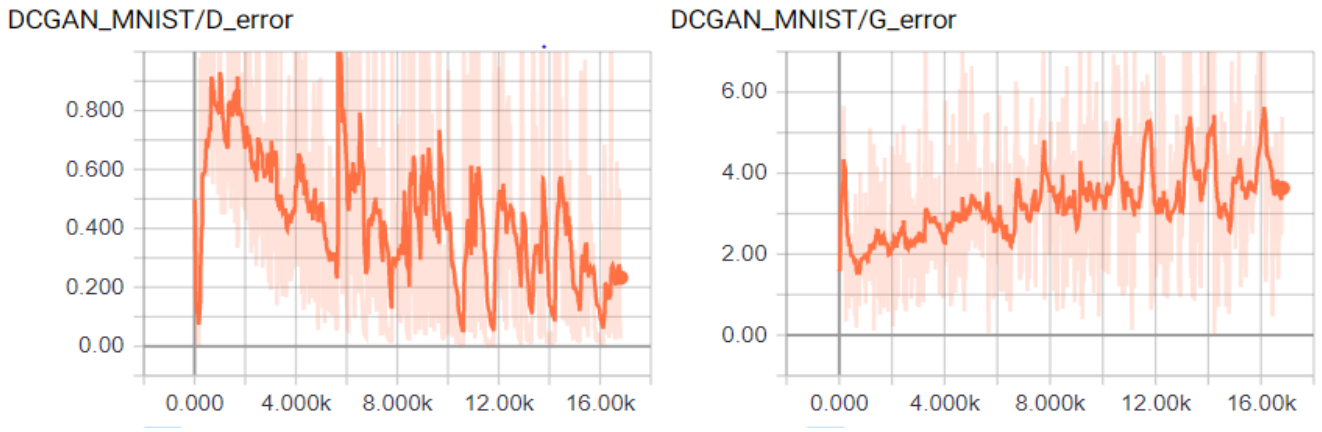


Figure 9: Discriminator and Generator error of DCGAN (x-axis: no of steps, y-axis: error)

7.3. TEXT TO IMAGE SYNTHESIS

As this implementation is built on top of DCGAN architecture, the learning pattern is same as the above. In this implementation, even though images are trained with its text descriptions, the error pattern remains the same.

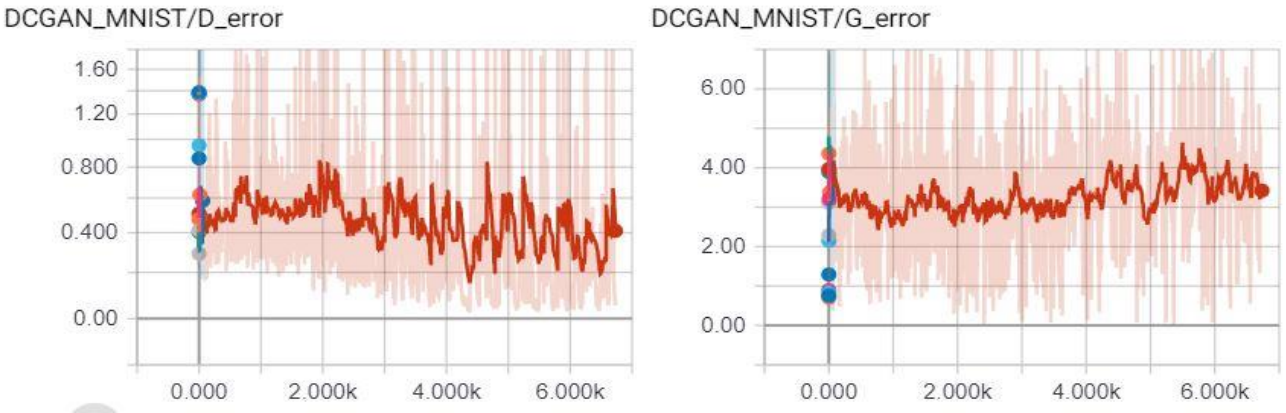


Figure 10: Discriminator and Generator error of Text to Image Synthesis (x-axis: no of steps, y-axis: error)

8. FUTURE AREAS OF EXPLORATION

As mentioned in the section 5 (Limitations and Challenges) we are currently creating a data set with arithmetic expressions as text descriptions and images (each image consists a multiple digit number) representing the results of arithmetic expressions. Once this is complete, we can use this dataset to train text to image synthesis implementation to generate results of arithmetic expressions as an image.

9. CONCLUSION

In this project we developed a simple and effective model for generating handwritten images based on simple textual description. We demonstrated that the model can synthesize many plausible visual interpretations for a given text. In this work we learnt the concept of Generative Adversarial Networks and implemented Vanilla GAN and DCGAN from scratch. We further built a Text to Image Generator using DCGAN and GAN-CLS algorithm. In future work, we aim to further scale up the model to generate images for more complex text descriptions such as supporting arithmetic expressions.

10. REFERENCES

[1]. Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, Honglak Lee (2016) Generative Adversarial Text to Image Synthesis.

[2]. Ian Goodfellow, NIPS 2016 Tutorial: Generative Adversarial Networks.

[3]. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio (2014), Generative Adversarial Networks.

[4]. Alec Radford, Luke Metz, Soumith Chintala (2015), Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.