## Creating a GAN for the MNIST Dataset

Yash Patel

Big Data Analytics, Worcester Polytechnic Institute, [yppatel@wpi.edu](mailto:yppatel@wpi.edu)

### Introduction

Being able to generate completely new images is one of the benefits of unsupervised learning. One of the classic examples of this is the MNIST dataset, which are images of numbers from 0 - 9. This is a good starting dataset to be able to practice creating GAN’s which is what this report is about.

### Methodology

**Input**

I decided to flatten the images from 28 x 28 to 1 x 784. This would allow me to use Linear layers. I could have used convolutional layers but decided against it because linear layers can achieve the same result.

I also created my noise vector to be 1 x 128. This would be fed to generate an image of 784.

**Generator**

I created a Generator with 4 fully connected layers with leaky relu activation function with dropout 0.2 at every layer. I then use tanh at the end for my final activation function because the it was stated on multiple sites that this would be the best activation function to use. I use the Adam optimizer for momentum and descent.

Neurons per layer:

400 -> 800 -> 1600 -> 784

**Discriminator**

My Discriminator is 4 fully connected layers with leaky relu activation function and dropout of 0.2 at every layer. I use sigmoid activation function at the end for classification. I use the Adam optimizer.

Neurons per layer

784 -> 1568 -> 512 -> 256 -> 1

I also used the Binary Cross Entropy loss for the generator and discriminator. I attempted to use the Wasserstein loss function however I couldn’t get a better model.

**Parameters**

| Learning Rate | Used in Adam optimizer | 0.0002, 0.0005, 0.00025 |
| --- | --- | --- |
| Epochs | Number of times to iterate dataset | 300, 100, 50 |
| Batch Size | Batch to train at once | 512 |
| Threshold to Reject | Generate a random number, if below this threshold then discriminator will use a mask | 0.02, 0.08, 0.007 |
| Mask Value | If we want to reject we will replace it with this value to not completely ignore the data. | 0.2, 0.4, 0.75 |
| Number of Times to train Discriminator | During every batch, how many times we train the discriminator to generator | 5, 2, 1 |

**Training**

I train the models for about 300 epochs. In each epoch, I get the batch size. For each batch where I do the training, I obtain the real images equal to the batch size. I then train the discriminator for k number of steps. The purpose is to have the discriminator stronger near the beginning.

While training the discriminator I generate noise vectors that are passed into my generator and output as fake images. I then use this set of fake and real images for training the discriminator. I would create positive (1) labels for the real images and negative (0) labels for the fake images. While training my data, my discriminator would get too strong so my generator would produce only one image which is mode collapse. To solve this problem, I would sometimes reject only certain numbers in a batch. This would allow for more variety in the images produced. I then calculate the loss for both the real and fake images and sum them to get the total sum for the discriminator. Remember that this is done k times in one batch.

Then we train the generator. We create a noise vector that we will use to generate fake images. The label we use is all positive because we want the discriminator to think that all of the generated images are real. We calculate the loss from the discriminator and we update the result.

### Experiments

To be able to get a recognizable set of images, I had to test many different parameters. The batch size was a large one early on where I wasn’t able to get the generator to decrease in loss. The learning rate I found to be best close to a very small number otherwise the loss wouldn’t ever decrease. While training, I found it best to train for a starting of 300 epochs and then change the threshold and mask value and train for a smaller set of epochs. This would allow me to get a stronger model without using up all of the RAM google colab provides. I found that increasing the threshold to reject causes a larger variety in numbers but the with a lot of noise. So I would change it in with different number of epochs. The mask value was low when I wanted to forget numbers but I change it back to higher values when I wanted to remove noise. I also tested using a Wasserstein loss function however I found no improvement in my models.

### Results

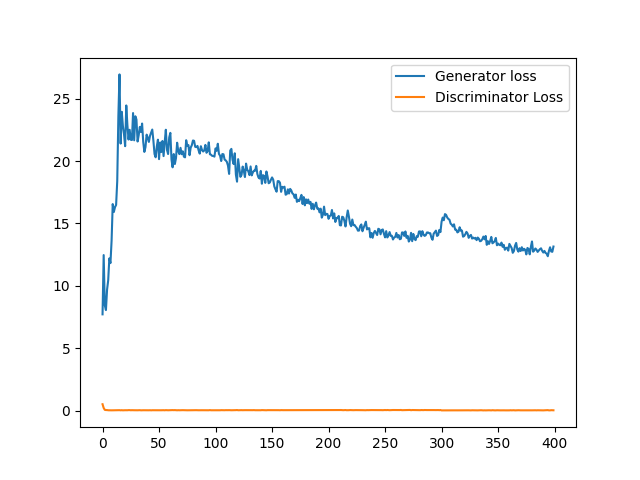


Figure 1: Loss over epochs

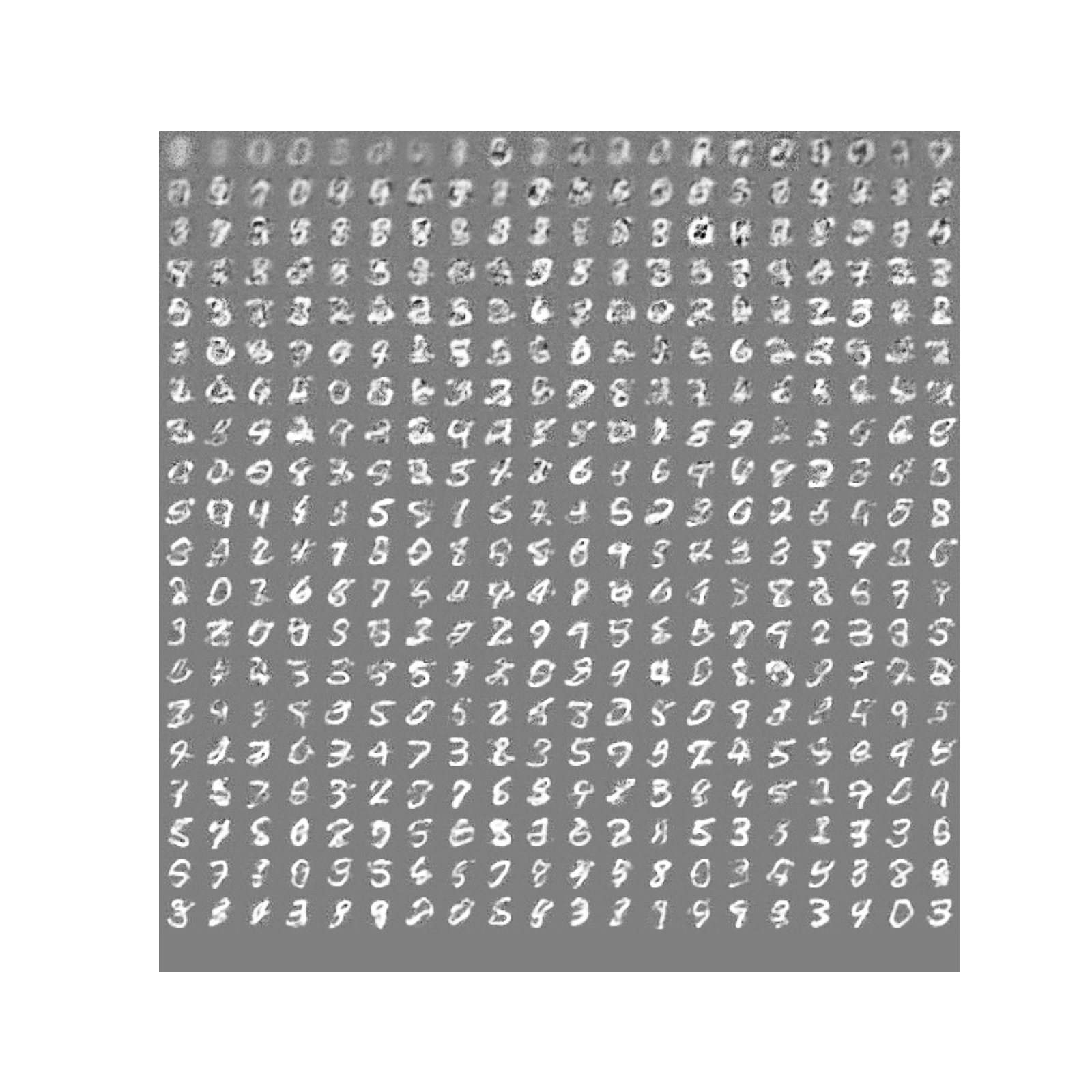


Figure 2: Image after every epoch



Figure 3: 25 images created

My loss function for the generator eventually started to decrease. It however doesn’t approach close to the discriminator. This I believe is because I trained the discriminator longer than the generator. You can run the results from evaluation.py. I am able to generate most of the numbers although some are rarer than others.