Visualizing loss landscapes of Generative Adversarial Networks using Filter Normalization

Badhri Narayanan Suresh (bns332)
Shantanu Tripathi (st3810)

1 Introduction and Problem Description

In the following work, we aim at visualizing the loss landscapes for the GANs. For the purpose of visualization, we will use Filter Normalization technique. We will begin by first training a GAN on a small dataset such as MNIST and try to visualize both generator and discriminator networks separately. Once the visualization for GANs turns out to be a success, we will study the effects of few hyperparameters such as batch size, number of epochs and infer its effect on the learning from the visualizations.

2 Literature Survey

We started by studying Generative Adversarial Networks and got to know the challenges in training a GAN such as avoiding collapse during training. It was clear that GANs had the ability to construct an arbitrary loss function depending on the problem and data it was given unlike other deep networks. We want to apply the filter-normalization technique introduced in Hao Li et al.(2017)[1] for visualizing the loss landscape of GANs and other deep neural networks. The reason for this is to visually study and analyze the challenges in training GANs and possibly infer other interesting properties of GANs from the landscape.

3 Datasets

We used MNIST for visualizations, as GANs are shown to work well with this simple dataset. Using a complex dataset might make it harder for GANs to learn without any collapse and this might be a problem for analysing the loss landscapes and the effects of different parameters on the loss. We also tried training the network with CIFAR 10, but the performance was not satisfactory for pursuing the visualizations. In order for the generator and discriminator visualizations to be reliable, good performance is crucial and that's the main reason we chose MNIST.

4 Models

We used Convolutional networks for both the generator and discriminator. The generator had 1 dense and 3 convolutional layer and the discriminator had 2 convolutional and 1 dense layers. The non-linearities used in the generator are ReLU and sigmoid (at the end) whereas discriminator had LeakyReLUs to enable non-zero gradients to pass during backprop. Initially, we had batchnorm layers in the generator and that was causing instability while training and therefore, we removed all such layers.

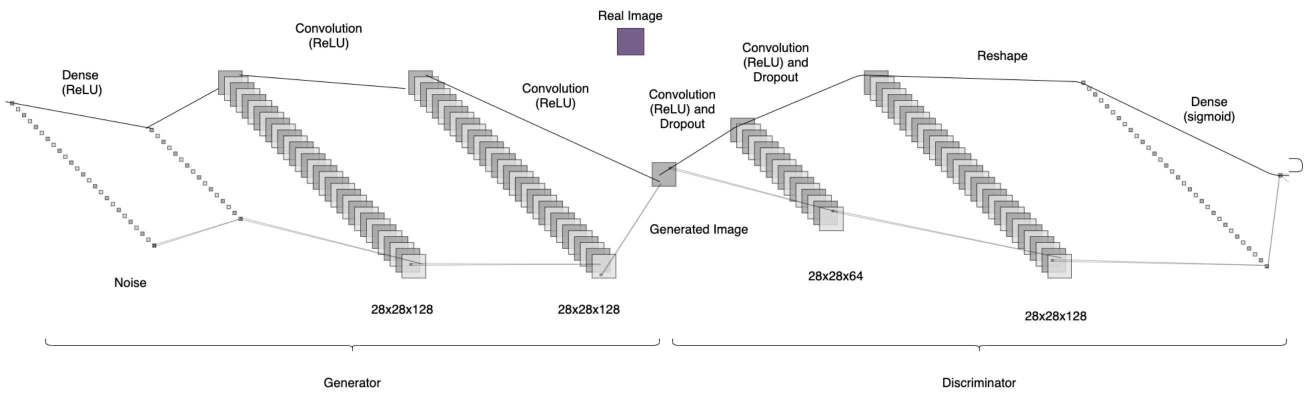


Figure 1: GAN Architecture used

Later, we came across a paper(Sitao Xiang et al.(2017) [2] that studied the negative effect of batchnorm layers on the GAN's performance. Intuitively, batchnorm tries to restrict the distribution of feature maps and that restricts the generator in producing images with a large diversity and it also sometimes leads to partial mode collapse. We also included dropout in the discriminator network and the presence of batchnorm and dropout might have also added to the training instability in the previous attempts that included batchnorm layers. Since we did not use any explicit regularizers, dropout layers helped the network to avoid overfitting.

5 Training Details

The loss function we try to optimize is the standard min-max loss comprising the binary cross entropy. The discriminator tries to minimize the binary classification loss and the generator tries to maximize the same, both working alternatively.

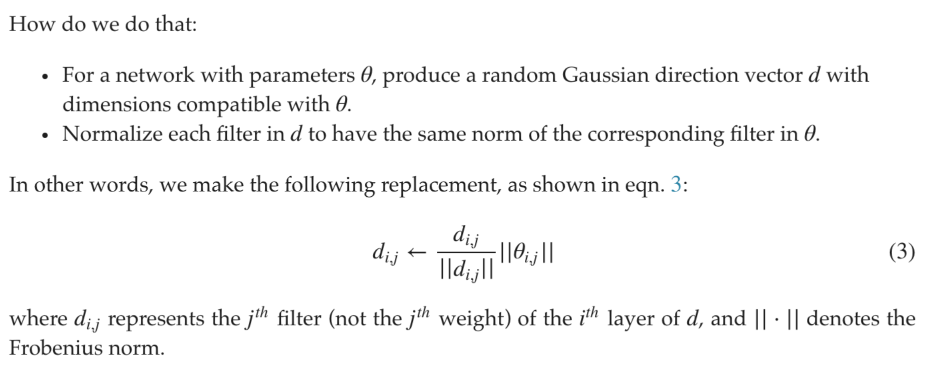
Table 1: Hyperparameters

Optimization	Adam
Input Size	28x28
Latent dimension	100
Learning rate	0.0002
Batch size	128
Loss	Binary Cross-entropy

The training is alternative in the sense that the generator first gets trained on a batch while the discriminator weights are frozen and then, discriminator is trained on both real and fake samples with the generator's weights frozen. Several other alternatives for the hyperparameters were tried and the above set were shown to perform the best.

6 Filter Normalization

Once the network is trained and generated images are verified to be semantically correct, we pass the network and it's weights to the filter normalization block. Suppose the number of weight parameters or the weight vector dimension is k, filter normalization chooses two orthogonal vectors in the k-dimensional space with the original weight vector describing the origin or the intersections. Once these two orthogonal vectors are chosen, a 2D raster is created by uniformly sampling k-dim vectors across the plane created. Each point in this 2D plane corresponds to a weight setting in the network.


How do we do that:

- For a network with parameters θ, produce a random Gaussian direction vector d with dimensions compatible with θ.
- Normalize each filter in d to have the same norm of the corresponding filter in θ.

In other words, we make the following replacement, as shown in eqn. 3:

$$d_{i,j} \leftarrow \frac{d_{i,j}}{||d_{i,j}||}||\theta_{i,j}|| \qquad (3)$$

where $d_{i,j}$ represents the j^{th} filter (not the j^{th} weight) of the i^{th} layer of d, and $||\cdot||$ denotes the Frobenius norm.

Figure 2: Filter Normalization procedure(Taken from [3])

However, there exists a problem of scale difference between the original optimal weights and the sampled weights. Scale invariance prevents us from making meaningful comparisons between plots, unless special precautions are taken. A neural network with large weights may appear to have a smooth and slowly varying loss function; perturbing the weights by one unit will have very little effect on network performance if the weights live on a scale much larger than one. However, if the weights are much smaller than one, then that same unit perturbation may have a catastrophic effect, making the loss function appear quite sensitive to weight perturbations. To solve this problem, we perform what is called the filter normalization. For each point sampled in the raster, we iterate through all the layers in the network, and normalize the weights in each layer and bring its magnitude equal to the corresponding layer's weight magnitude in the original optimal network.

7 Discriminator Loss Landscape

Once this raster is formed, we iteratively place the weights onto the discriminator networks and evaluate the loss with sample data keeping the generator weights constant. The loss computed is plotted in the z-axis and therefore, we get a 3D visualization of the loss landscape. The discriminator's performance is based on how well it classifies between real and fake images and therefore, the data used to compute the loss has equal number of real and fake images. The fake images are obtained from the generator by passing in Gaussian noise. Like we saw above, we have the optimal discriminator weights as the origin, and raster a number of equally spaced points in the 2D grid and compute loss at each location.

8 Generator Loss Landscape

The generator loss landscape is quite tricky because the loss always comes out at the discriminator end. The generator's performance is based on how well it fools the discriminator and therefore can only involve fake images for evaluating loss. We provide noise to the generator, fake images are generated and they are passed to the discriminator. The ideal output class favouring the generator must be 1 and the loss is computed accordingly. But it is important to note that the discriminator weights must not change during the filter normalization as we only intend to observe the generator's performance. Therefore, we mask the changes in the discriminator's side during rasterization of grid points.

We were facing an issue with visualizing generator loss before where the landscape was filled with Nan for the most part and few non-Nan values at the origin. This issue was solved by removing the batchnorm layers in the generator.

9 Results

The baseline results consists of generator and discriminator performance for 4 different number of epochs. The idea is to study the GAN with both the networks having various levels strength. For instance, for the first setting (10 epochs), the generator has not learnt enough to produce good images and therefore, the discriminator's task is simple and it has 100 percent classification accuracy. As the number of epochs increase, the generator starts to learn to produce better images and therefore, the networks start competing. As a result, the discriminator's performance starts to drop from 100 and this is evident in the loss curves below.

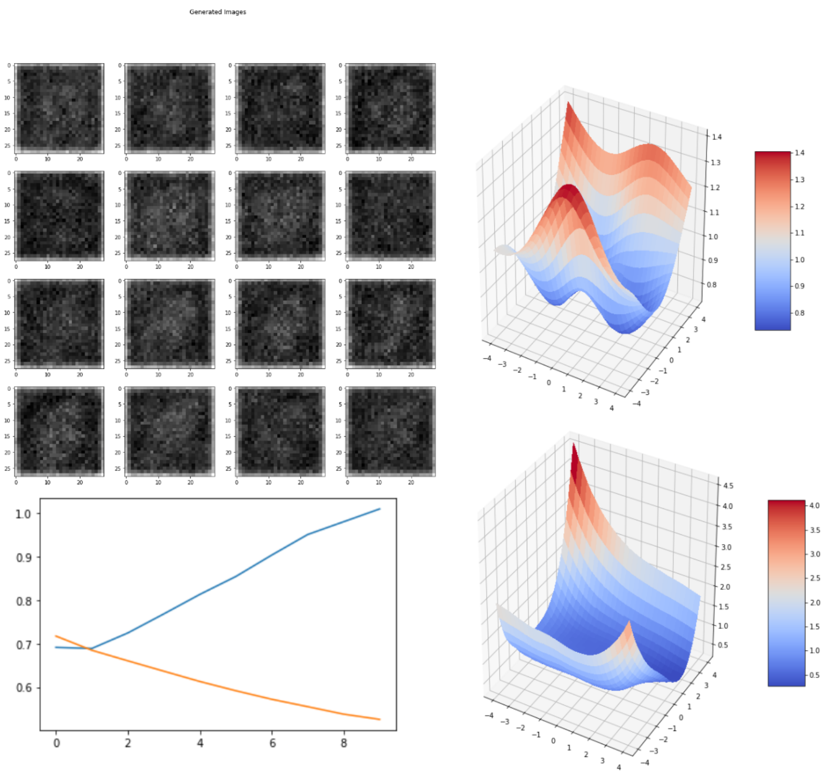


Figure 3: (Top left : Generated Images, Top right : Generator Landscape, Bottom left : Loss curve, Bottom right: Discriminator Landscape)

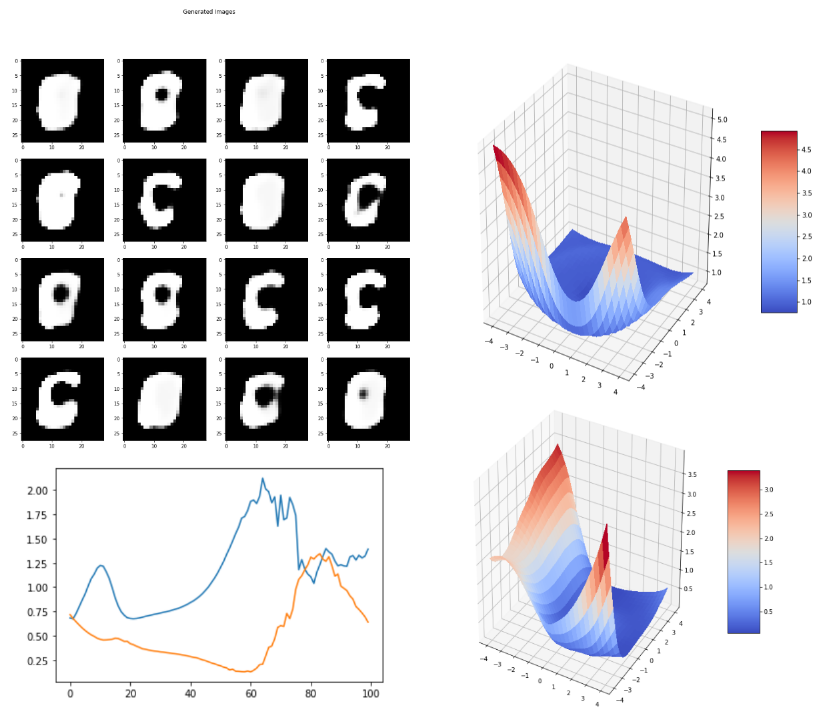


Figure 4: (Top left : Generated Images, Top right : Generator Landscape, Bottom left : Loss curve, Bottom right: Discriminator Landscape)

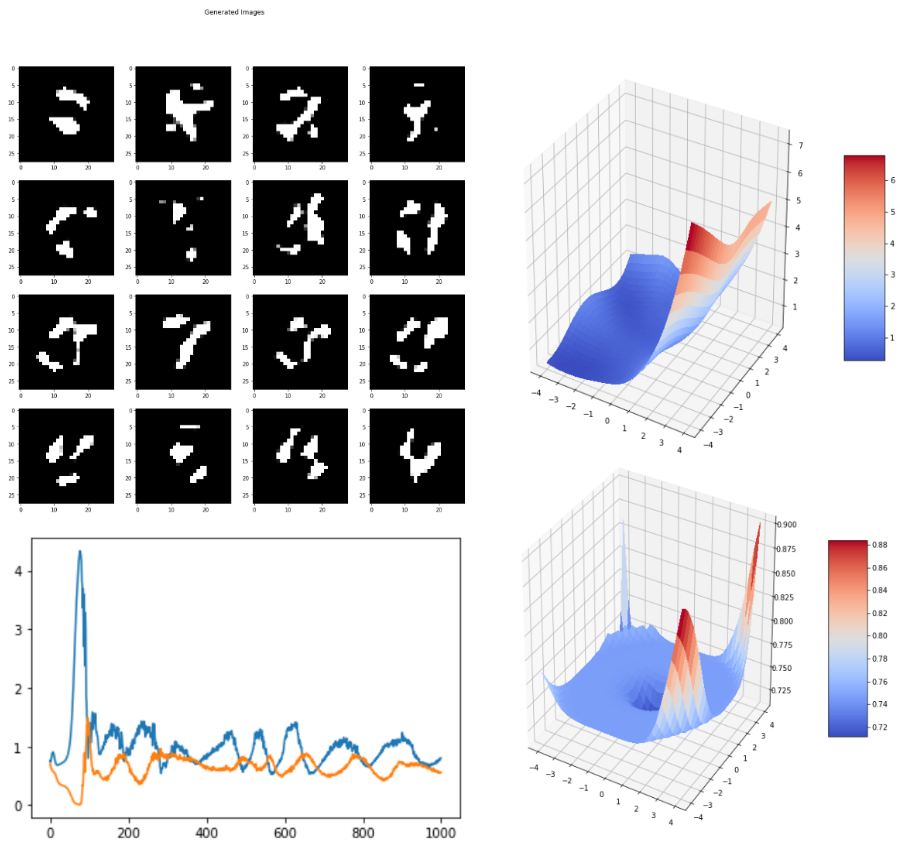


Figure 5: (Top left : Generated Images, Top right : Generator Landscape, Bottom left : Loss curve, Bottom right: Discriminator Landscape)

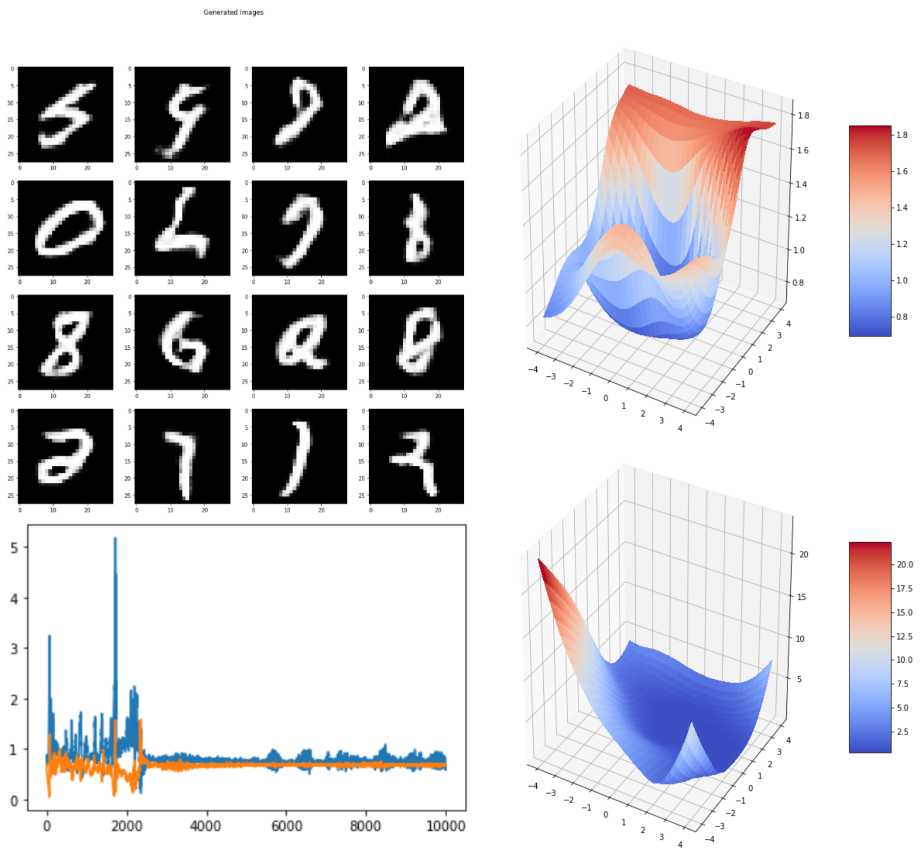


Figure 6: (Top left : Generated Images, Top right : Generator Landscape, Bottom left : Loss curve, Bottom right: Discriminator Landscape)

Epochs	Generator Loss	Discriminator Loss	Discriminator Accuracy
10	1	0.52	100
100	1.39	0.639	55.86
1000	0.805	0.551	73.83
10000	0.612	0.702	45.7

In most standard neural network architectures and specifically in classification based networks, the structure of the loss landscape would contain a global minima in the center and possibly several local minimas around it. However, the loss landscapes of the generator and discriminator showed that instead of a global minima, there is a "lake" of minimas that approximately have the same loss values. This is interesting because, there is a large set of weight settings that essentially give the same performance. Although, each weight set gives the same loss value, there might possibly be differences in what exactly it learns on the input. Also, since the performance of the networks are interdependent, it is very unlikely to have a point global minima for the networks.

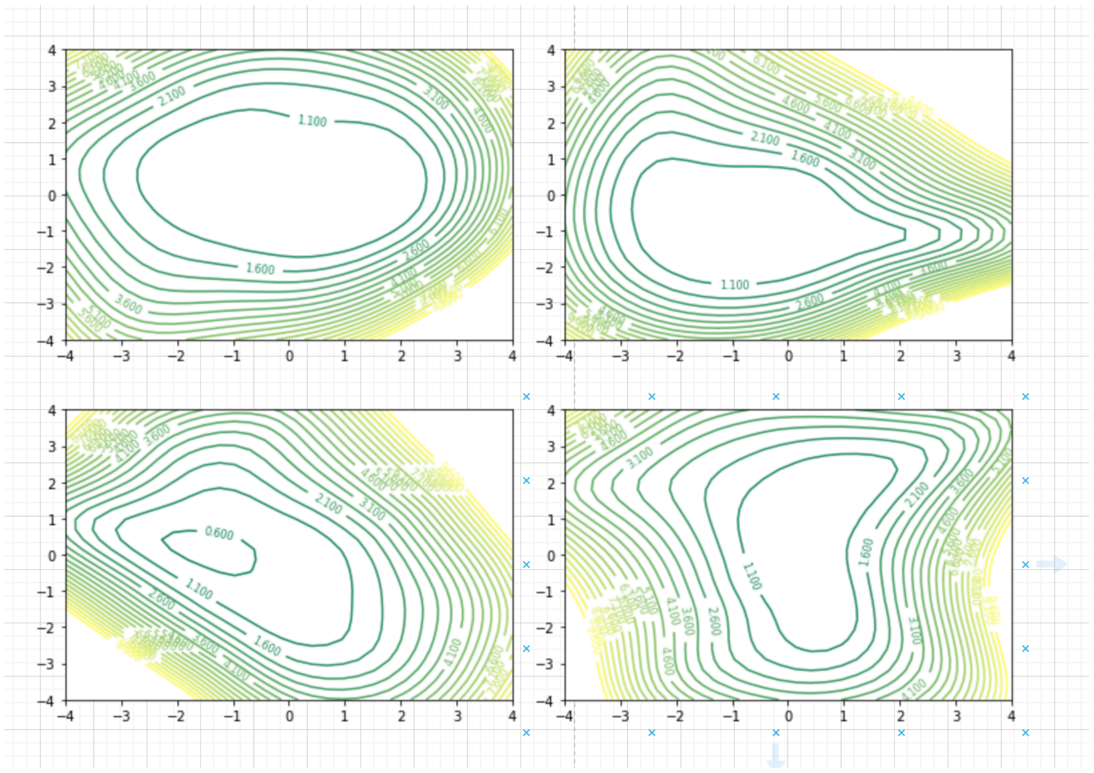


Figure 7: Discriminator contours with varying batch sizes (Top left : 16, Top right : 128, Bottom left : 512, Bottom right: 1024)

Epochs	Generator Loss	Discriminator Loss	Discriminator Accuracy
16	0.776	0.676	53.12
128	0.861	0.703	51.95
512	0.864	0.684	54.59
1024	0.706	0.68	56.54

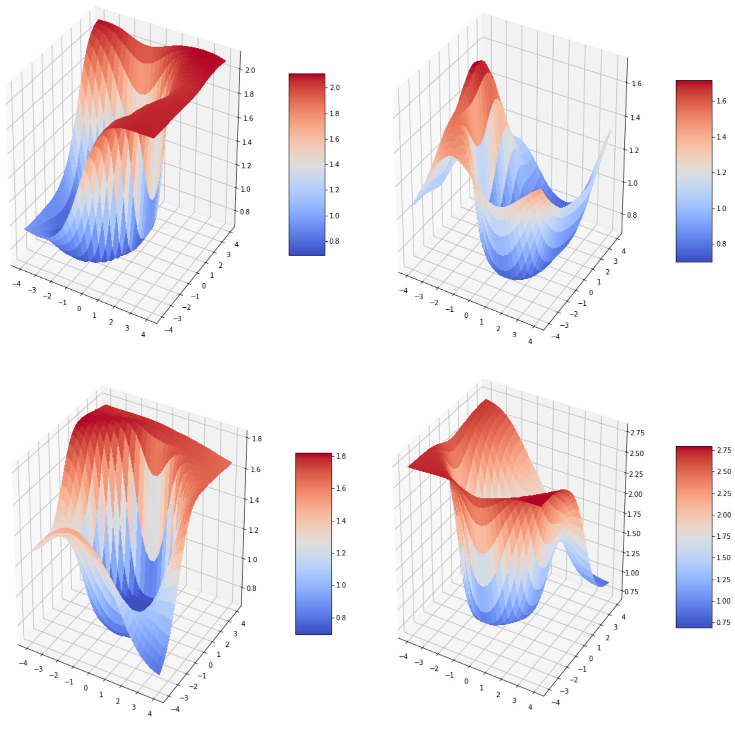


Figure 8: Generator landscapes with varying batch sizes (Top left : 16, Top right : 128, Bottom left : 512, Bottom right: 1024)

We also conducted another experiment where the batch size was changed and the contour plots were observed. It was clear from the plots that as the batch size increased, the contour plots were shrunk towards the center.

Conclusion

We intended to visualize the loss landscapes of Generative Adversarial Networks and analyze the effect of certain hyperparameters on the landscape and contour plots. From our literature survey and study, we found that this is one of the first attempts to visualize the loss landscapes of GANs. The landscape surfaces were obtained for a Convolutional-GAN trained on MNIST at several stages of training by varying the number of epochs thereby getting results for varying strengths on Generator and Discriminator. We also studied the effect of batch size on the convergence and loss contours of GANs and observed that as the batch size increased, the contour plots started to shrink. We plan on making this process modular and available at a ready-to-use library in python for others to visualize the loss landscapes of the networks they train as it is crucial to get a sense of how complex and non-convex the loss landscapes are.

The code can be found at : https://github.com/badhri123/Visualizing-GANs

References

[1] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, Tom Goldstein. Visualizing the Loss Landscape of Neural Nets. In ICLR 2018

[2] Sitao Xiang, Hao Li. On the Effects of Batch and Weight Normalization in Generative Adversarial Networks.

[3] https://jithinjk.github.io/blog/nn_loss_visualized.md.html