***Applying Dropout to GANs***

Generative adversarial networks belong to one of the most popular machine learning algorithms developed in recent times. They belong to a set of algorithms named generative models. These algorithms belong to the field of [unsupervised learning](https://en.wikipedia.org/wiki/Unsupervised_learning), a sub-set of ML which aims to study algorithms that learn the underlying structure of the given data, without specifying a target value. Generative models learn the intrinsic distribution function of the input data p(x) (or p(x,y) if there are multiple targets/classes in the dataset), allowing them to generate both synthetic inputs x and outputs/targets y’ typically given some [hidden parameters](https://en.wikipedia.org/wiki/Latent_variable).

***Generative adversarial networks***

GANs they have proven to be really successful in modeling and generating high dimensional data, which is why they’ve become so popular. Nevertheless they are not the only types of Generative Models, others include Variational Autoencoders ([VAEs](https://arxiv.org/abs/1312.6114)) and [pixelCNN](https://arxiv.org/abs/1606.05328)/[pixelRNN](https://arxiv.org/abs/1601.06759) and [real NVP](https://arxiv.org/abs/1605.08803). Each model has its own tradeoffs. Generative adversarial networks are a framework that integrates adverserial training in the generative modeling process.

Some of the most relevant GAN pros and cons for the are:

* They generate sharp and accurate images
* They are easy to train since no statistical interference is required and only back propogation is required to get the required gradients
* It is difficult to optimize GANs because of unstable training dynamics

the framework is composed of two models - one generator and one discriminator - that train together by playing a minimax game. The generator tries to fool the discriminator by producing random sample images and keeps learning to make the images as realistic as possible. The discriminator tries to differentiate between the real and fake samples generated by the generator. The discriminator gets better at distinguishing between the real and fake images over time.



***Shortcomings of a traditional GAN framework***

One of tha main problems of GANs is mode collapse where the generator is able to fool the discriminator by producing data only from connected components of the data manifold which leads to similar looking samples produced from a narrow scope of data. This leads to sub-optimal generator learning where the generator learns only a small segment of the actual distribution of the data. To counter this dropout was introduced and was proven to widely successful in countering the problem of overfitting. The idea behind using dropout is that neurons are not specifically dependent on certain other neurons to produce output.

In our case we try and test if we can use the dropout method with GAN and extend the concept by applying dropout in conjunction with two other methods viz. a viz. batch normalization and variance shift and test if we get better results.

***MNIST dataset***

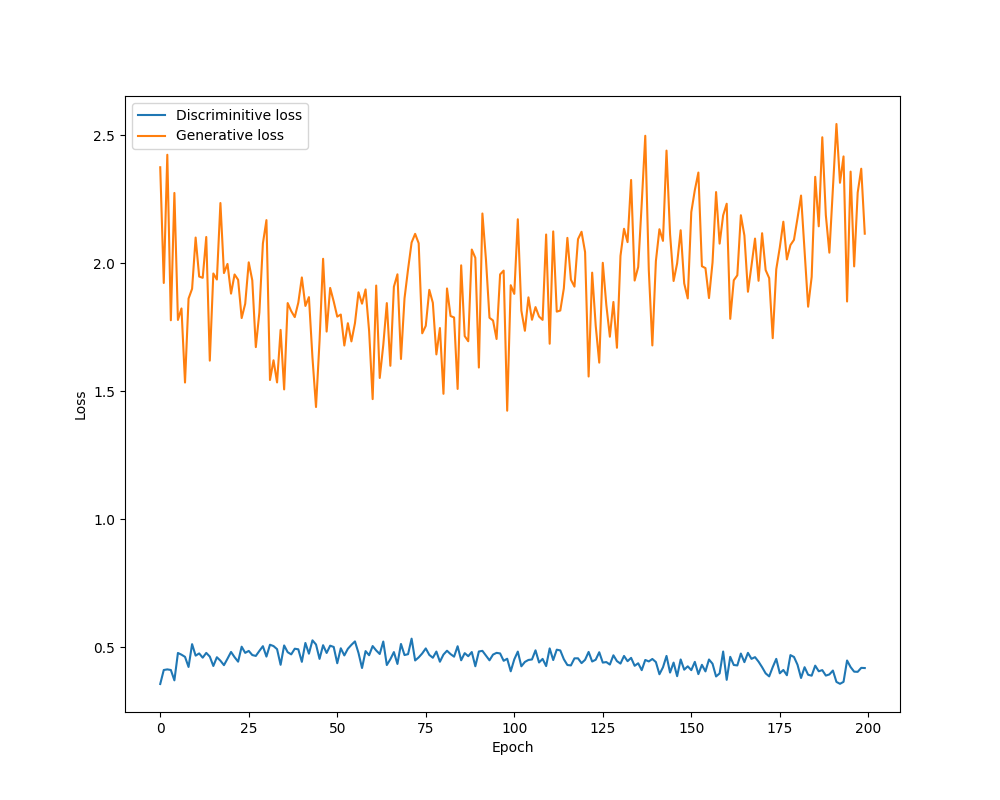
The MNIST dataset is composed of 10 classes of handwritten digits varying from 0 to 9. It is visible that the quality and variation of the produced samples increase while using dropout rate values of 0.2 and 0.5 across all different sized discriminator sets. On the other hand, the quality of the produced numbers deteriorates considerably while using high dropout rates, i.e., 0.8 and 1, or no dropout rate at all. However, the quality gets slightly better when using more discriminators on such extreme end dropout rates, since Generator might still get enough feedback to be able to learn at the end of each batch.

***Experimental Setup***

We built a discriminator using 3 hidden layers and used tensorflow to setup the neural network. First we normalize the images between 1 and -1. We provide tanh as the activation function for the last layer of generator inputs. Although traditionally loss functions are defined as min(1 – log(D)), we use max(log(D)) as our loss function so as to counter the effects of vanishing gradients, early on. For the problem of batch normalization we construct different mini-batches for real and fake i.e. each mini batch either contains only real images or only generated images. To avoid sparse gradients we use leakyRelu as our activation function in hidden layers which is stable in both the discriminator and the generator. We use the ADAM optimizer for both the discriminator and generator. Adam uses first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. We use adam because of its simplicity and computational efficiency. We set the learning rate equal to 0.002 and the beta\_1 equal to 0.5 keeping all other parameters default. We keep the batch-size equal to 128 setting the number of epochs to 200.

***Standard dropout results on the MNIST dataset***

We observe that the quality and variation of images steadily improve for lower values of the dropout. We test for dropout rate values of 0.2 and 0.5 find much better values for the generative loss as compared to standard GAN algorithm implementation with no dropout. In the first figure (GAN with no dropout) we can see that the generative loss values are considerably higher. However using dropout values of 0.2 and 0.5 we see a high improvement in the values of generative loss although the discriminative loss tends to stay about the same. However as predicted the generative loss for higher values of dropout at 0.8 tends to deteriorate. We also observe much higher values of discriminative loss for higher values of dropout.



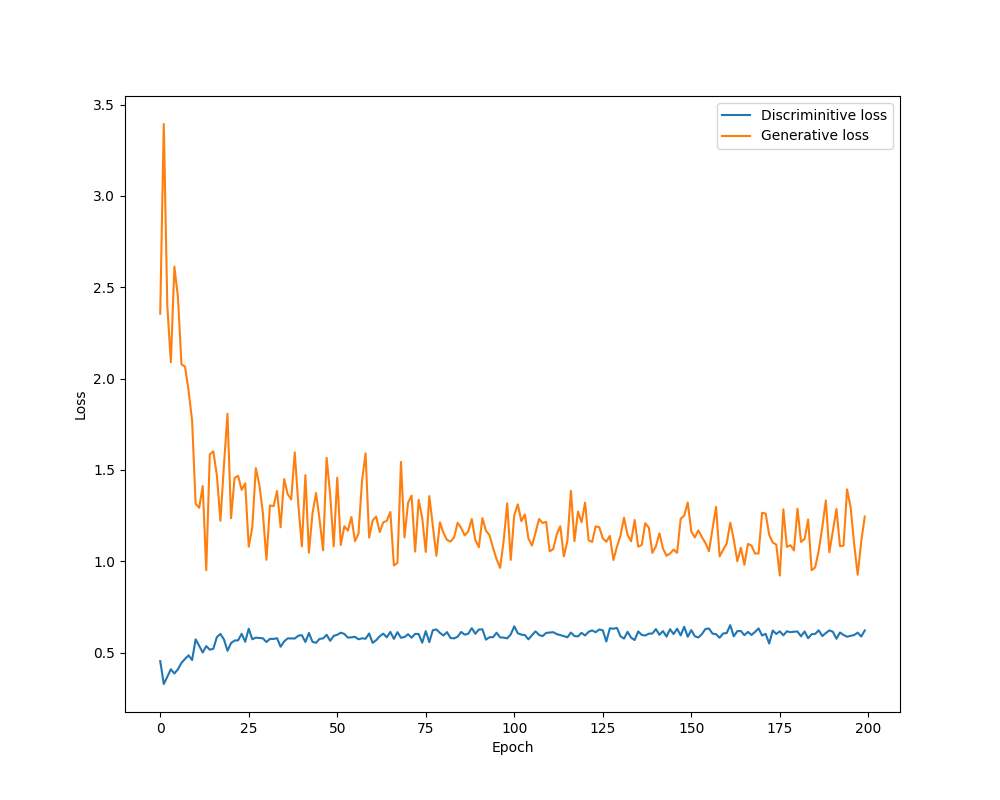
Fig: GAN without dropout

Fig: GAN with dropout 0.5

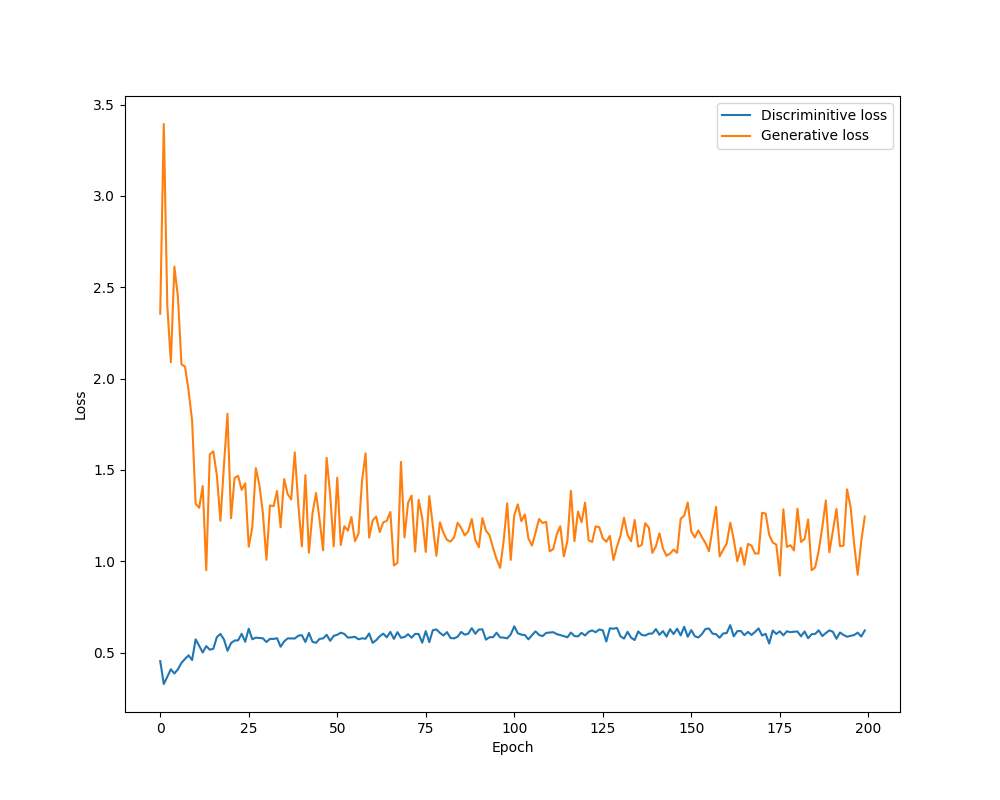
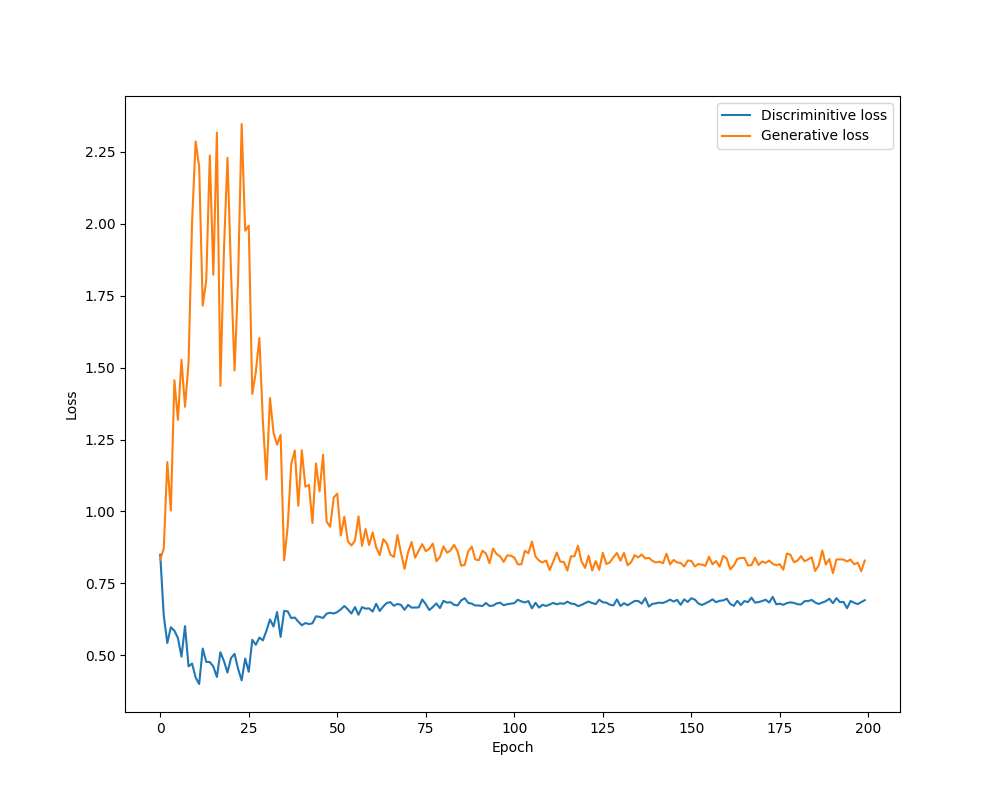
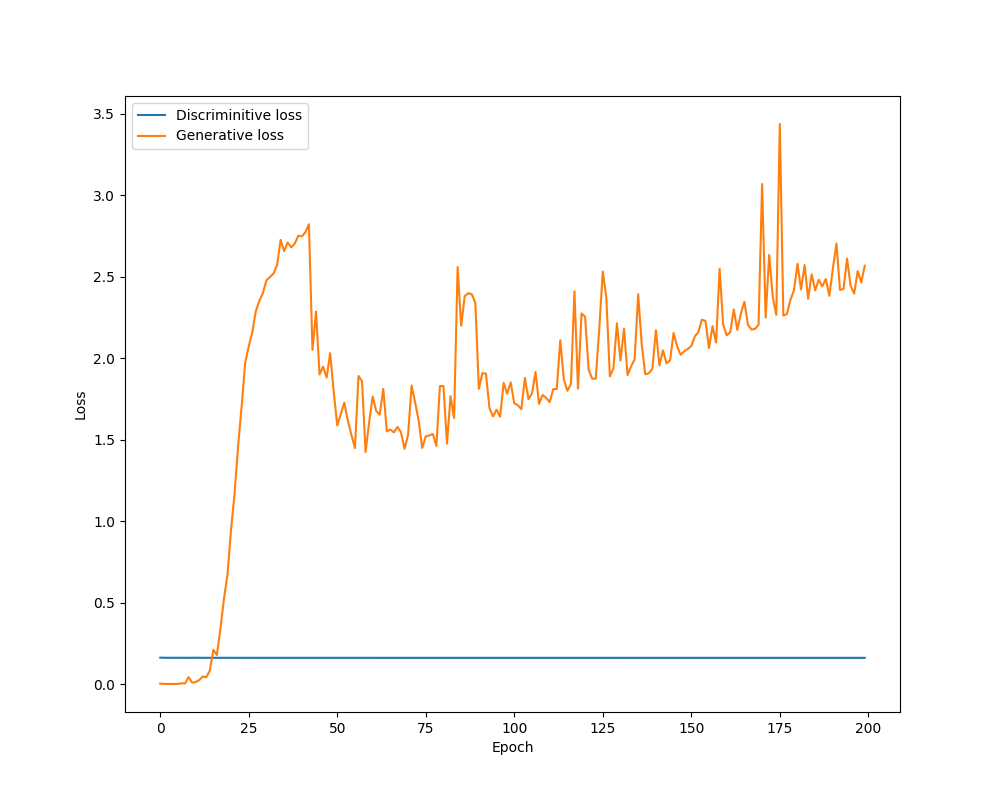


Fig: GAN with dropout 0.2

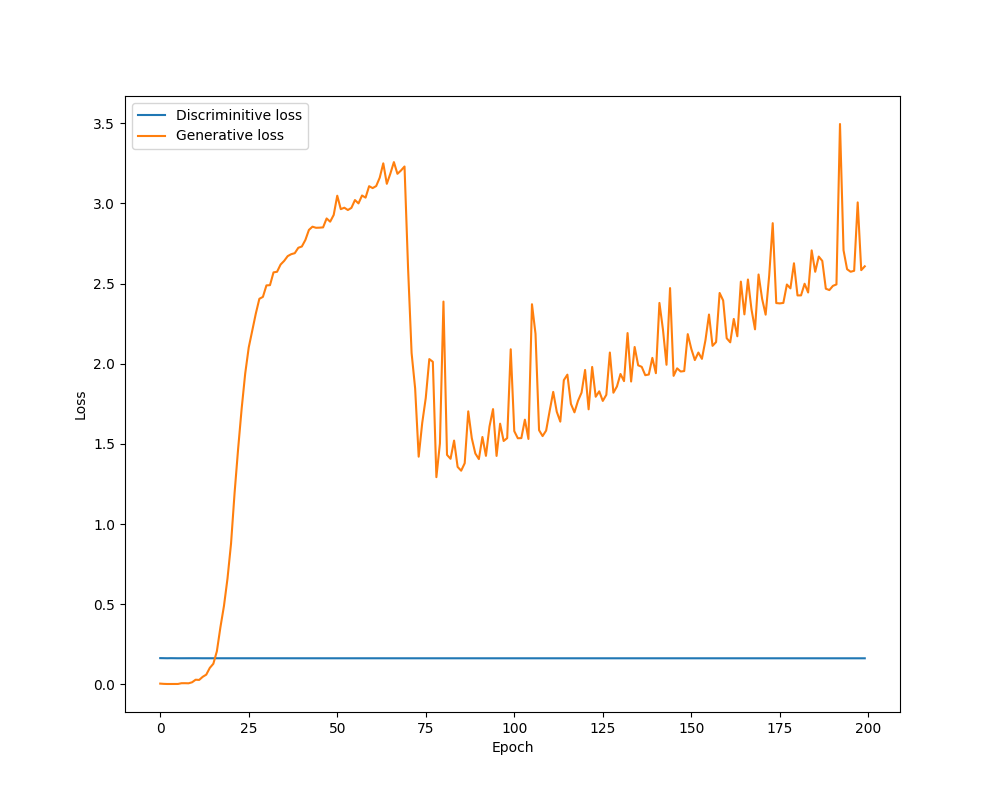
Fig: GAN with dropout 0.8

***Adding Batch Normalization with dropout***

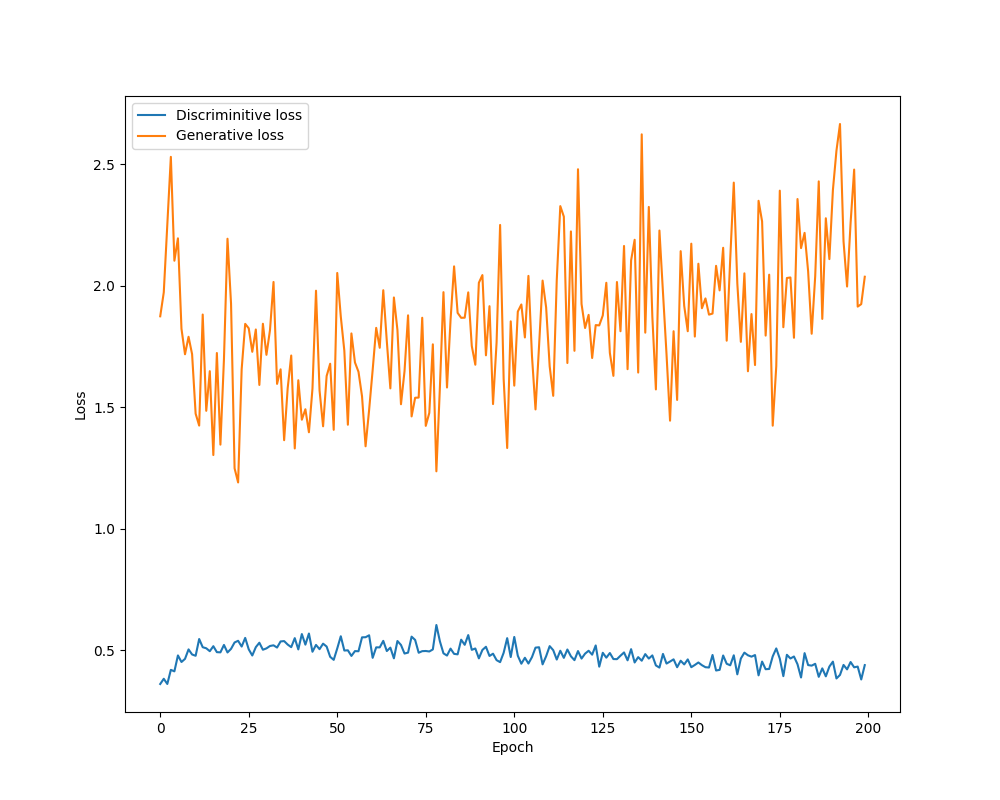
We observe that if we apply batch-normalization with momentum 0.9 the generative loss increases for higher epochs although a sharp drop is seen between 25th and 50th epochs. He experiment was setup where batch-normalization was applied for the first two hidden layers along with dropout with dropout ratio 0.2. A dropout of 0.2 was also applied in the last hidden layer without batch-normalization. We also tested by lowering the momentum and increasing the dropout ratio and we observed the same results. As such we conclude that this strategy is not feasible.

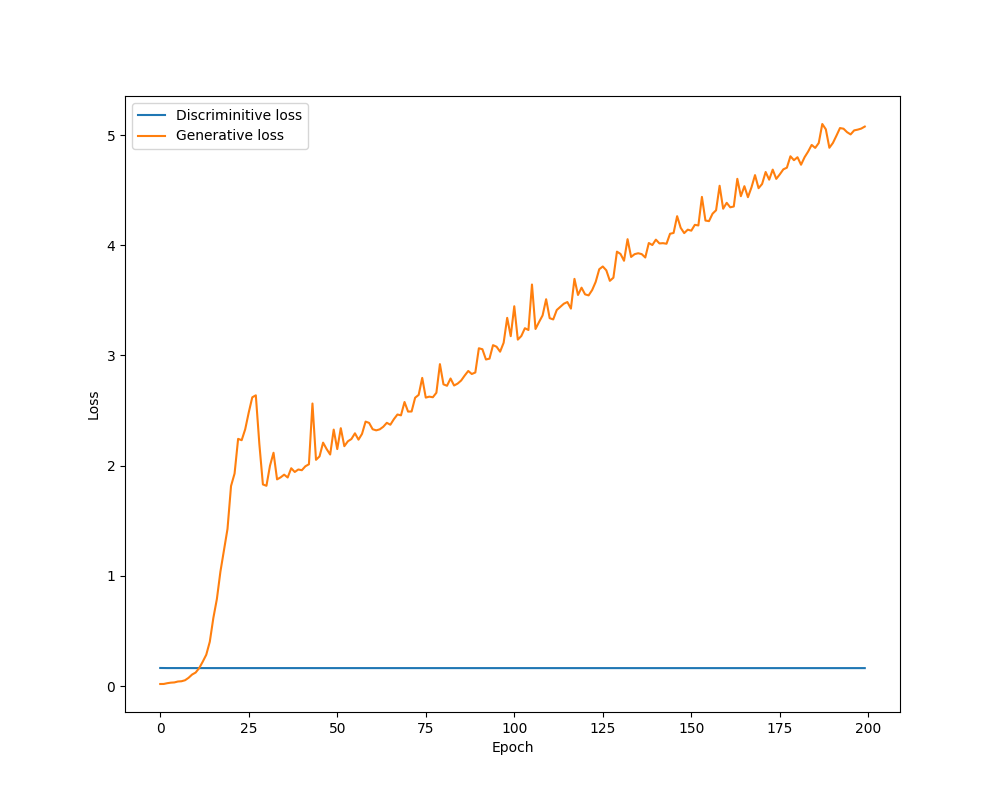
Fig: Batch normalization with dropout in the first two layers and only dropout in the third layer

In the second strategy, we apply batch-normalization to the first two layers without dropout and we apply a dropout with dropout ratio of 0.2 on the last layer. Here again, we see that there is sharp spike in the generative loss between epoch 1 and epoch 25 and although there is a sharp drop in generative loss close to epoch 75, we see a clear marked increase in the generative loss after 200 epochs. Although the values for discriminative loss look constant, there is in fact a slight variation in the values and we can refer to the appendix regarding the same. We can intuit the reason to be because of the variance shift occurring due to the fact that Dropout would shift the variance of the specific neural net whereas batch-normalization would try and maintain the variance which might lead to adverse results. As such both our strategies fail.

Fig: Batch normalization only in the first two layers and only dropout in the third layer

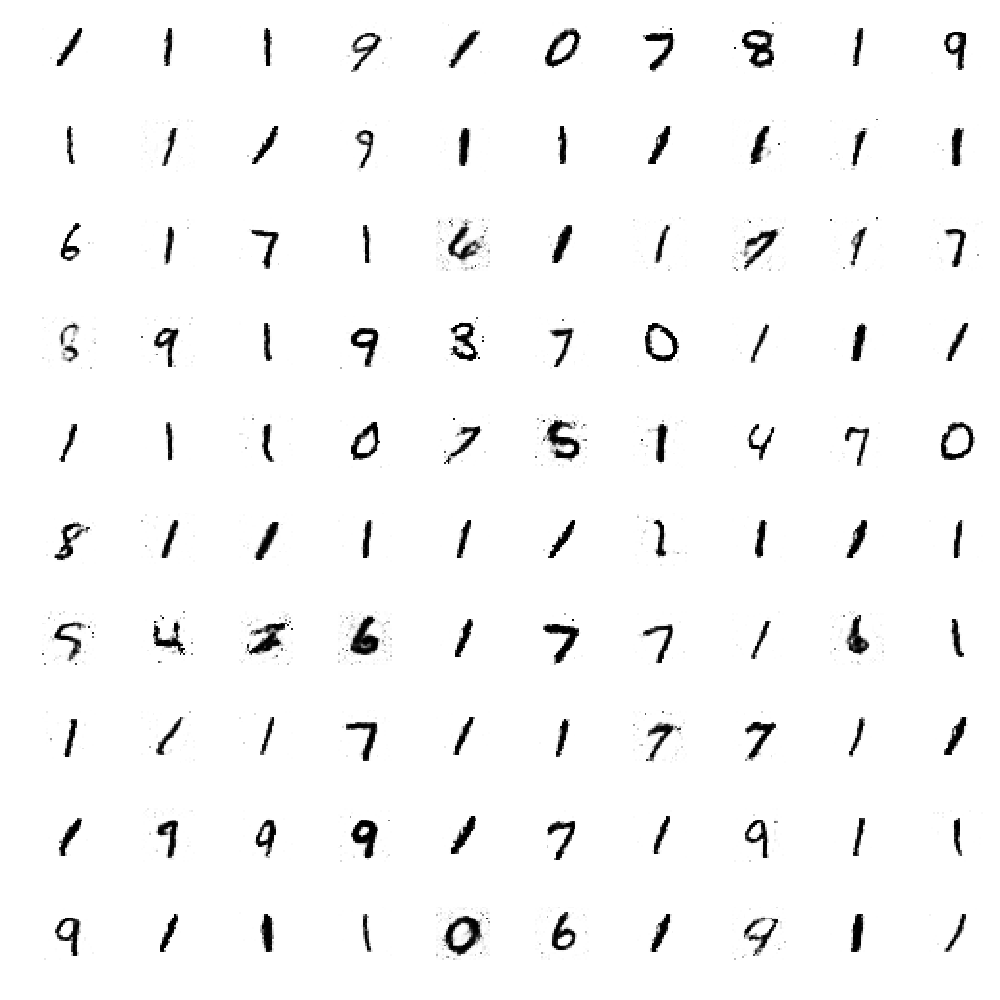
***Adding random variables with dropout***

Fig: Strategy with random variable (U ~ (-0.1, 0.1) added to x added with dropout of 0.2 in the first two layers and only dropout of 0.2 on the third layer

Fig: Strategy with only random variable in the first two layers and only dropout of 0.2 on the third layer

***APPENDIX***

Image1: Image after 200 epochs for no dropout strategy with d = 0.0

Image2: Image after 200 epochs for dropout strategy with d = 0.2

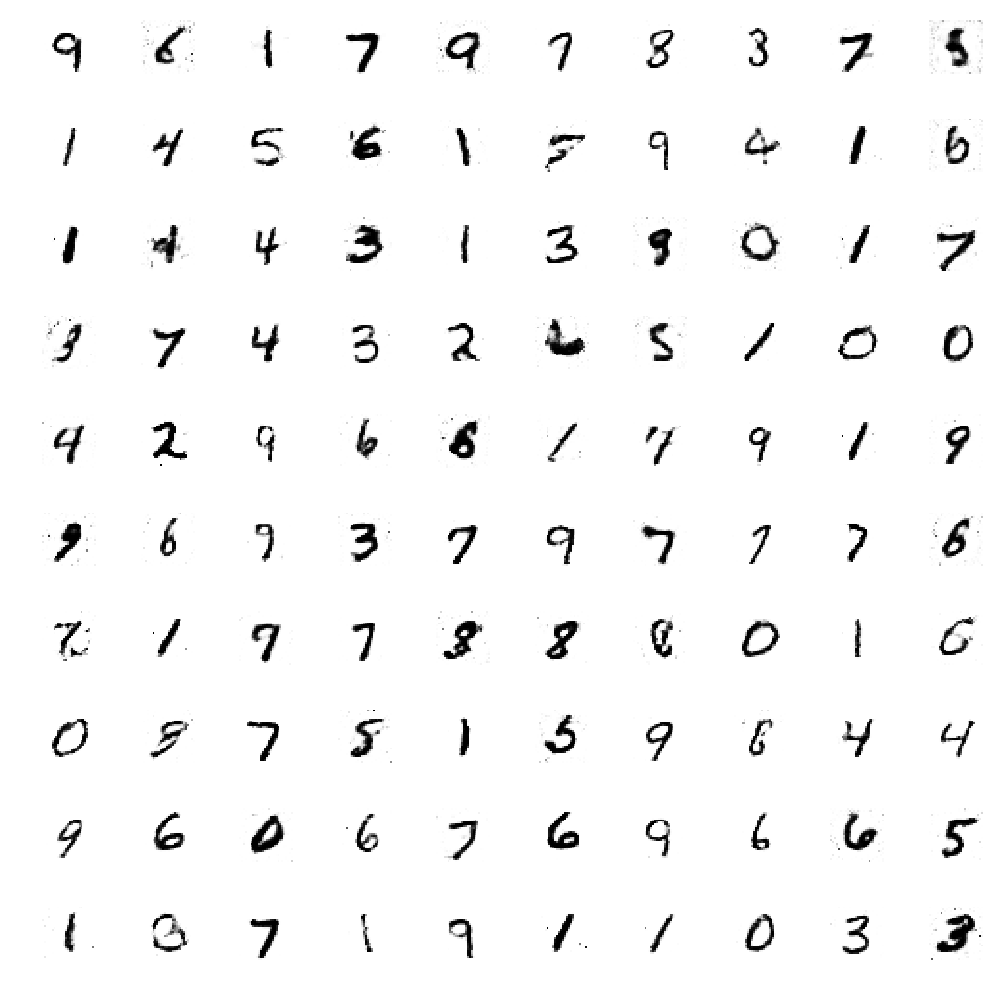
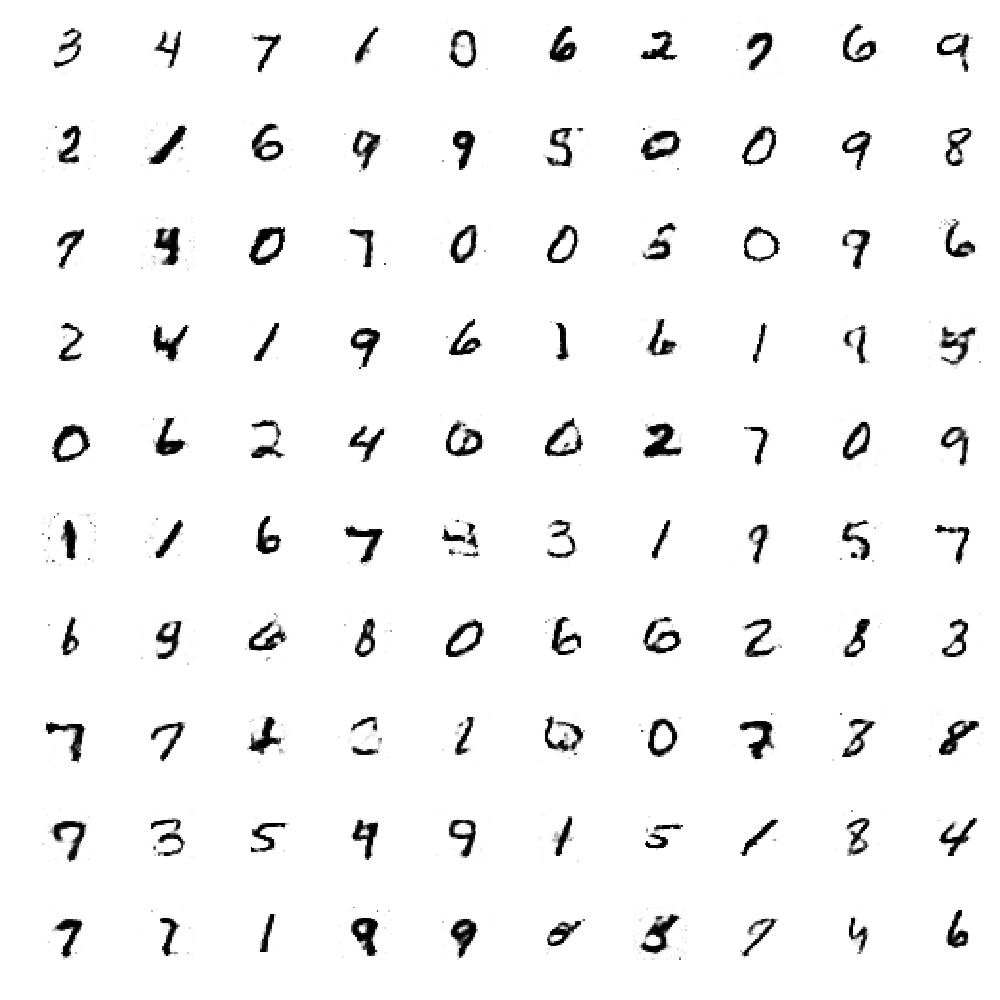


Image3: Image after 200 epochs for dropout strategy with d = 0.5

Image4: Image after 200 epochs for dropout strategy with d = 0.8

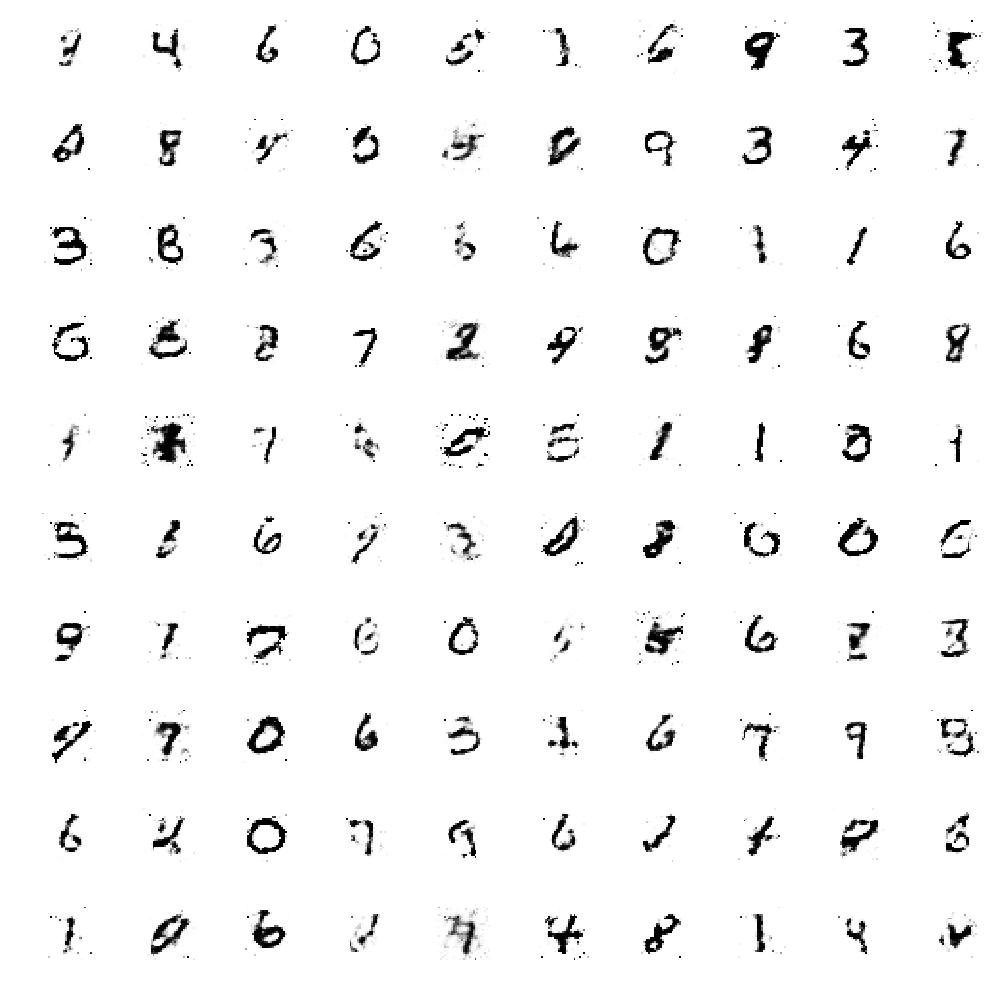
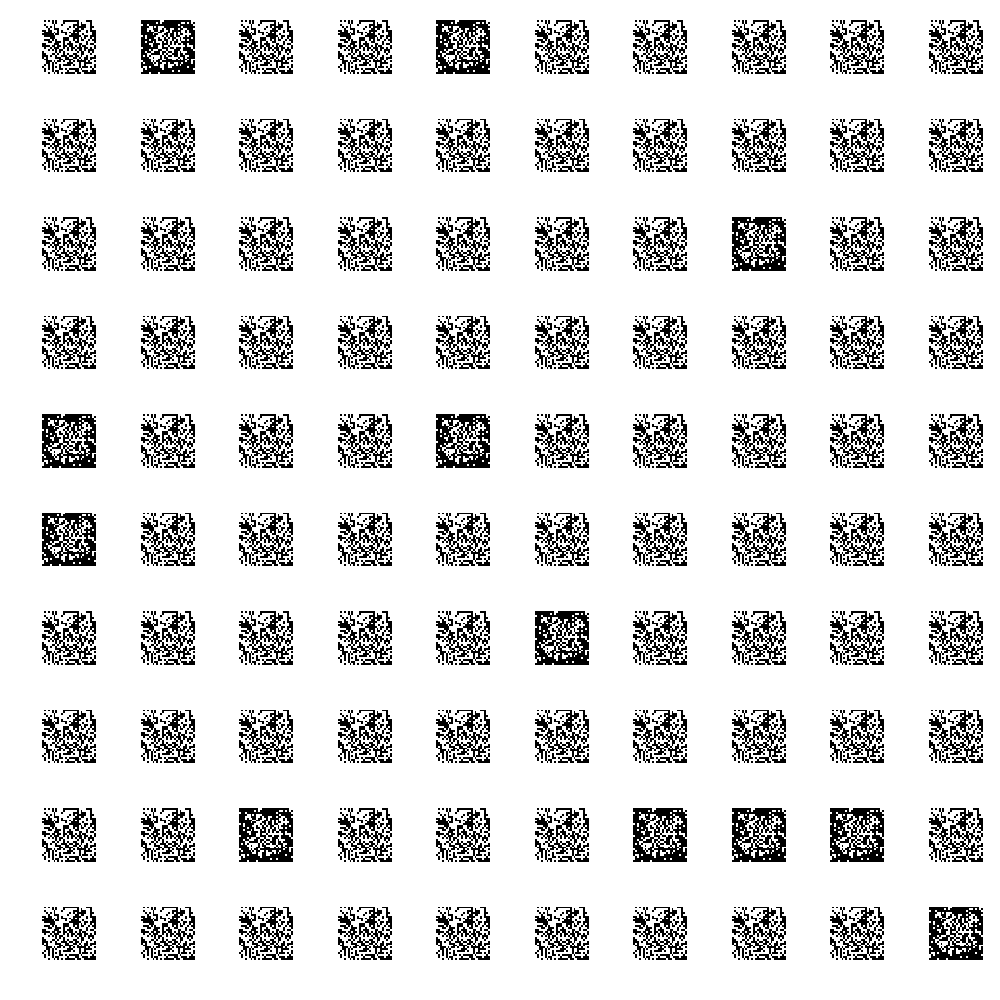


Image5: Image after 200 epochs for dropout + batch normalization strategy 1 with d = 0.2

Image5: Image after 200 epochs for dropout + batch normalization strategy 1 with d = 0.2

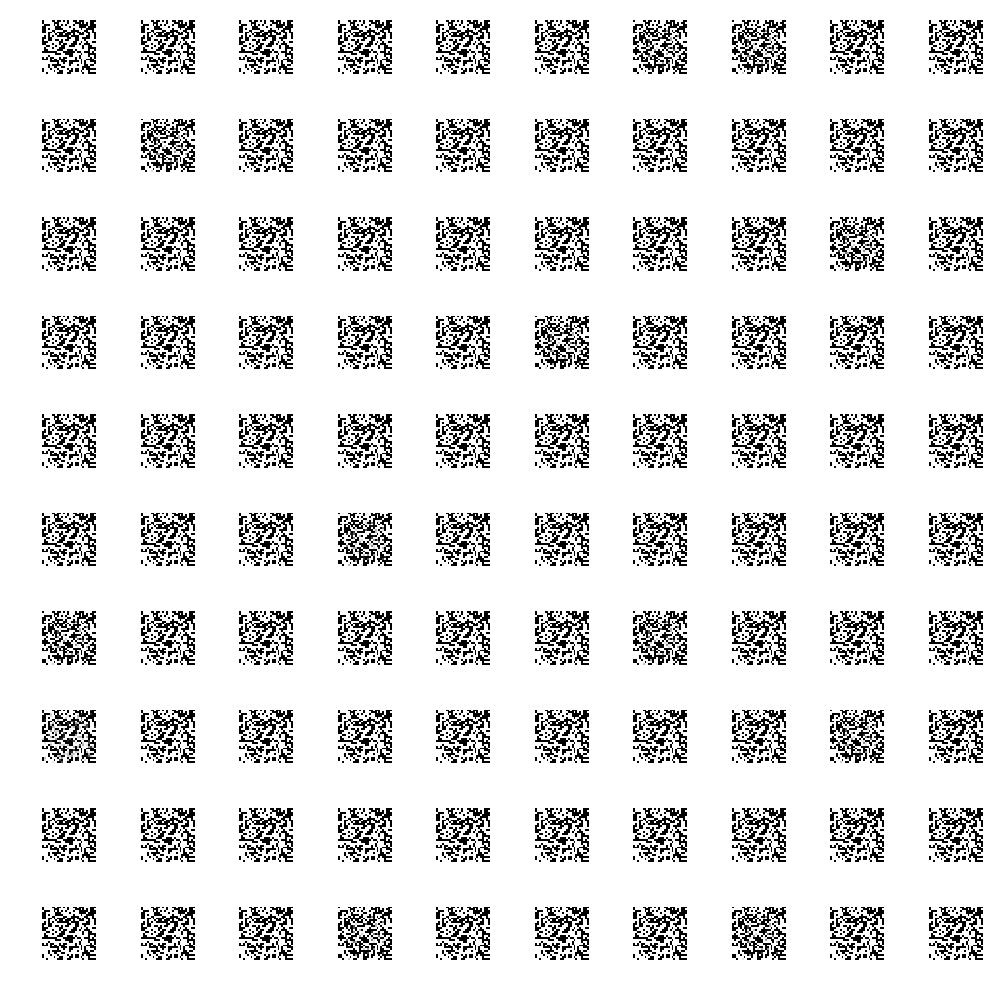
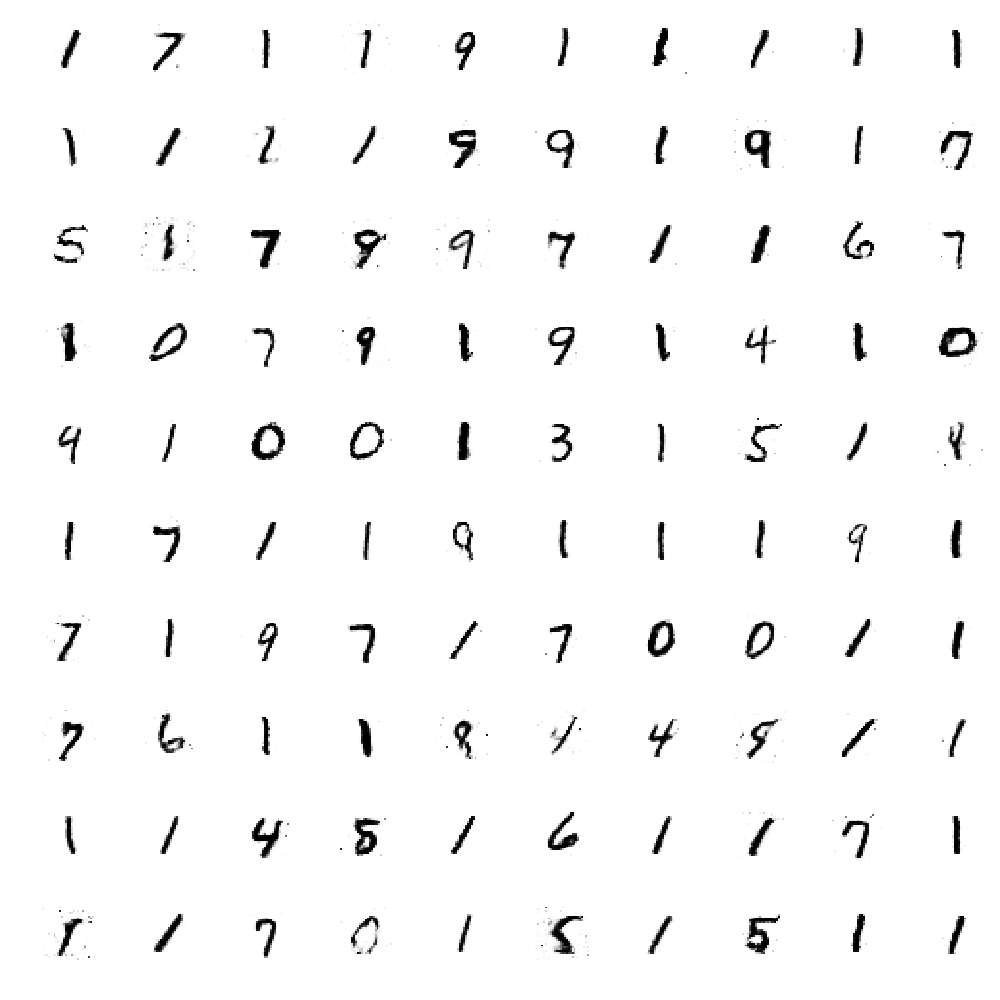
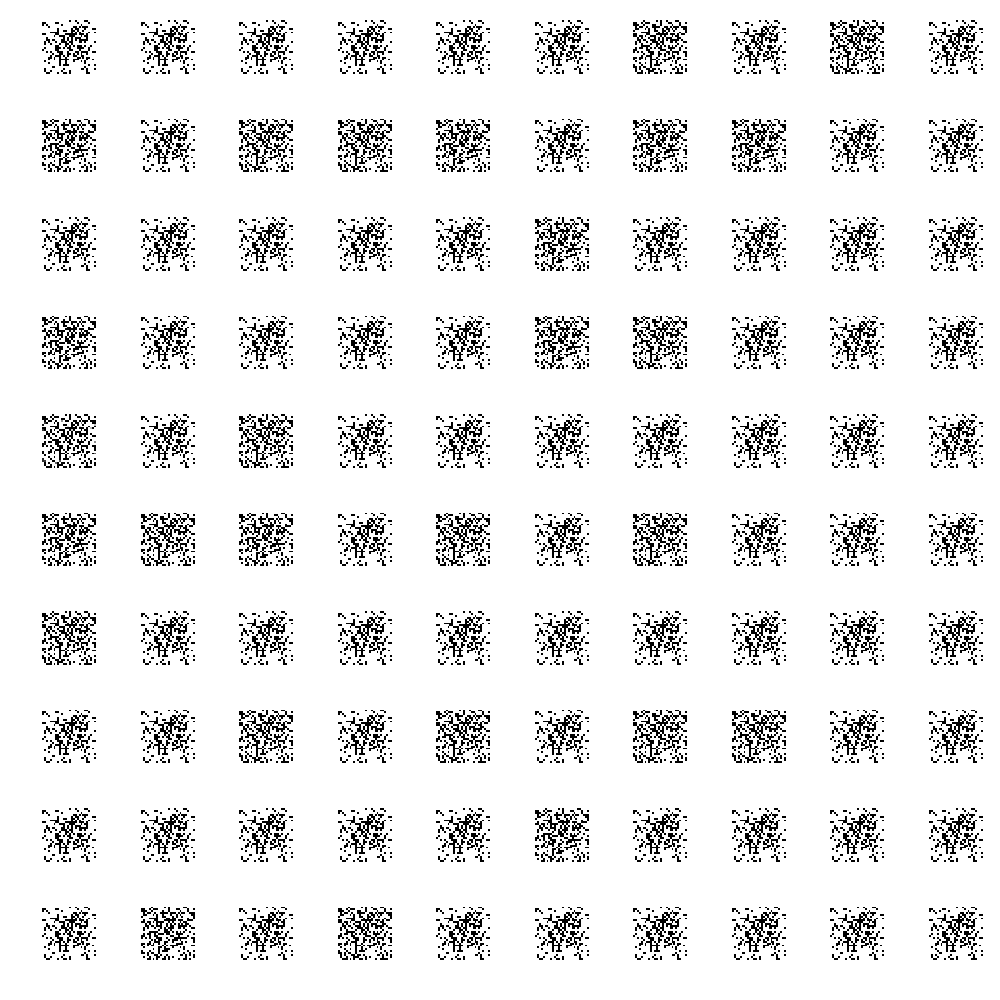


Image6: Image after 200 epochs for random variables + dropout strategy 1 with d = 0.2

Image5: Image after 200 epochs for random variables + dropout strategy 2 with d = 0.2

PS. - All the loss tables can be found under the link https://github.com/anuragdutt/Dropout\_DL\_TeamD/tree/master/src/dropout\_gan