Assignment Report # 3

Chaitra Jambigi (SR 16951)
DS 265 Deep Learning for Computer Vision

June 9, 2020

1. **Recurrent Neural Networks (RNNs) for text generation**

   (a) **Data preprocessing**

      i. The given text files are read and the Unicode characters are converted into standard ASCII characters. So all the printable ASCII characters present in string library of python and preserved. Rest of the artefacts are removed.

      ii. Also the case of characters is maintained as it is as in the text. This is because in the document, there are many nouns and other words which need to start with a capital letter, for example name of a person. So the model has to learn the naming pattern and the grammar correctly hence all punctuation's and character case is kept intact.

      iii. Around 91 distinct characters are present in the vocabulary.

   (b) **RNN Implementation**

      i. Character level RNN is implemented from scratch. An RNN class is created with forward and backward methods.

      ii. Each character is represented as a one hot vector of size of vocabulary.

      iii. Also a sampling method is implemented which takes as input a sequence of characters based on which the hidden states are computed and new characters are predicted. So this part does not contribute in loss calculation, it is just used to keep a check on how the training is progressing, how meaningful the characters are getting generated. So after every 100 iterations, a small input text is fed and generated text is printed.

      iv. To maintain randomness, we have cherry picked few words from the corpus, like 'Harry', 'Ron', 'Hermione', 'Dumbledore', 'Hogwarts' and few more. We then give only half of these words as inputs and see how well the model can finish the word and generate the text.

   (c) **Hyperparameter tuning**

      i. 1 Layer is used in the current implementation.

      ii. To fix the hyperparameters, we experiment with various values of hidden nodes, leraning rates, time step lengths and temperature values.

      iii. The hidden nodes tried are 100 and 256.

      iv. The learning rates tested are 1e-1, 1e-2, 1e-3 and 1e-4

      v. Also for different time step lengths like 20, 30, 40, 50, experiments are carried out.

      vi. So basically we train first only only 1 text file for 30000 iterations.

vii. To compare models we see two things, **Loss plot** and **generated text**. Smooth loss is used to plot the loss (Followed by Andrej Karpahy's blog post) which helps us reduce the noise and see the decreasing trend clearly.

viii. The accuracy is computed as the number of correct character predictions in given sequence length. The accuracy values did not provide much information about which model is good. The accuracy was very fluctuating and almost similar for all models and is around 50% accuracy (Figure 1. Exponential moving average values are plotted for Accuracy as values were noisy.

ix. We observe that this accuracy is correct because at character level, if model is predicting around 80 or 90% correct next characters, it's as good as it has memorised the words. For example, 'Ha' can be 'Harry','Hagrid','Happened', 'Has' or many different words. So we cannot ask the model to give very high accuracy for each character. However this can be ensured that if 'Harr' is present, next character in generated text should be 'y'.

(d) **Comparison based on Loss**

i. The training is done for 30000 iterations, but loss values are plotted after every 100 iterations, so total we have 300 loss values in the plot.

ii. For learning rate 1e-4, the generated text had very few correctly spelled words. Also the loss was very high for 100 nodes and 20 as time step length. For lr as 1e-1, it was repeating the same few characters again and again, so it was not generalising well. Thus we consider lr 1e-2 and 1e-3 for which we get some loss comparisons.

iii. Figure 1 shows loss plots for hidden nodes as 100 and 256 with time step as 20, 30, 40, 50. We can see see clearly for any number of hidden nodes, the loss decreases as the time step increases. Between 100 nodes 30 timestep and 256 nodes 30 time step, 256 one is having lesser loss. Same argument for 100, 40 and 256, 40. Within 30 and 40 time steps, 40 is having lesser loss.

iv. So as the number of hidden nodes is increased from 100 to 256, loss decreases. So by looking at loss we can say having more nodes and more time steps is beneficial. Also for lr as 1e-2 we get lower loss than 1e-3.

(e) **Comparison based on generated text**

i. Now we also see the generated text by few of these configurations.

A. lr: 1e-2, Nodes: 100, Time step: 20
Initial Text = "Dumbledore again," Generated text = nick-?orgir?" the sat in I me!win Harry tearfyod't but im taded't Is teent's reefing."
To'd. Tersed oo hele exterce fithed Marry. "Indbetterst asd to theve toughin callets fulfo fidn Ser," slowers b

B. lr: 1e-3, Nodes: 512, Time step: 50
Initial Text = "Dumbledore again," Generated text = Vo botresley,.."Enytay fingperodded. If soid-det s licken vo busker shap woodhy wher hith tuisted be huy dors cxointloag to souys?edewh thamted Dfoyen, dtind he was vecesedlale or imehtarred said? fl

C. lr: 1e-3, Nodes: 256, Time step: 50

Initial Text = Avad Generated text = owas sumped a sal teant tree on the meworns. "What wholing a seemed coures, Pate lexs fulder to revers to cemping at he sacat to see ow your of Hermione say.!" Harry pack a wall the sad in the Dumble

D. lr: 1e-3, Nodes: 256, Time step: 20
Initial Text = Pott Generated text = er. "Hol dore blans-" she said," she see, that of itno that need can gome!" "Whink worved, said Harry." The yout his eyes ment as the toba pocks; Dust," said Gistanderipsiots saye. They he pelling to

E. lr: 1e-3, Nodes: 100, Time step: 40
ing," said Ron there all was in Xelove arounly Hermione. "Why prosing under dis wand yough there when a worsed me a rame for I Polden Dark tegully begars. Weded him. "Howed all really. "The think blow

F. lr: 1e-3, Nodes: 100, Time step: 20
Initial Text = "Dumbledore again," Generated text = leffelly Kelly steep. "And Rons!" said Ron like and boldead loster." Ceaprully looked as decrutem, dretturing. As encered as stared retoldure was pitter to his feellors were all she." "We she think th

ii. We can see that with lr-2 the text is having many spelling mistakes and words are not getting learnt. Also with 512 nodes and time step 50 although loss is very less, but the generated text doesn't have correct words. With lr 1e-3, 100 nodes and 20 time step, more words are generated correctly. So we cannot completely rely on models with less loss.

iii. We can say that Loss is basically the Softmax of logit which is present at ground truth index, so if loss values are becoming very less, then similar to temperature logic, the probabilities will be more or less similar giving some random results with spelling mistakes.

iv. Thus to fix a model we need to look both at loss curve and generated text.

v. Based on all these observations we fix our hyperparameters as, 256 hidden nodes, time step 20, lr 1e-3. Tanh activation is used for hidden nodes and for final output softmax is applied. SGD is used as optimizer to update the weights. Also the gradient values are clipped between -5 to 5 to mitigate the exploding gradients problems.

(f) **Temperature effect on predictions**

i. The logits or output of the network before applying softmax can be divided by a hyperparameter Temperature.

ii. Using a higher value of temperature (temperature >1) will result in logits being smaller in value, so the probabilities for all characters will be more or less similar. Thus the chosen characters will be more diverse having more variety, but it may not be always correct leading to spelling mistakes.

iii. Using a lower temperature value (temperature<1) will result in logits being higher in value, so the higher probable character will be pushed more towards high value, and low probable character will be pushed more to lower side. Thus the exact characters as in training data will be picked leading to less randomness and less diversity in generated text.

iv. Random 500 character text is generated for various temperature values like 0.4, 0.6, 0.8, 1, 1.2, 1.4 and generated text is compared. The file Temperature.txt shows the text.

v. It can be easily observed that as temperature value increase like 1, 1,2, 1.4 the kind of words being generated are not common words and some other random word. So these sentences don't make much sense and have spelling mistakes.

vi. When temperature is small like 0.4 or 0.6 the generated text mostly repeats words and almost all words are from the input training set creating less variations.

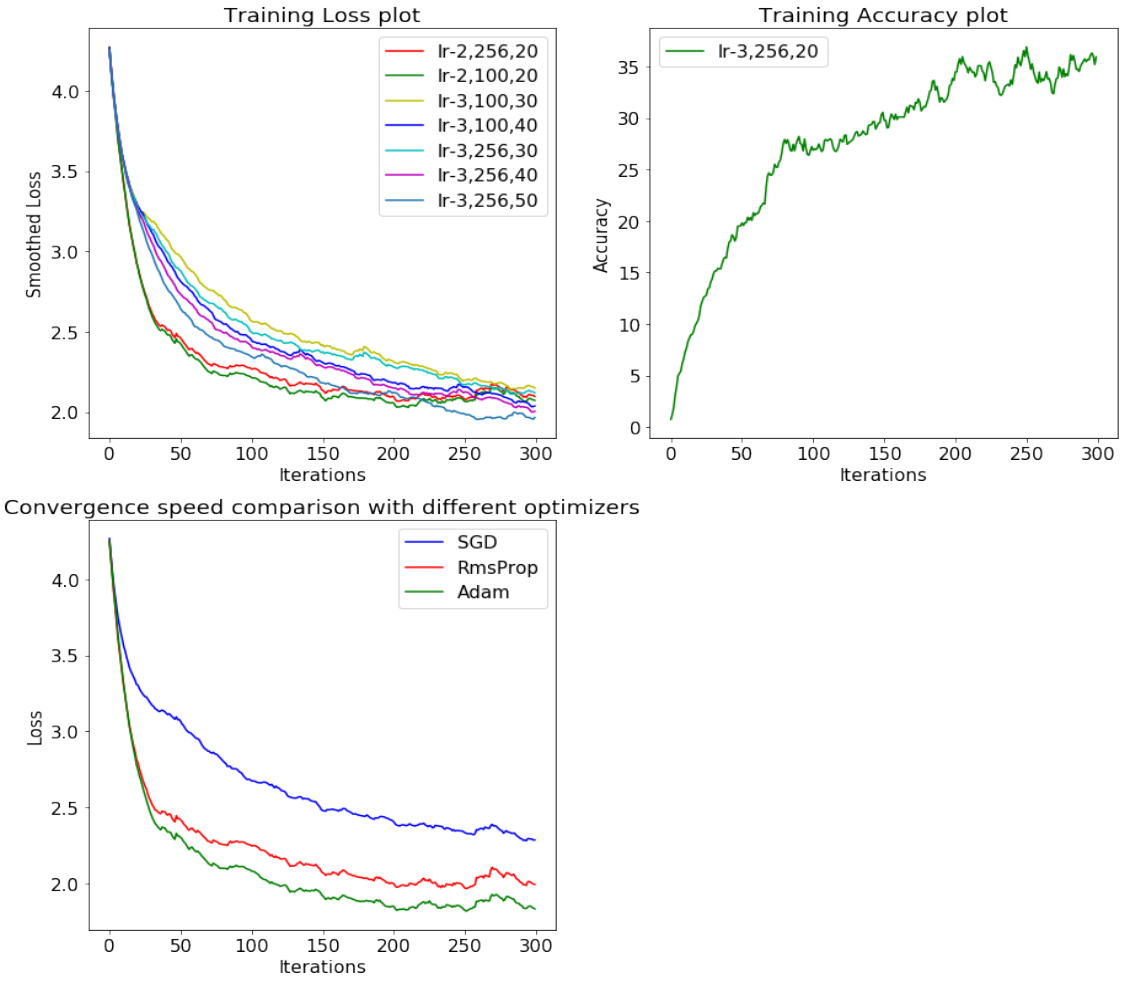

Figure 1: Training loss and accuracy plots

(g) **Text generation** A thousand character text is generated and saved as generated_text.txt file. We have used temperature value as 0.8 to generate this text. Random initial sentence inputs are given instead of initial zero activations.

(h) **Convergence speed by varying Optimizers** We have implemented both RmsProp and Adam optimizer in the RNN class in code. It can be seen from Figure 1 that SGD is the slowest to converge, followed by RmsProp and Adam being the fastest.

(i) To generate sensible words using SGD it took minimum 100K iterations and with Adam, it took minimum 30K iterations, to get good sensible results in which words are getting framed properly.

2. **Generative Adversarial Networks**

(a) **Gan Training**

i. We have trained a GAN to generate MNIST images. Pytorch framework is used for this question.

ii. DCGAN Architecture is used. The network architecture for Generator and Discriminator is given in Appendix section in Figure 11 and Figure 12 respectively.

(b) **Loss and accuracy plots**

i. Tensorboard plots for Discriminator loss for Real and Fake samples and Generator loss is shown.

ii. Also Discriminator accuracy for Real and Fake samples is shown.

iii. Initially, the Discriminator loss and Generator loss have lot of fluctuations. If Discriminator loss increases then gen loss decreases and vice versa.

iv. As the training progresses the variance in the loss is present. As the network reaches stability, the Generator loss lies around an average of 2 and Discriminator loss around average 0.4.

(c) **Hyperparameter comparison**

The hyper-parameters used for comparison are Number of layers in Discriminator used (with dropout changes), learning rate, Number of layers in Generator and Weight initialisation.

i. **Number of layers in Discriminator**:

A. 2 different configurations with Discriminator are tried. First one has 3 Conv layers in Discriminator and the second one has 4 Conv layers. As we increase the number of layers from 3 to 4, dropout layer is added with increasing dropout values to see how increasing the Discriminator capacity affects the learning process.

B. We try to analyse by changing the Discriminator capacity how the convergence speed varies.

C. So for comparison, we have six candidates, first one with 3 layers, second one with 4 layers and dropout 0, third with 4 layers and dropout 0.2, then with dropout 0.4, 0.6 and 0.8. The figure 12 shows Discriminator architecture with 4 layers and Dropout layer.

D. Figure 2 shows the reconstructed image for all the six cases at epochs 25, 50, 100, 150 and 200.

E. **Observations**

- We can see that on 25th epoch itself, the reconstructed output with 4 layers is better than that with 3 layers. Also on 25th epoch, 4 layer dropout 0 outperforms all other configurations.
- At epoch 50, 3 layer output is still unclear. 4 layer dropout 0.2 and 0.4 also start producing better results.

- By epoch 100, 4 layer dropout 0, 0.2 and 0.4 have good results. Dropout 0.6 and 0.8 and 3 layered one suffer badly.
- Thus we can say that having 4 layers without any dropout produces plausible results with very fast convergence at around 50-100 epochs.
- Figure **??** shows the loss plots for these six cases. We can see that in first row, the red graph is for 3 layered network and gray for 4 layered dropout 0. As the network capacity increases the variance also increases in both the Generator and Discriminator losses.
- In second row the variance decreases as network capacity is decreased. In third row, and fourth row as the drop is furthur increased, the variance again decreases.
- Thus we can infer that if the network capacity is less, then the variance in losses is also less throughout the training. Thus having high variance is indication of good training.

ii. **Learning rate**:

A. We have tried to train the network fixed in previous step (4 layers dropout 0) with four different learning rates, 0.00001, 0.0001, 0.0002, 0.01.

B. Figure **??** shows the image generation results at the end of training for each of the learning rates.

C. It can be observed that when the learning rate is 0.00001 the learnt representations are not so good. Almost all shapes do not have structure.

D. Also lr 0.01 lacks proper structure and has poor generation quality.

E. lr 0.0001 and 0.0002 give better results and lr 0.0002 gives best results of all.

F. Thus the chosen lr is 0.0002.

iii. **Number of layers in Generator**:

A. The 100 dimensional latent vector has to be mapped to a feature map and then passed through conv blocks. So two configurations of Generator are analysed, one with 3 transposed conv layers and another four.

B. In the 3 layered one, 100 dimensional vector is reshaped as 128x7x7 and then enlarged to 1x28x28.

C. In 4 layered architecture, 100 dimensional vector is reshaped as 128x4x4 and then enlarged.

D. It can be seen in Figure **??** that around 150 epochs we are getting good results. So increasing Generator capacity also leads to better learning.

iv. **Weight initialisation**:

A. In the original DCGAN paper they mention weights should be initialised as Normal distribution with mean 0 and variance 0.02 and to use Batch Norm before all activation functions.

B. The reason these constraints need to be imposed is that the input to activation function should not go in saturation region.

C. Along with normalising inputs using batch norm, proper weight initialisation is also needed to keep the inputs in linear region. Another work in this direction which tries to overcome exploding or vanishing gradients is using Xavier initialisation.

D. So instead of Normal with mean 0 and deviation 0.02, Xavier initialisation is tried to see how it performs. Figure **??** shows that the results are good however normal initialisation outperform Xavier initialisation.

(d) **Traversing in the latent space of GAN**

i. Walking on the manifold of learnt latent space can help us see if the network has memorised the digits in training or it has learnt semantic information. If traversing the space creates new objects showing smooth transitions from one image to other, then we can say that the model has learnt relevant and interesting representations.

ii. Figure **??** shows the traversal of learnt latent space. For all further analysis the Generator shown in Figure 11 and Discriminator shown in Figure 12 with Dropout 0 is used.

iii. Randomly 10 points in the 100 dimensional space is sampled as input from Noise. Between each of the points, around 100 new points are linearly interpolated. From all these resulting points, new images are generated. So we plot around 900+ images.

iv. It can be seen that there are smooth transitions as we go from 1 digit to other.

v. Also different shapes of 4, 7, 9 is learnt.

vi. Slowly new objects are getting added as we traverse and it indicates that the model has not memorised the 10 digits, instead it has learnt a smooth manifold.

(e) **Testing accuracy of generated images on trained classifier**

i. The classifier having mentioned architecture is trained on MNIST training data.

ii. The training accuracy is around **99.5**% and test accuracy of **98.8**%

iii. 10,000 images are generated and the distribution of each class is measured by plotting a Histogram as shown in Figure **??** Approximately each digit has 1000 samples generated. With exact numbers, 0-1035 images, 1-1076 images, 2-987, images, 3-1027 images, 4-935 images, 5-958 images, 6-1033 images, 7-1061 images, 8-871 images, 9-1017 images

iv. 30 randomly selected images by GAN are shown in Figure **??** along with their predicted labels.

v. It can be seen that all the digits are getting classified as a person with normal vision will do. Also different shapes of 8 is getting classified correctly as 8, different and very thin shapes of 9 also are getting classified correctly as 9. Also a distinction is being done between 7 and 3 in third row by just a small kink at the middle for 3.

vi. Thus we can say that the generated digits are having similar distribution as the MNIST input data and hence the classifier performs well even on generated image samples.

(f) **Architectural changes**

i. We tried to create more diverse images for each of the digits. Two approaches were tried, but both didn't scale well.

ii. Now from the 100 dimensional noise, the digit is getting generated. But while generating, we do not have any control on the height, shape slant of the digit.

iii. We can create more diverse images if we can encode these shape, slant, height and width properties separately and the identity of digit separately.

iv. Approach 1

A. Since it is an unsupervised training with no label, we cannot directly enforce any embedding to capture identity information.

B. We propose to take two noise inputs ($z1, z2$) each having dimension 50, and before giving to generator, we concatenate them and make a 100 dimension input and pass to generator.

C. We try to impose constraints such that $z1$ is discrimninative and $z2$ encodes shape information. So that at test time with mixing different combinations, we can generate all shapes for all digits.

D. We train a classifier whose penultimate layer is 50 dimensional vector, and the last layer gives the softmax probabilities for 10 classes. We assume that the penultimate 50 dimensional vector is discrimninative, and we try to match the distribution of $z1$ with this 50 dimensional vector.

E. We train the network for around 100 epochs till it starts generating good images, then we add another KL divergence loss to the training. We pass this generated image through our classifier and get the 50 dimensional vector and minimise the KL loss between output and $z1$. However the generated images were getting corrupted so further analysis was not done.

v. Approach 2

A. Generally, after the model gets trained the latent space visualisation (like TSNE) shows that each of the digits are clustered separately in 2D space.

B. We try to impose the constraint that these clusters are far apart and each have a fix mean which we give.

C. So now we have 10 mean values (0,0.5,1,1.5,2,2.5,3,3.5,4,4.5) and randomly the 100 dimensional noise is sampled from a Gaussian having any of the means, with fixed variance of 1, and each mean corresponds to 1 label.

D. So we try to enforce that the noise coming from 0 mean must generate a 0, and from 0.5 must generate a 1. We do this by passing generated image from pre trained classifier and adding Cross entropy loss on that.

E. However this could not be implemented totally because of time constraints.

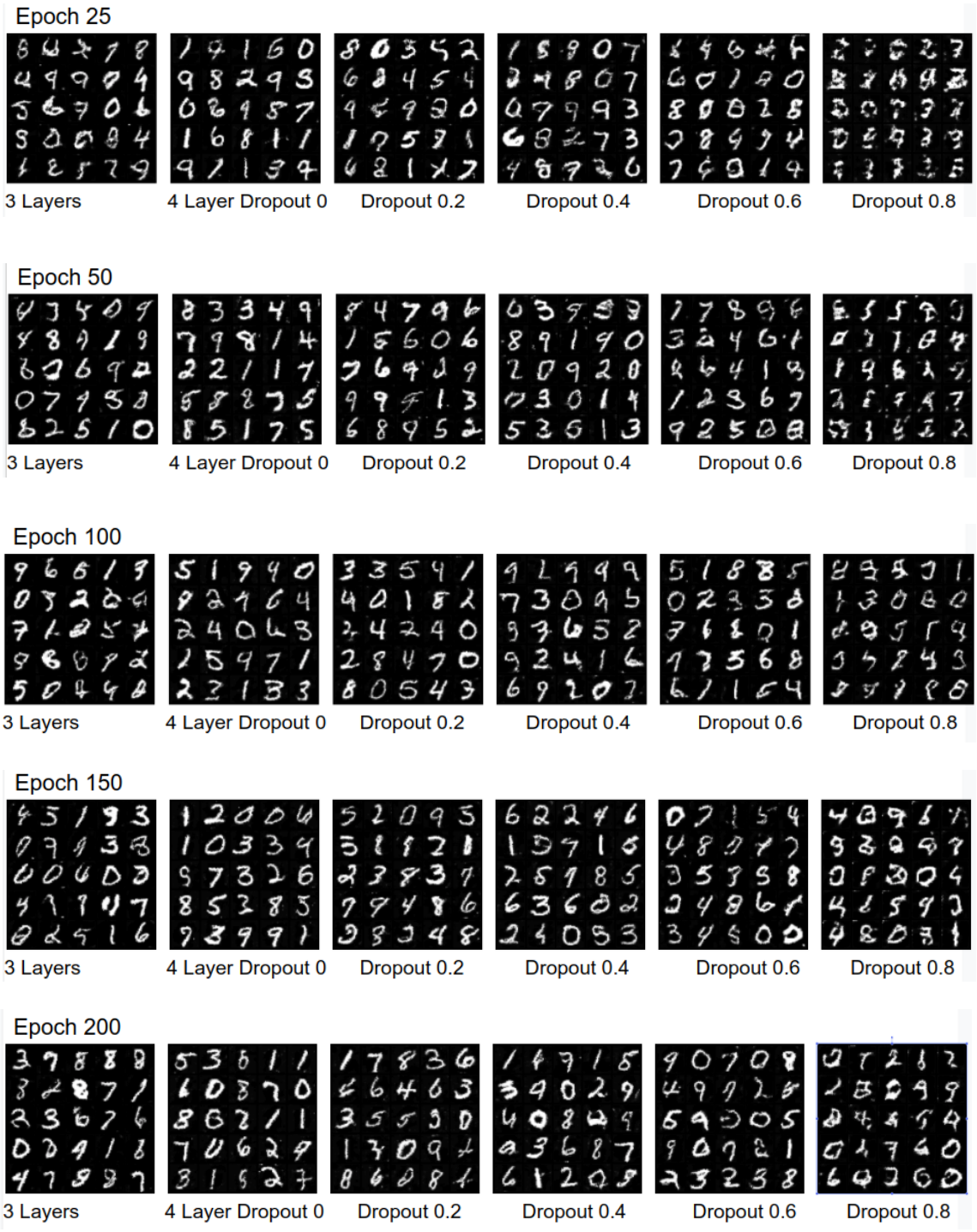

Figure 2: Convergence speed for different architectures

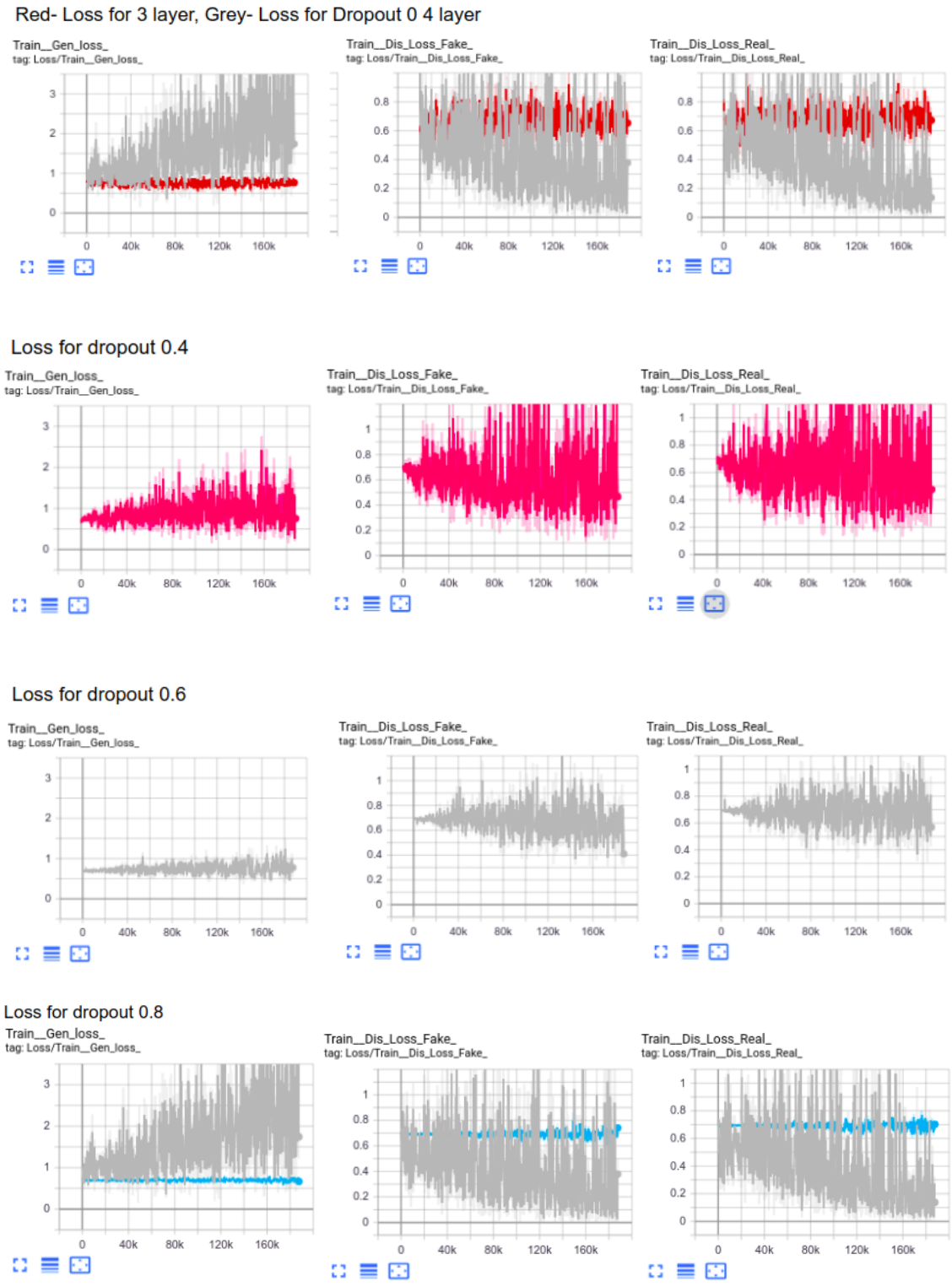

Figure 3: Tensorboard plots showing Variance in Loss values

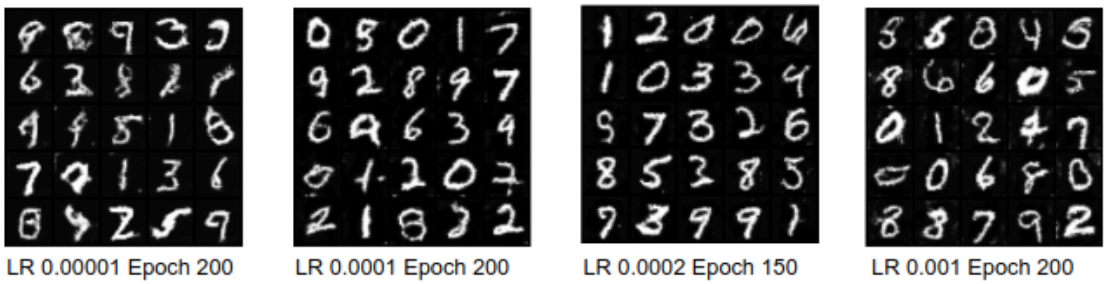

Figure 4: Image generation results for different learning rates

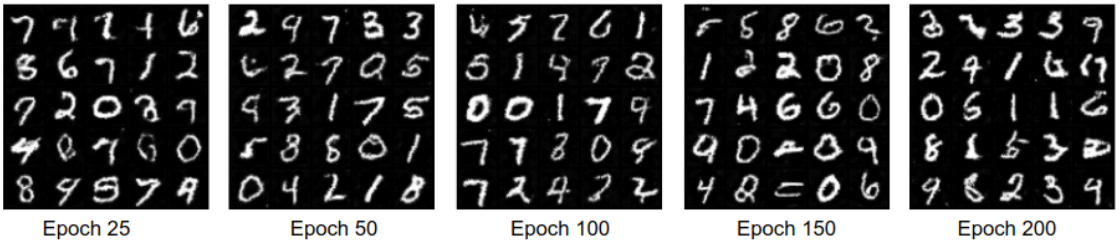

Figure 5: Image generation for Generator with 4 layers

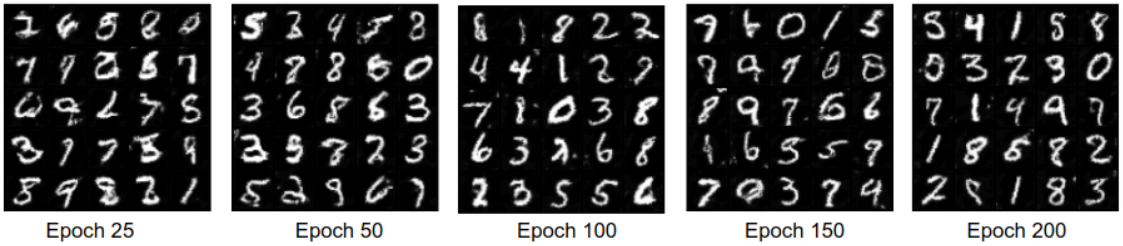

Figure 6: Image generation using Xavier initialisation

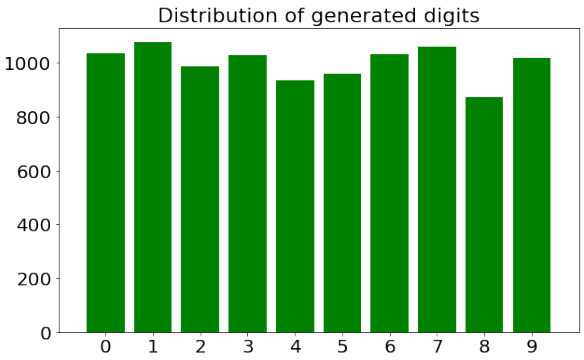

Figure 7: Distribution of generated digits

Figure 8: Classifier prediction on random generated samples

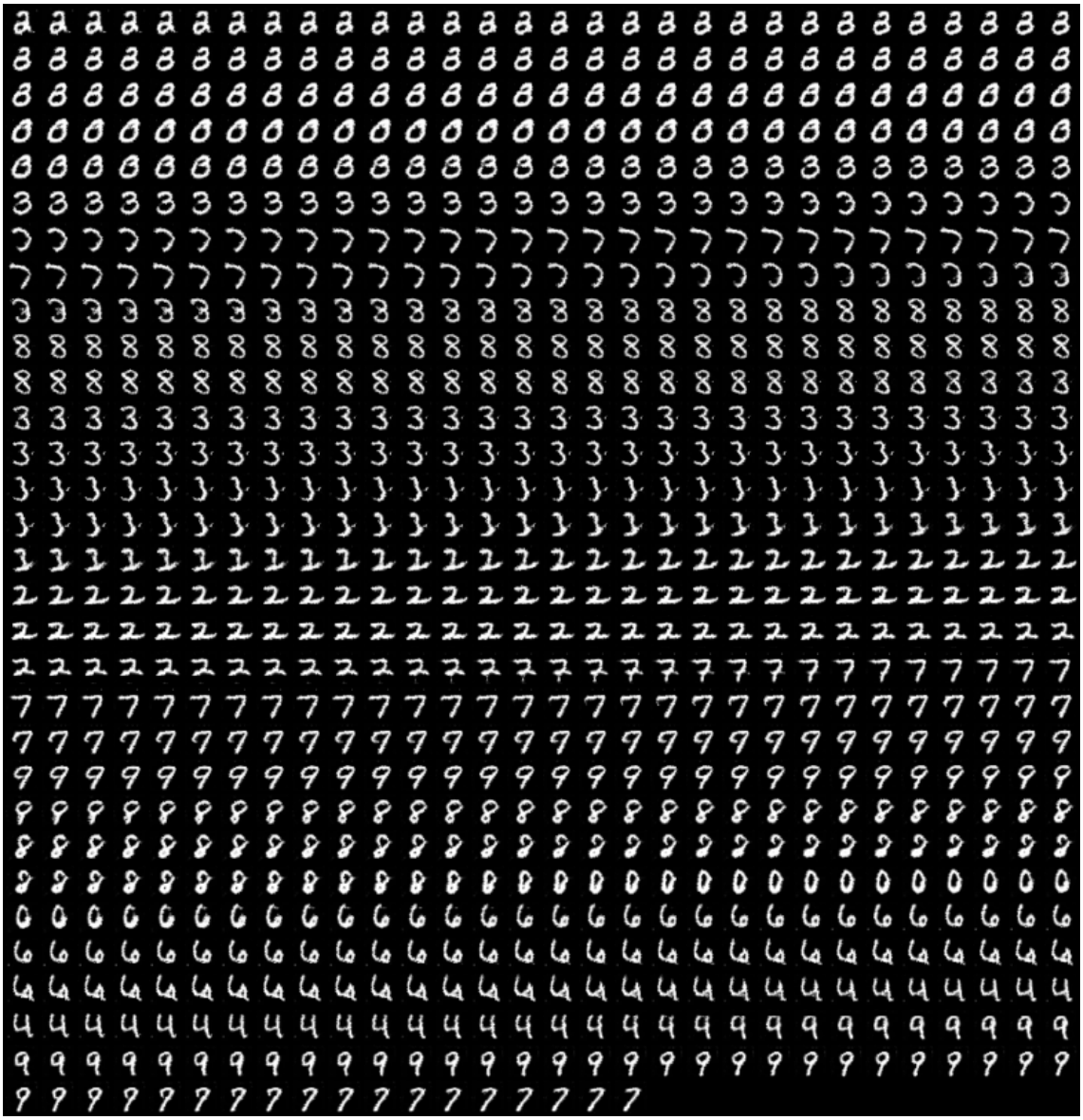

Figure 9: Traversing the latent space in GAN

3. **Variational Auto-encoders**

   (a) **Training a VAE to generate MNIST images**

      i. Keras framework using Tensor flow as the back end is used to train the VAE.

      ii. The network has three modules, Encoder, Decoder and the VAE which combines Encoder and Decoder.

      iii. The Encoder takes as input the MNIST training images and outputs the latent space representation. The latent dimension is chosen as 2 dimensions.

      iv. The Mean and Variance is obtained from the Encoder. This is then reparametrized by sampling an epsilon from Zero mean, Unit variance Gaussian and then scaling it by

the predicted variance and shifting it by the mean. By this trick the back propagation can now be done, which would be difficult if the values were directly randomly sampled from the Gaussian having mean and variance as predicted by the network.

v. Adam optimizer is used with a learning rate of 0.0002, batch size of 128 and trained for 30 epochs.

vi. The Decoder architecture is similar to the Generator used in previous question of GAN. The Encoder is thus the mirror image of this Decoder. The architecture table is shown in Figure **?? ??**

(b) **Comparison of VAE with GAN**

i. Figure 10 shows the latent space traversal in VAE model. Trials were done with different latent dimensions like 100 and 2.The best results were obtained using 2 latent dimensions. Also this is what is followed in most of the literature works.

ii. This latent space traversal is plotted in the same way as GAN traversal where 10 points are selected at random from the output of Encoder on test data. Between each points, over 100 points are linearly interpolated and together around 900+ points are gathered.

iii. These points are then passed through Decoder to get the images. These images are plotted which show smooth transitions from one digit to other which shows that the VAE model has not memorised the representations. It has effectively learnt the manifold.

iv. When comparing the latent space of GAN and VAE, we can see that the GAN generated images are more realistic as compared to VAE.

v. Also the images generated by GAN capture more minute changes as they traverse in the latent space. Incrementally a new part gets added in the generated image when we see traversal in GAN. However with the VAE traversal, although different digits are getting generated, but the changes are less smooth and more abrupt in comparison to GAN.

vi. Thus using GANS we can generate more different varieties of digits as compared to VAE.

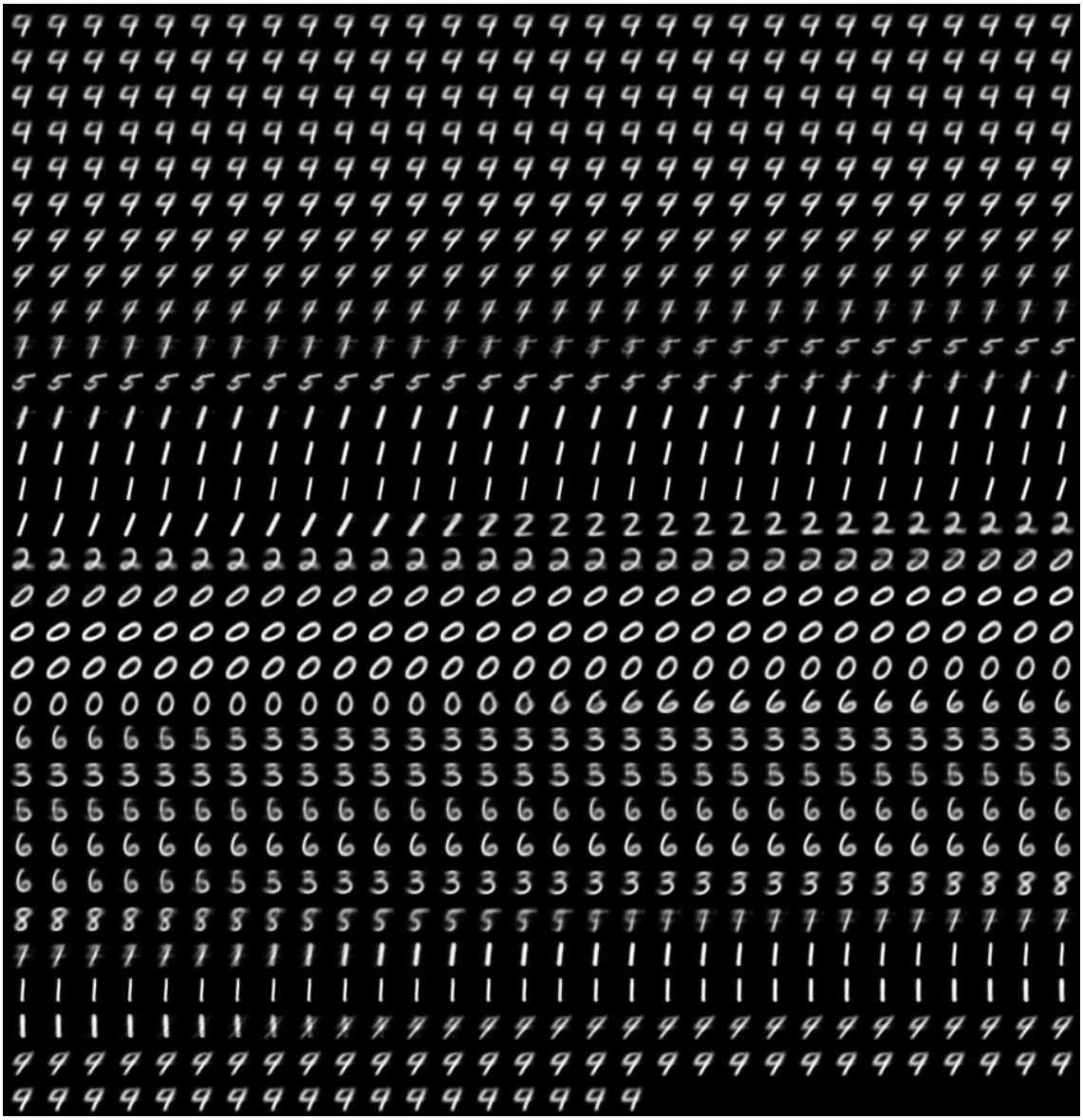

Figure 10: Traversing the latent space in VAE

APPENDIX

Layer (type)	Output Shape	Param #
Linear-1	[-1, 1, 6272]	633,472
BatchNorm2d-2	[-1, 128, 7, 7]	256
ConvTranspose2d-3	[-1, 64, 14, 14]	131,136
BatchNorm2d-4	[-1, 64, 14, 14]	128
ReLU-5	[-1, 64, 14, 14]	0
ConvTranspose2d-6	[-1, 64, 28, 28]	65,600
BatchNorm2d-7	[-1, 64, 28, 28]	128
ReLU-8	[-1, 64, 28, 28]	0
ConvTranspose2d-9	[-1, 1, 28, 28]	577
Tanh-10	[-1, 1, 28, 28]	0

Figure 11: Architecture for GAN Generator

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 16, 14, 14]	160
LeakyReLU-2	[-1, 16, 14, 14]	0
Dropout-3	[-1, 16, 14, 14]	0
BatchNorm2d-4	[-1, 16, 14, 14]	32
Conv2d-5	[-1, 32, 7, 7]	4,640
LeakyReLU-6	[-1, 32, 7, 7]	0
Dropout-7	[-1, 32, 7, 7]	0
BatchNorm2d-8	[-1, 32, 7, 7]	64
Conv2d-9	[-1, 64, 4, 4]	18,496
LeakyReLU-10	[-1, 64, 4, 4]	0
Dropout-11	[-1, 64, 4, 4]	0
BatchNorm2d-12	[-1, 64, 4, 4]	128
Conv2d-13	[-1, 128, 2, 2]	73,856
LeakyReLU-14	[-1, 128, 2, 2]	0
Dropout-15	[-1, 128, 2, 2]	0
BatchNorm2d-16	[-1, 128, 2, 2]	256
Linear-17	[-1, 1]	513
Sigmoid-18	[-1, 1]	0

Figure 12: Architecture for GAN Discriminator

Model: "Encoder"

Layer (type)	Output Shape	Param #	Connected to
Input (InputLayer)	[(None, 28, 28, 1)]	0	
Conv_1 (Conv2D)	(None, 28, 28, 64)	640	Input[0][0]
BatchNorm_1 (BatchNormalization	(None, 28, 28, 64)	256	Conv_1[0][0]
Conv_2 (Conv2D)	(None, 14, 14, 64)	36928	BatchNorm_1[0][0]
BatchNorm_2 (BatchNormalization	(None, 14, 14, 64)	256	Conv_2[0][0]
Conv_3 (Conv2D)	(None, 7, 7, 128)	73856	BatchNorm_2[0][0]
BatchNorm_3 (BatchNormalization	(None, 7, 7, 128)	512	Conv_3[0][0]
flatten_22 (Flatten)	(None, 6272)	0	BatchNorm_3[0][0]
Dense_1 (Dense)	(None, 100)	627300	flatten_22[0][0]
Mu (Dense)	(None, 2)	202	Dense_1[0][0]
Log_sigma (Dense)	(None, 2)	202	Dense_1[0][0]
z (Lambda)	(None, 2)	0	Mu[0][0] Log_sigma[0][0]

Figure 13: Architecture for VAE Encoder

Model: "Decoder"

Layer (type)	Output Shape	Param #
Input (InputLayer)	[(None, 2)]	0
Dense_1 (Dense)	(None, 100)	300
Dense_2 (Dense)	(None, 6272)	633472
reshape_10 (Reshape)	(None, 7, 7, 128)	0
Conv_1 (Conv2DTranspose)	(None, 14, 14, 64)	131136
BatchNorm_1 (BatchNormalizat	(None, 14, 14, 64)	256
Conv_2 (Conv2DTranspose)	(None, 28, 28, 64)	65600
BatchNorm_2 (BatchNormalizat	(None, 28, 28, 64)	256
Conv_3 (Conv2DTranspose)	(None, 28, 28, 1)	577

Figure 14: Architecture for VAE Decoder