Assignment 1: Report

Ronak Dedhiya Dinesh (ronakdedhiya@iisc.ac.in; SR No: 06-18-05-19-52-21-1-20116)
Prashanth N Bhat(prashanthbn@iisc.ac.in; SR No: 04-03-06-19-52-21-1-20082)

September 28, 2024

1 Problem Statement

1. Implement a standard VAE with MSE loss for the likelihood on the following datasets
   (a) CelebA
   (b) dSprites

   Calculate the marginal likelihood scores for both the case using the procedure outlines in Appendix D in this paper. Plot 100 random samples of the generated images in a 10x10 grid for both the datasets post training.
2. Implement a naive Deep-Convolutional Generative Adversarial Network (DC-GAN) on theBitmoji Dataset. Plot a 10x10 generated image grid post training. Plot the training loss curves for both Generator and Discriminator functions and comment on their behavior. Sample 1000 images from the generator and estimate the Frechet Inception Distance between the generated 1000 samples and 1000 samples from the real data.

2 Variational Auto Encoder

2.1 Model Structure

VAE is constructed with 5 layer CNN encoder q(z|x) with μ & σ as the output. A reparametrization trick is applied to sample from standard Gaussian and construct the latent z as an input to the decoder. The decoder p(x|z) is constructed with 5 layer ConvTranspose layer to generate the image output. The encoder architecture includes 5 layer of blocks [Conv, Batch Normalization, LeakyRelu] while decoder includes 5 layer of blocks [ConvTranspose, Batch Normalization, LeakyRelu]. Entire architecture is shown in Figure 1 with encoder channel dimensions as [16,32,64,128,256] and vice versa for decoder with 512 as latent dimensions. The described architecture is used for both the dataset CelebA and dSprites, however, for dSprites dataset (binary dataset) we experimented with light weight 2 layer MLP encoder - decoder VAE architecture and we obtained similar results. Inputs were resized to (64,64) and converted to normalized tensor.

2.2 Loss Function

Training VAE includes combination of two losses: a) MSE loss between input & reconstructed Input b) KL Divergence between q(z|x) and p(z) which is reduced to

$$D[Q(z|x) \parallel P(z)] = \frac{1}{2}[-\sum_i (log\sigma_i^2 + 1) + \sum_i \sigma_i^2 + \sum_i \mu_i^2] \tag{1}$$

Encoder

Conv2d-1	[-1, 16, 32, 32]	448
BatchNorm2d-2	[-1, 16, 32, 32]	32
LeakyReLU-3	[-1, 16, 32, 32]	0
Conv2d-4	[-1, 32, 16, 16]	4,640
BatchNorm2d-5	[-1, 32, 16, 16]	64
LeakyReLU-6	[-1, 32, 16, 16]	0
Conv2d-7	[-1, 64, 8, 8]	18,496
BatchNorm2d-8	[-1, 64, 8, 8]	128
LeakyReLU-9	[-1, 64, 8, 8]	0
Conv2d-10	[-1, 128, 4, 4]	73,856
BatchNorm2d-11	[-1, 128, 4, 4]	256
LeakyReLU-12	[-1, 128, 4, 4]	0
Conv2d-13	[-1, 256, 2, 2]	295,168
BatchNorm2d-14	[-1, 256, 2, 2]	512
LeakyReLU-15	[-1, 256, 2, 2]	0
Linear-16	[-1, 512]	524,800
Linear-17	[-1, 512]	524,800

Decoder

Linear-18	[-1, 1024]	525,312
ConvTranspose2d-19	[-1, 128, 4, 4]	295,040
BatchNorm2d-20	[-1, 128, 4, 4]	256
LeakyReLU-21	[-1, 128, 4, 4]	0
ConvTranspose2d-22	[-1, 64, 8, 8]	73,792
BatchNorm2d-23	[-1, 64, 8, 8]	128
LeakyReLU-24	[-1, 64, 8, 8]	0
ConvTranspose2d-25	[-1, 32, 16, 16]	18,464
BatchNorm2d-26	[-1, 32, 16, 16]	64
LeakyReLU-27	[-1, 32, 16, 16]	0
ConvTranspose2d-28	[-1, 16, 32, 32]	4,624
BatchNorm2d-29	[-1, 16, 32, 32]	32
LeakyReLU-30	[-1, 16, 32, 32]	0
ConvTranspose2d-31	[-1, 3, 64, 64]	435
Sigmoid-32	[-1, 3, 64, 64]	0

Figure 1: VAE Model Architecture

$$TotalLoss = MSELoss + \beta * D[Q(z/x) \parallel p(z)] \quad where \beta = 1 \tag{2}$$

2.3 Results

Data is split into train and test set and trained using the proposed architecture. Model is trained with the combination of losses for 20 epochs as shown in Figure 2.

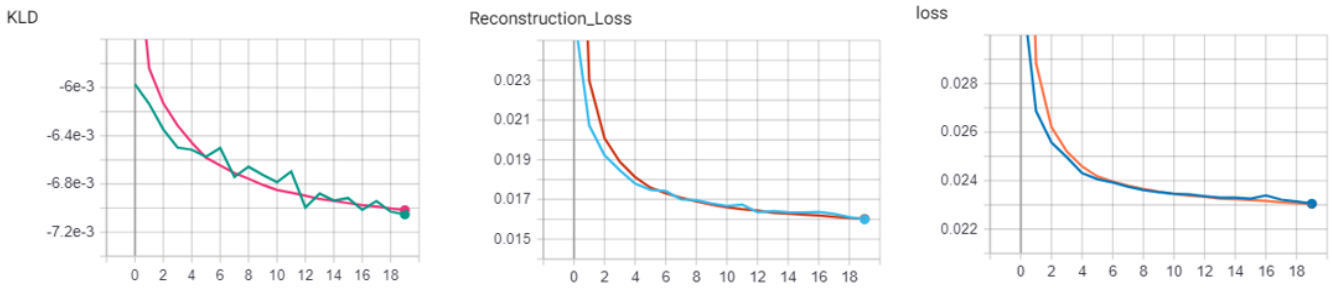

Figure 2: Train and Test loss curve for KLD, Reconstruction and total Loss

We found $\beta = 1$ has very hard regularization preventing the decoder to reconstruct the output. We thus trained model with varying β = [2.5e-1,2.5e-2,2.5e-3,2.5e-4]. The sample generated with each of the model is shown in Figure 3. As it is evident that with higher KLD weight(β) value, high regularization is seen on the decoder output failing to reconstruct or generate the data, As KLD weight is decreased, we obtained better reconstructed output.

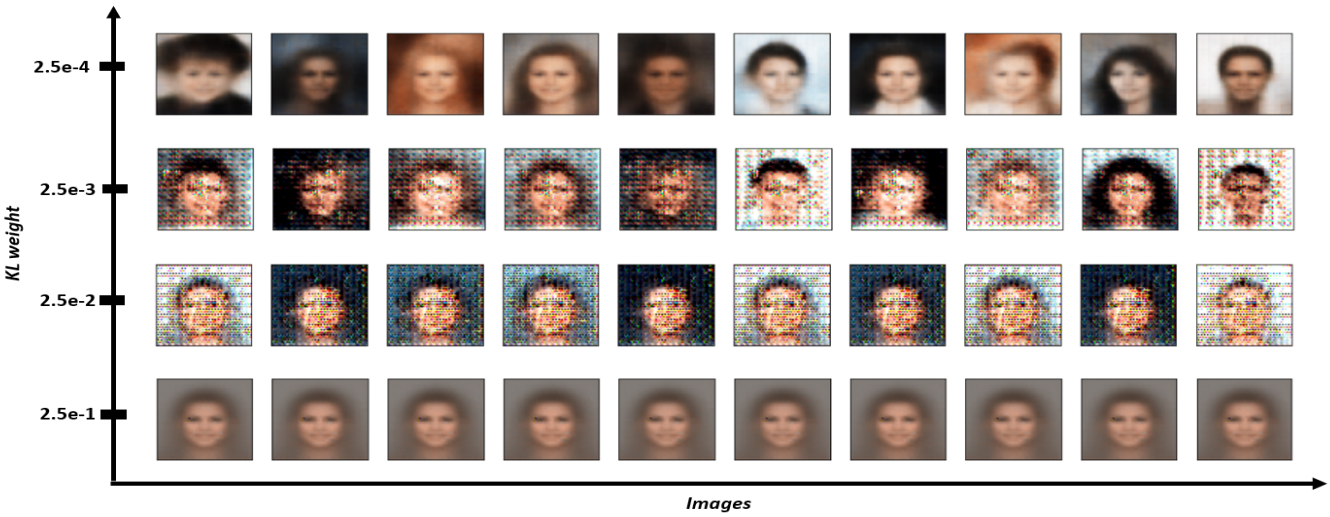

Figure 3: Sample generated with varying KLD weight(β)

Similarly, We trained models with varying latent dimensions of the network as [4,16,128,512] with KLD weight as 2.5e-4. The sample generated with each of the model is shown in Figure 4. A latent capacity of 4,16 are found to be not enough to reconstruct the output, with latent 128,512, we obtain better reconstructed output.

On dSprites dataset, we found latent dimensions of 6 enough to generate the data, with latent dimensions of 32, we obtained similar reconstruction results. Also, we obtain blurry reconstructed output when KLD weight(β) = 2.5e-2 as compared to 2.5e-3 as shown in Figure 5

After various hyper parameter optimization, we found that batch normalization helps is reconstructing better output with celebA dataset, we also found that at least 128 (512 is used) dimensional latent capacity for celebA dataset is required with found optimal KLD weight(β) = 2.5e-4. For dSprites dataset, since it is binary data, a small 2 layer MLP Encoder-decoder architecture is sufficient to reconstruct the output. We found latent dims of 6 with KLD weight (β) = 2.5e-3 sufficient to get decent output. We show the samples in 10x10 grids from the best fitted models in Figure 7. Some of the code was referenced from [2].

Figure 4: Sample generated with varying latent dimensions for celebA Dataset

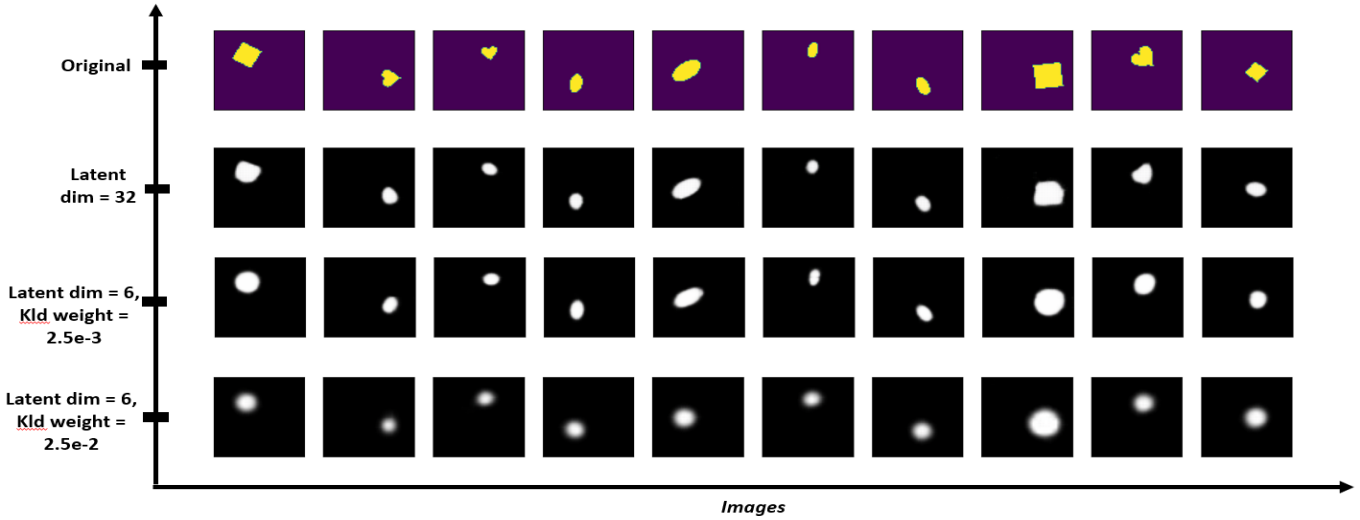

Figure 5: Sample generated with varying latent dimensions and KLD weight for dSprites Dataset

Figure 6: Samples from best fitted model for celebA and dSprites Dataset

2.4 Marginal Likelihood Estimation

Marginal Likelihood is estimated using the steps described in the Appendix D section of [4] with modifications as described below:

Step 1: To use gradient based MCMC method, it is required to get the gradient of the decoder p(x|z) prior p(z) with respect to z. Since decoder is a neural network, we required autograd to calculate the gradient of p(x|z) with respect to z.

$$\bigtriangledown_z logp_\Theta(z|x) = \bigtriangledown_z logp_\Theta(z) + \bigtriangledown_z logp_\Theta(x|z) \tag{3}$$

logp(z|x) is given as

$$logp_\Theta(z|x) = -\frac{z_2}{2} - \frac{(x - x`)^2}{2} + const \qquad wherex = reconstructed, x` = originalImage \tag{4}$$

This function is given as an input to the hamiltonian MCMC library[1] with an initial z value to perform autograd with respect to z and perform leaf frog iterations to obtain L samples. Initial z is obtained by taking sample x_i from dataset and passing it through the encoder p(z|x) to μ and σ and reparametrize to obtain initial z.

Step 2: Fit density estimator like GMM to fit on the L samples. Since I had a very big latent dimension of 512 and difficulty in fitting the model. Here approximation is performed. I have assumed multivariate (512) gaussian with known μ and σ from initial z as parameters of my multivariate gaussian.

Step 3: logq(z) is calculating the norm.pdf with the known μ and σ from initial z. logp(z) is calculated using norm.pdf with 0 mean and unit diagonal variance and log p(x|z) is calculated as

$$logp(x|z) = -\frac{Dln(2\Pi)}{2} - \frac{(x - x`)^2}{2} \tag{5}$$

$$temp = logq(z) - logp(x|z) - logp(z) \tag{6}$$

Estimate the temp for each of the L samples from single data point x_i and sum it over. Take logsumexp on top of it and subtract with log(L) to obtain -logp(x). Do it for all the data point and compute the average value.Below table shows the value obtain from the above calculations for both the dataset and different configurations

	Config	logp(x)	Bits/dim
CelebA	Latent = 512	-11821	1.388
	Latent = 128.	-12038	1.413
dSprites (Latent=6)	Beta = 2.5e-3	-11572	1.358
	Beta = 2.5e-2	-11619	1.364

Figure 7: Table showing marginal likelihood value and bits/dim for two dataset with different config

3 Deep Convolutional Generative Adversarial Network

3.1 Introduction

The DCGAN implementation is based on the paper *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks* by Radford et al[5]. The paper introduces a class of CNNs called Deep Convolutional Generative Adversarial Networks (DCGANs) for unsupervised training on various image datasets to learn a hierarchy of representations of object parts to scenes in a generator-discriminator pair.

DCGAN generator takes a vector as input and learns to predict an image. It avoids the problem of mode collapse where the generator gets biased only towards some classes and cannot produce outputs of every class in the dataset.

3.2 Model Structure

The model structure for Generator and Discriminator are in 8 and 9

3.2.1 Generator

The generator takes a 128 bit random vector samples where each bit $\sim \mathcal{U}[0, 1]$.
This vector is then fed into a series of *Conv2DTranspose* layers to form an image of size $64x64x3$.
Each layer is followed by Normalization layer followed by LeakyReLU except the final layer. Only the final layer has sigmoid activation.

3.2.2 Discriminator

The discriminator takes a $64x64x3$ image as input.
It has 3 *Conv2D* layers each of which is followed by Normalization layer and LeakyReLU. The final layer is a Dense layer with a sigmoid activation.

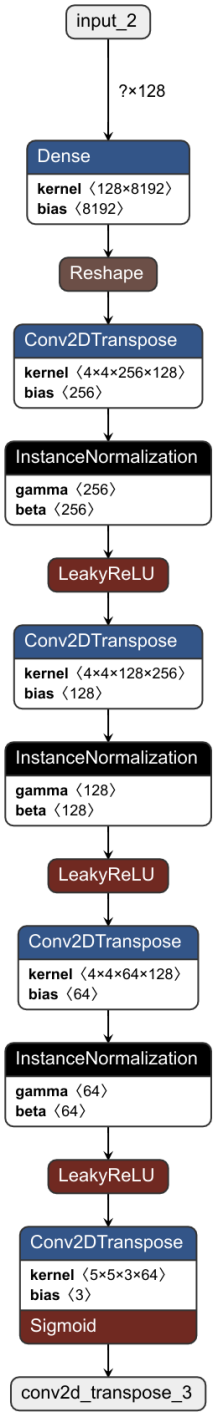

Figure 8: Generator Model

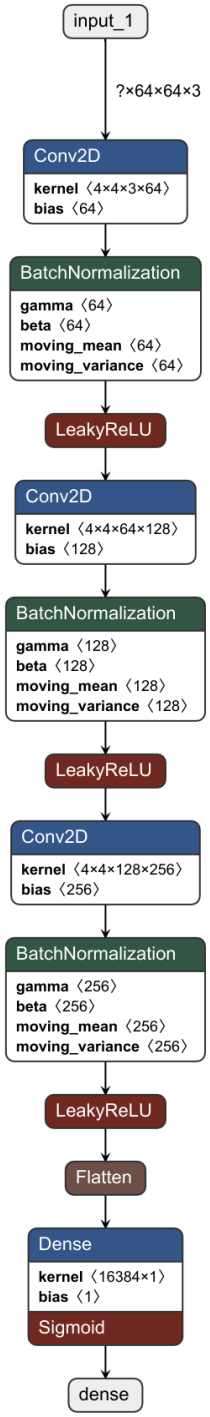

Figure 9: Discriminator Model

3.3 Training

3.4 Variations tried

1. We tried normalization on Discriminator alone, Generator alone, both Discriminator and Generator, and no normalization layers.
    - Having Normalization layers on only Discriminator or Generator improved the performance(loss numbers) of that component while the other component's performance degraded.
    - Having no normalization layers showed a similar performance as having normalization on both Discriminator and Generator.
    - Both Batch Normalization and Instance Normalization showed similar performance. The best results were observed with Batch Normalization on Discriminator and Instance Normalization on Generator.
2. Initialization of Discriminator and Generator with HeNormal and RandomNormal showed similar performance.
3. The images generated from the generator are good after 5 epochs. We ran the training for up to 25 epochs. Generator performance improves with the number of epochs with the improvement slowing after 5 epochs.
4. If the training is stopped after a number of epochs, models saved, reloaded, and training resumed, the optimizer for discriminator is not loaded properly, hence discriminator stops working, and training cannot be continued.

3.5 Results

3.5.1 Generator vs. Discriminator Loss

Figure 10 is a plot of Generator vs. Discriminator Loss.

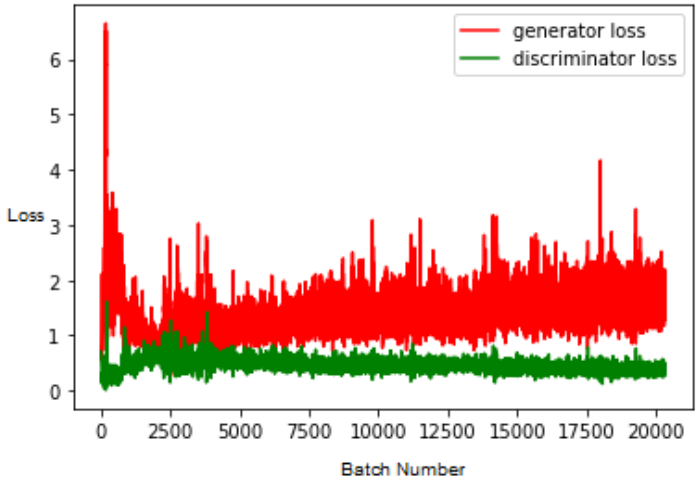

Figure 10: Generator vs Discriminator Loss

Observations:

- Generator shows a higher loss than discriminator.
- Generator loss is not stable and monotonic. It varies widely from batch to batch.
- Loss cannot be used to evaluate Generator performance because even with high loss, the generated images show good fidelity.

3.5.2 Example Generator Output

Figure 11 is an example 10x10 output from the Generator:

Figure 11: A 10x10 grid of bitmojis generated by generator

3.5.3 Frechet Inception Distance

Frechet Inception Distance (FID) was first described by Heusel et al [3] in the paper titled *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium.* It is a metric used to assess the quality of images generated by a generative model against real world dataset.
If there are two multivariate Gaussian distributions, $\mathcal{N}(\mu, \Sigma)$ and $\mathcal{N}(\mu', \Sigma')$, then the FID between them can be calculated by (Image from Wikipedia)

$$d_F(\mathcal{N}(\mu, \Sigma), \mathcal{N}(\mu', \Sigma'))^2 = \|\mu - \mu'\|_2^2 + \mathrm{tr}\left(\Sigma + \Sigma' - 2\left(\Sigma^{\frac{1}{2}} \cdot \Sigma' \cdot \Sigma^{\frac{1}{2}}\right)^{\frac{1}{2}}\right)$$

We can use Inception V3 model trained on Imagenet without it's final classification layer to obtain a 2048 dimensional vector for each image.
We use real images to obtain the first multivariate Gaussian distribution $\mathcal{N}(\mu, \Sigma)$, and generated images to obtain $\mathcal{N}(\mu', \Sigma')$.
The FID obtained for our GAN model was 155.14.

3.5.4 Other Notes

- The model structure was plotted using Netron

References

[1] AdamCobb. mcmc. `https://github.com/AdamCobb/hamiltorch`, 2022. [Online; accessed 09-Oct-2022].

[2] AntixK. Github. `https://github.com/AntixK/PyTorch-VAE/blob/master/models/vanilla_vae.py`, 2022. [Online; accessed 09-Oct-2022].

[3] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[5] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.