CMPE 597 HW2

İzzet Emre Küçükkaya
Boğaziçi University
Istanbul, Turkey
izzet.kucukkaya@boun.edu.tr, 2017401123

I. Introduction

In this homework, it is asked to implement VAE, GAN and WGAN using MNIST data. In VAE, LSTM layer is used in encoder, transpose convolution used in decoder. In GAN and WGAN, fully connected linear layers are used.

II. Variational Auto Encoder (VAE)

A. Encoder

As encoder, an LSTM layer is used with 128 hidden dimension and generated 4 means and standard deviations to generate the random vector.

B. Decoder

As encoder, 3 transpose convolutions are used. First of them has 32 channels and 8 kernel size, the second of them has 16 channels and 4 kernel size and the last of them has 8 channels and 2 kernel size. Then a fully connected layer is used to generate the image. The loss plot can be seen in Fig. 1 and KLD plot can be seen in Fig. 2

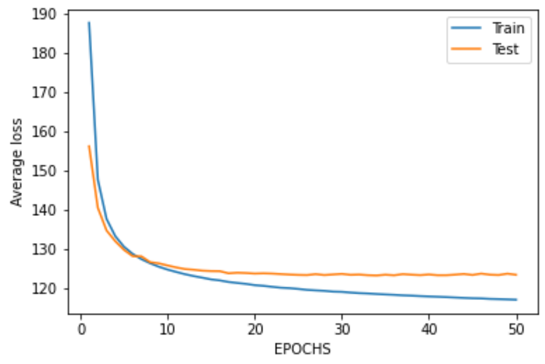

Fig. 1. Loss Plot of VAE

C. Loss

Binary cross entropy loss function is used and regularization term with KL-divergence. As expected, the loss is decreasing and the KL-Divergence is increasing which means that it can encode and decode image better and the probability distribution of images differs from the normal distribution as training proceeds. Furthermore, The Adam optimizer is used. The Adam optimizer is used.

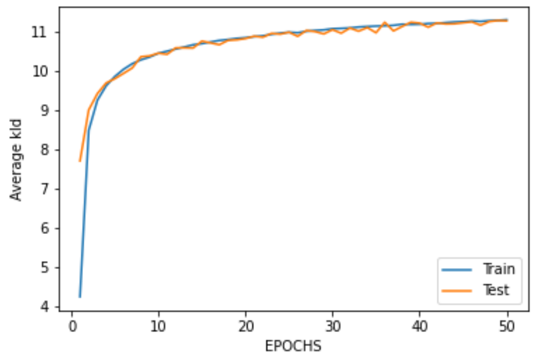

Fig. 2. KLD Plot of VAE

D. Result

The generated images can be seen in Fig. 6. As we can see the digits are clear to see but a bit blurry.

III. GAN with BCE loss

A. Generator

In generator, 256,512,1024,28*28 dimensional fully connected layers are used. Leaky RelUs are preferred for the hidden layers and the tanh is preferred for the last layer.

B. Discriminator

In discriminator, 1024, 512, 256, 1 dimensional fully connected laters are used. After each hidden layers leaky RelUs and dropouts are used. Sigmoid function is preferred for the last layer.

C. Loss

Binary cross entropy loss function is used. The Generator is trained to make the discriminator cannot decide if the generated image is real or not. Then the discriminator is trained to differ the fake images and the real ones. The Loss plot can be seen in Fig. 3. As we can see, as training proceeds, the losses are converges at a level, and the loss it becomes more stable. Furthermore, Adam Optimizer is used.

D. Result

The generated images can be seen in Fig. 7. As we can see the digits are clear to see but a bit noisy and most of the images are the digit 0.

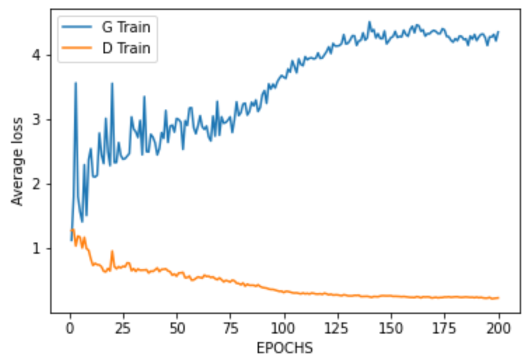

Fig. 3. LOSS Plot of GAN

IV. GAN WITH WASSERSTEIN LOSS

A. Generator

The generator is exactly same with III-A.

B. Discriminator

In discriminator, the activation function of the output layer is changed to linear. The rest of it is same with III-B

C. Loss

The loss function is changed to Wasserstein Loss based on the paper [1] and the implementation [2]. The methodology is still same with the III-C. The Loss plot can be seen in Fig. 4. As we can see, as training proceeds, the losses are converges at a level, and the loss it becomes more stable. Furthermore, RMSprop optimizer is used.

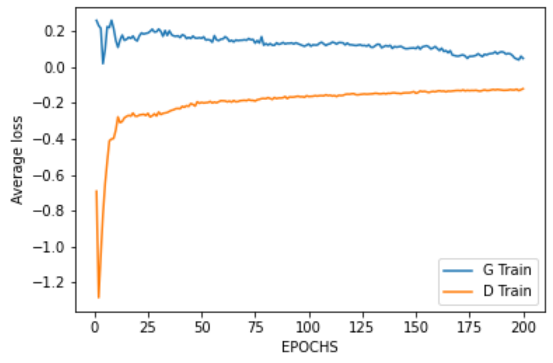

Fig. 4. LOSS Plot of WGAN

D. Result

The generated images can be seen in Fig. 8. As we can see the digits are clear to see but a bit blurry and harder to differ sometimes.

V. COMPARISON

As we can see, the generated images are satisfying. But in my opinion the VAE looks much differable and satisfying. Furthermore, GAN with BCE loss generates the smoother images but some images are hard to be detected. In the WGAN, The resultant digits are more varying but some digits are not able to good enough. For quantitative approach, Inception Scores [3] are calculated. I used the implementation in [4]. The results can be seen in Fig. 5. As we can see, VAE is the closest one to the 1.0 and the standard deviation of scores is smallest which means all of the images are well and the success of the model does not vary much. But WGAN has the largest average inception score and the standard deviation which means it gets worse solutions than the others averagely and the results are differ more than the others.

VI. GITHUB

This project is available on my github: [5].

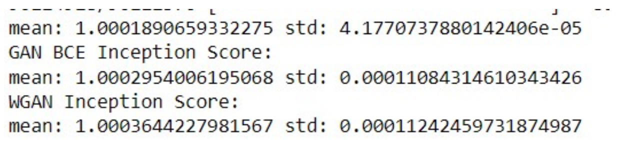
```
mean: 1.0001890659332275 std: 4.1770737880142406e-05
GAN BCE Inception Score:
mean: 1.0002954006195068 std: 0.00011084314610343426
WGAN Inception Score:
mean: 1.0003644227981567 std: 0.00011242459731874987
```

Fig. 5. Inception Scores

REFERENCES

[1] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein gan,” 2017. [Online]. Available: https://arxiv.org/abs/1701.07875

[2] A. Kristiadi, “Wasserstein GAN implementation in TensorFlow and Pytorch.” [Online]. Available: https://agustinus.kristia.de/techblog/2017/02/04/wasserstein-gan/

[3] A. Borji, “Pros and cons of GAN evaluation measures,” *CoRR*, vol. abs/1802.03446, 2018. [Online]. Available: http://arxiv.org/abs/1802.03446

[4] J. Brownlee, “How to Implement the Inception Score (IS) for Evaluating GANs.” [Online]. Available: https://machinelearningmastery.com/how-to-implement-the-inception-score-from-scratch-for-evaluating-generated-images/

[5] E. Küçükkaya, “CMP597 HW2 Github.” [Online]. Available: https://github.com/kardandon/CMPE597/tree/main/HW2/HW2/

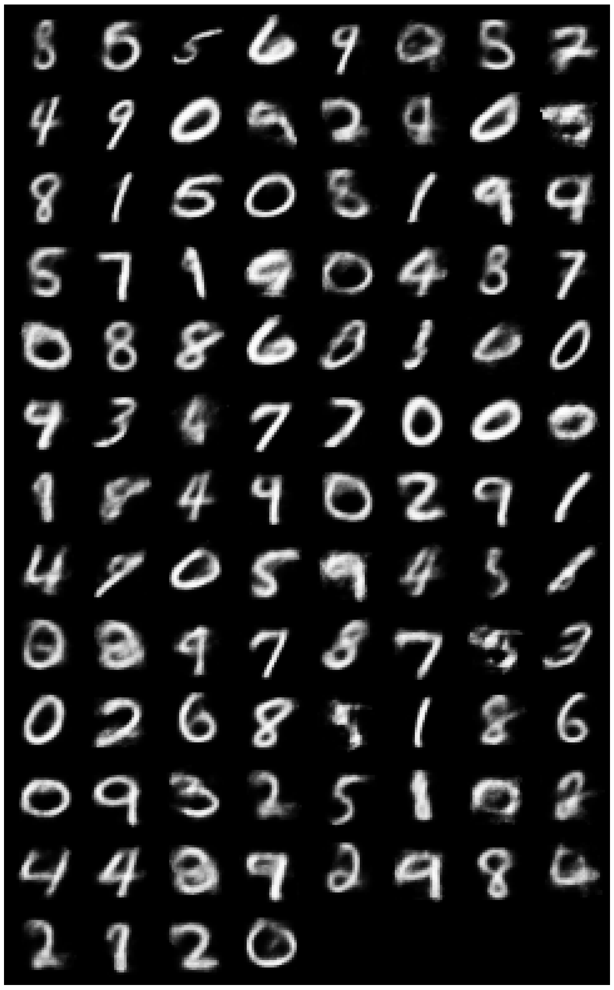
Fig. 6. Generated digits of VAE

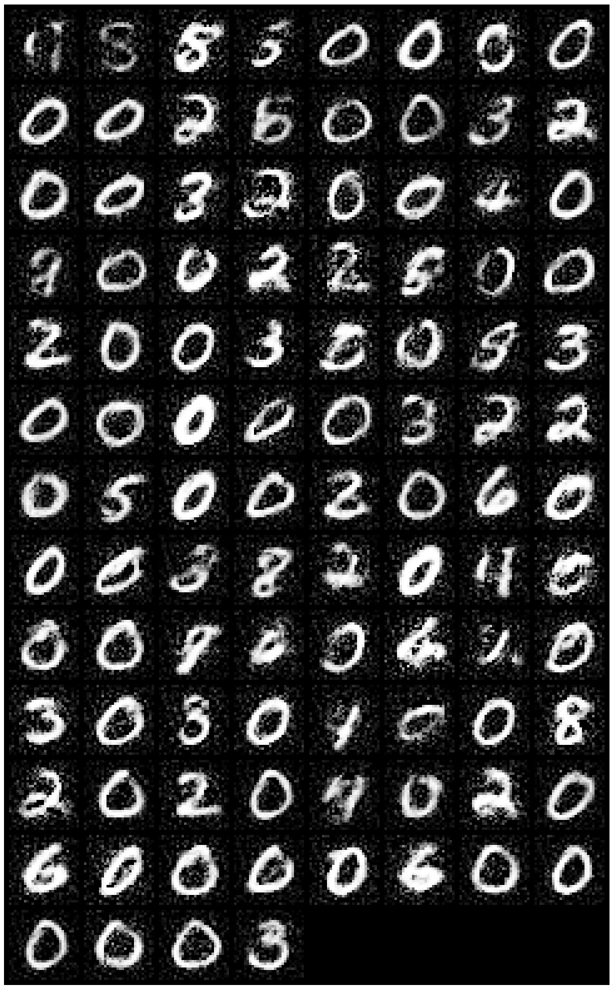
Fig. 7. Generated digits of GAN

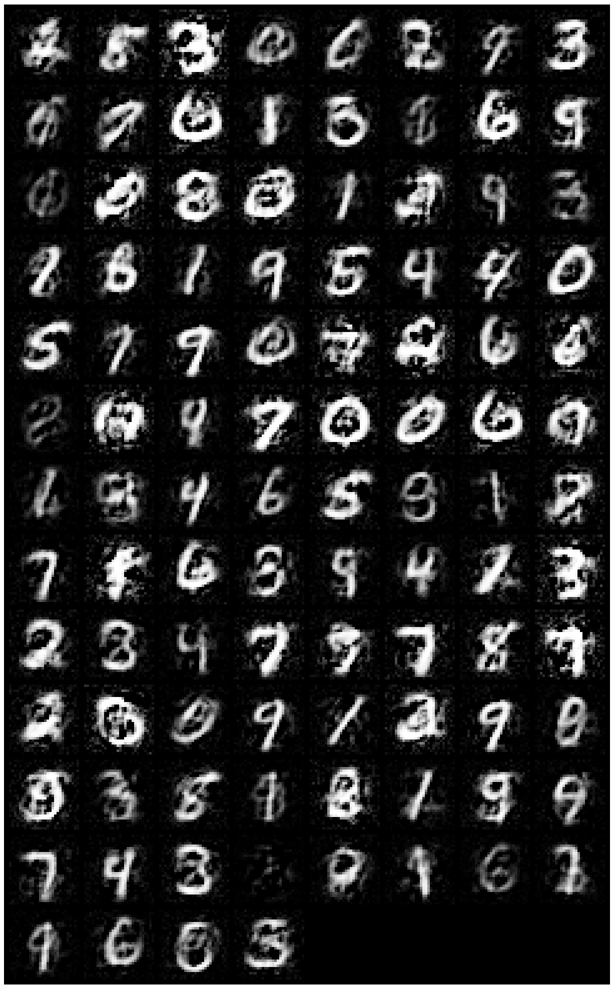

Fig. 8. Generated digits of WGAN