# Coursework 2

20170781 Seoyun Yang, 20170790 Deokmin Hwang

## I. DCGAN

In this document, we tried to find the best conditions to generate fake hand-written digit images using DCGAN. The basic architecture of the generator and discriminator is similar as given in the lecture slide. The generator uses 3 blocks of transpose convolution, batch normalization, ReLU and transpose convolution, tanh at the end. The discriminator uses 3 blocks of convolution, batch normalization, Leaky ReLU and convolution, sigmoid at the end.

### *A. Experimental Goal*

While training GAN, the discriminator usually wins early against the generator. It makes the gradient zero, then no more training is proceeding. In that result, the generator fails to make plausible images. We used several techniques to prevent this situation. First, we tried two kinds of objective functions for generator which are as follows

$$\mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))], \quad -\mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(D(G(\boldsymbol{z})))]$$

. In the case of the first objective function, when the discriminator is overwhelming the generator, the gradient becomes saturated. So we call this case a saturated case. While the gradient is not saturated when discriminator wins for the second one. We call it an unsaturated case. Second, we tried to find the proper ratio between depth of generator and depth of discriminator. Third, we tried the one sided label smoothing. Lastly, we tried to balance G and D by increasing the training epochs of D.

### *B. Experimental Result*

1) Adjusting the depth of G and D

Before entering, we adjust the depth of the network by increasing or decreasing the number of channels of each layer. First, we set the depth of the discriminator to be half to the generator. In Fig 1, images generated by G look not good. For the saturating function, images were different from each other, but in the case of non-saturating function the mode collapse happens. Next, when we set depths equal, we can see that mode collapse doesn't occur in both objective functions and we can see that the images were better generated when we use non-saturating one. But, they still look far from the digits. From now on, we proceed training using only the non-saturating function. When the depth of the discriminator was twice deeper than the generator, the generated images were clearer than before and seemed like digits. But still, the mean of the discriminator's prediction to the fake images was under 0.001.

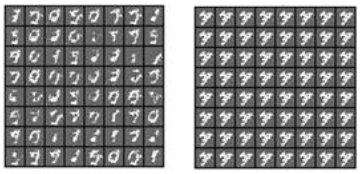


Fig 1. The fake images when the depth of the generator was twice deeper than the discriminator with saturating objective(left) and non saturating objective(right).

Fig 2. The fake images when the depth of the generator was same as the discriminator using saturating objective(left) and non saturating objective(middle). The fake images when the depth of the discriminator was twice deeper than the generator(right) .

2) One sided label smoothing

In this part, we set the true label as 0.9 to prevent gradient becoming zero. We found that when the depth of the discriminator is deeper than the generator works well before, so we tried the case of which the generator is deeper than and same with the depth of the discriminator. Both cases the mode collapse happens.

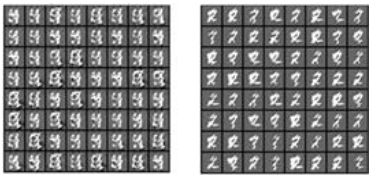


Fig 3.The fake images after adapting One-sided label smoothing. (left)The depth of generator is shallower than the discriminator, (right )the same depth.

3) Balancing G and D

In this part, we iterated training of the discriminator 3 times and 5 times to prevent the mode collapse. As we can see in Fig 4, there was no mode collapse, but when we iterated 3 times, it didn't look like digits. But, in the case of 5 times, we can get pretty clear images.

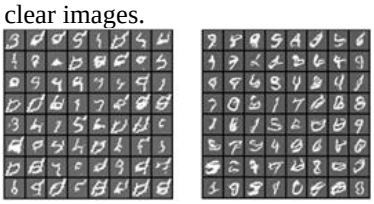


Fig 4. The fake images when increasing the training epochs of discriminator to 3(left) and 5(right).

### C. Discussion

One sided label smoothing was not sufficient to prevent mode collapse alone. And making the depth of discriminator deeper and iterating discriminator training more have effective results on generating digit images. Finally, we tried to generate images with all 3 techniques. In that result, we get the most looks-good digit images. But as we can see Fig 5, the discriminator was completely overwhelming the generator. Unlike the desired network which outputs 0.5 to the input images, our network outputs 1 for most of real data and 0 for most of fake data. So another way should be required to generate realistic generated images.

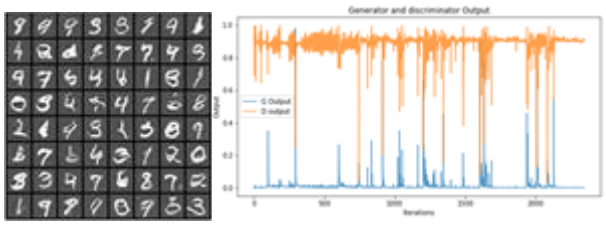



Fig 5. The fake images when we balanced depths and the number of training iteration and adopted label smoothing(left). The prediction of Discriminator to the real and fake images(right).

## II. CGAN

CGAN (Conditional GAN) is an extension of GAN where both the generator and the discriminator are conditioned on additional data. For this experiment, the label of the MNIST data is given as the additional data. So, when the generator is given a label from 0 to 9 as extra data besides the random noise, it generates an image that corresponds to that label.

### *A. Architecture and G and D Balancing*

The basic architecture of the generator and discriminator is built as given in the lecture slide. The generator uses three blocks of transpose convolution, batch normalization, ReLU and transpose convolution, tanh at the end. The discriminator uses three blocks of convolution, batch normalization, Leaky ReLU and transpose convolution, sigmoid at the end.

One-sided label smoothing, the number of iterations of training the discriminator, *dstep*, when the generator is trained once and the model size of the discriminator and generator were controlled for G and D balancing on top of the basic architecture.

### *B. Experimental Goal*

We aim to compare the inception score and mode collapse according to different G and D balancing.

Inception score is measured using TARR (Target Attributes Recognition Rate), where we train a CNN multi-class classifier using MNIST data and pass the generated image from the CGAN as the input. Then, compare the predicted label outputted by the CNN classifier with the label used for generating the image, as the success rate. The recognition accuracy of the CNN classifier on 10,000 real MNIST test images is 0.943. The success rates of the various CGAN measured by given 10,000 random noise and labels will be compared against this.

Mode collapse is when the CGAN starts generating one or a small output of images for each class label. In case of GAN, mode collapse happens regardless of the class labels. However, because CGAN gets a class label as an input, the mode collapse only happens inside a label. This would be shown in the experiment results.

### *C. Result and Discussion*

The first grid in Fig. 2.1 is the numbers generated on the basic architecture with *dstep* set to one and no one-label smoothing adjusted. The numbers are distinguishable between

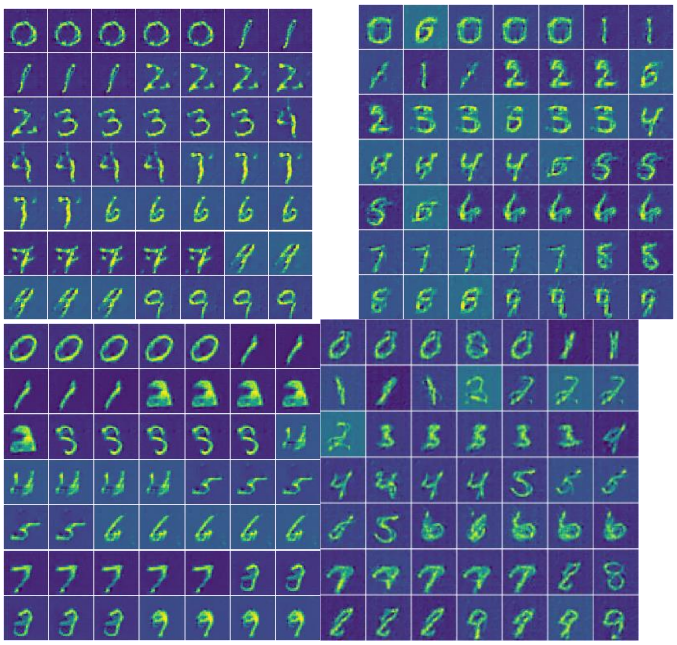


Fig. 2.1. The grid on the top left is the basic architecture with no other conditions. On the top right is the architecture with the deep discriminator only. On the bottom left is the architecture with the shallow generator and the deep architecture. On the bottom right is the architecture with the deep discriminator only with one-sided label smoothing.

classes, but the images within the classes are identical. This state is mode collapse, which was explained earlier. Also, some numbers are not trained well and hard to recognize. The success rate of these generated images is 0.799.

The first change we give is by changing the model size of the both generator and discriminator. We change the generator shallow into 2 blocks, and the discriminator deeper into 4 blocks in the architecture. The second grid in Fig. 2.1 is the numbers generated on this modified architecture. The generated numbers however are blurry even than the basic architecture. This is because if the generator is too shallow, the numbers wouldn't be able to be well generated. Also, mode collapse has still happened, as we can see the same types within classes, but the subset of numbers appearing is larger than the basic architecture, which only outputs one type for each label. The success rate went up to 0.864.

Another architecture we try is by keeping the generator to 3 blocks but by making the discriminator have 4 blocks. The third grid in Fig. 2.1, which are the generated numbers however, doesn't show any progress from the basic architecture. The success rate is also similar to 0.802. These results show that when the discriminator is quite deeper than the generator, CGAN generates better numbers.

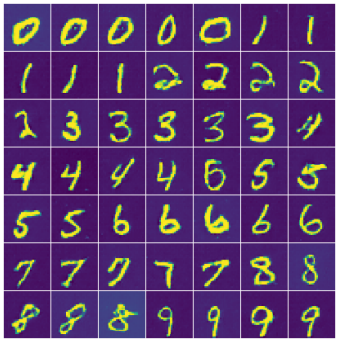


Fig. 2.2. The numbers generated on the basic architecture setting *dstep* to 5.

The second change is by adjusting one-sided label smoothing. Usually, the target label of real images is 1 and fake images is 0. For one-sided label smoothing, we set the target label of real images to 0.9. By smoothing the label like this, it helps reduce the risk of adversarial examples by preventing the discriminator from giving the generator very large gradient signals. The fourth grid in Fig. 2 shows the generated images using the modified architecture with only the deep discriminator and one-sided label smoothing. There is a small progress as we can check from the success rate that increased to 0.881.

Lastly, we increase *dstep* to 5. By training the discriminator more, as making the architecture of the discriminator deeper, it can learn better how to distinguish between the real and generated images, therefore sending back more meaningful feedbacks. The fourth grid in Fig. 2.2 shows the generated images with *dstep* set to 5 on the basic architecture, with one-sided label smoothing not adjusted. We trained this CGAN with less epoch than before because it still leads to a satisfying result. The numbers generated have diversity between classes and within classes and are very clear. The success rate using these numbers is 0.948, which is compatible to the recognition accuracy of the real MNIST test data.

So, we can conclude that training the generator profoundly leads to a better result. Among all the methods we have tried, increasing *dstep* is the best solution, that can generate numbers that are practically the same as the real MNIST data.

## III. RETRAINING THE HANDWRITTEN DIGIT CLASSIFIER

In the previous content, we improved the accuracy of the pre-trained digit image classifier from 0.943 to 0.948. The accuracy of 0.948 is high accuracy, but it is still not enough to use in real life. So in this chapter, we tried to make a more accurate classifier using the generated digit images from GAN.

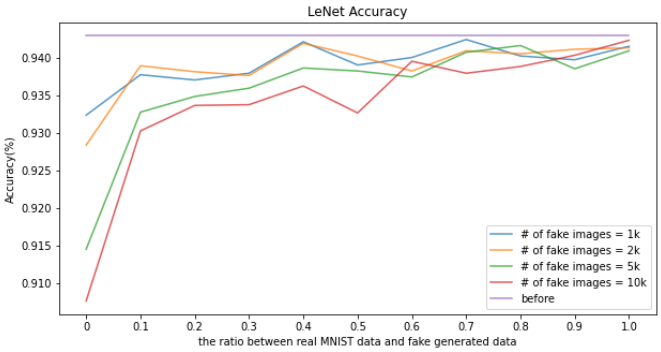
### A. Experimental Setting

We tried 4 experiments. First, we additionally trained the LeNet using the artificial digit image from GAN. In this experiment we tuned the number of images used for training and the ratio between the amount of real MNIST data and the amount of generated fake data. Next, we assumed that the badly-generated fake images will interrupt the training. So, we estimated the fake images using Inception Score and we used selected images for training of which IS was higher than the threshold. Like the first experiment, we tuned the amount of data and the ratio of data. Third, we initialized the LeNet parameter and trained it all over again. Lastly, under the situation that there is not enough data, we investigated the effects of fake data.

### B. Result

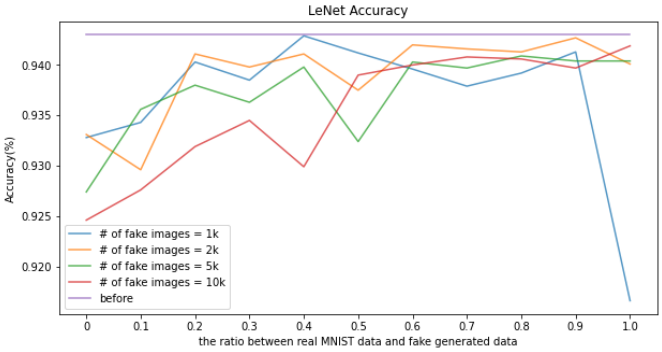
1) Additionally train the LeNet with fake images

We additionally retrained LeNet with different numbers of fake images generated from GAN and real data. And we also changed the number of real MNIST data used for additional training. There was no case for surpassing the original one.



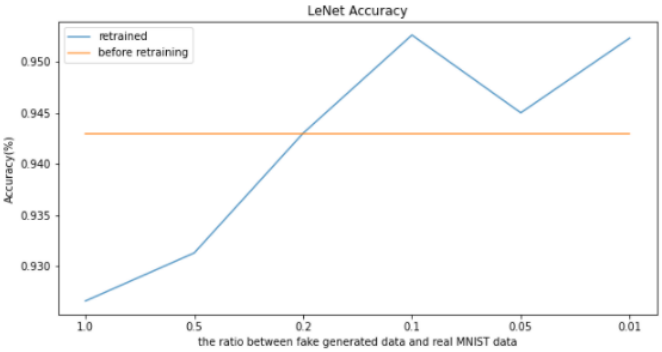

2) Additionally train the LeNet with selected fake images

We assumed that low-quality fake images interrupted the training. So, we only used fake data of high Inception Score for retraining. We determined it was good if the IS was over 0.5. Like the first experiment, we tried several different numbers of fake images and real images. Also, there was no case for surpassing the original one.




3) Train the LeNet from scratch

Next, we trained LeNet from scratch using the real data and fake data. In this case, when the ratio of fake images over real images was less than 0.1, it shows much higher accuracy. It records 0.9526 when the ratio was 0.1.




4) Training when the real data is not enough

Lastly, we preceded experiments under the situation where the real data is not enough. In this experiment, we trained the network from scratch using only 500, 1000, 10000 MNIST real data. And also we additionally retrained with increasing the number of fake data from 0 to twice of the number of real data.


| # of fake / # of real | 0 | 0.01 | 0.1 | 0.5 | 1 | 2 |
|---|---|---|---|---|---|---|
| 500 | 0.3468 | 0.3827 | 0.4494 | 0.4423 | 0.4346 | 0.5978 |
| 1000 | 0.6279 | 0.5608 | 0.608 | 0.5872 | 0.6833 | 0.656 |
| 10000 | 0.8695 | 0.8788 | 0.8706 | 0.8813 | 0.8901 | 0.8685 |

### C. Discussion

The model learned through real mnist data already shows an excellent accuracy of 0.948. To increase this, additional learning of fake data, which was not previously seen, was expected to increase the accuracy of test data. However, as shown in Experiments 1 and 2, simply re-training a well-learned model was not enough to increase accuracy. Positive results were obtained in Experiment 3. If fake data made through GAN is used to learn from the beginning, it can be confirmed that the accuracy increases by the original network. This shows the possibility of training a network through fake data in a limited situation. However, depending on the performance of the generator, a lot of noise and outlier data occur. When the fake data ratio was small in Experiment 3, the accuracy was improved, indicating that the proportion of outliers should be small. In particular, the possibility of GAN can be confirmed well in Experiment 4. In the situation where actual data is insufficient, the accuracy increased by 1.7 times depending on the amount of fake data. Of course, there is still a problem that making a better fake image requires a lot of actual data for training.

## IV. GAN AND PCA

PCA (Principal Component Analysis), a dimensionality reduction technique that generates images on a lower dimension using the principal components, is also another generative model besides GAN. By making a combination of the principal components, we can generate new images. We will look through the strengths and weaknesses of PCA and GAN.

### *A. PCA*

The strength of PCA is that it performs dimensionality reduction. First, this helps the model to train much faster. Also, by keeping only the important information and getting rid of the ones that aren't, we can get a better understanding of the data. As data on high dimensionality with unnecessary information included, degrades the analysis.

The weakness, however, also comes from dimensionality reduction. Even though the principal components are kept, there still exists information loss. Also, features presented by principal components are harder to interpret and recognize than the original features.

### *B. GAN*

The strength of GAN is that it can train well even without labels of the data. Also, among the generative models, it is most available for generating images that are close to real images, not blurry but clear. Also, it can train with multi-modal outputs. The weakness is that it takes a long time to train the model. Also, it suffers from mode collapse and non-convergence problems, meaning the models are unstable. Due to the fact that it has to train both the generator and the discriminator, finding the balance between the two is also hard work, consuming a lot of time.

### *C. PCAGAN*

PCAGAN[1] is a combination of PCA and GAN to gain the advantages of both and compensate for the shortcomings. By applying PCA first, we can obtain the important features while reducing the dimension of the original data. Then, by applying GAN, we can generate more accurate and clear images in a much shorter time.

## References

[1] C. Wang, P. Wu, J. Fu, Y. Zhou, J. Hu and L. Yan, "Image Classification Based on Principal Component Analysis optimized Generative Adversarial Networks," *2020 IEEE 5th International Symposium on Smart and Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS)*, 2020