Generative Adversarial Nets Report

Haoyuan Zhang (Robotics MSE, GRASP Lab, UPenn)

December 12, 2017

Abstract

This project aims at implementing, analyzing and tuning/improving **deep generative models**. Deep generative models have emerged to be one of the promising directions in both generative models and deep learning. Recent success in deep generative model has shown the potential to generate high resolution real-looking images. In this project, we implement and analyze one major branch of deep generative model, **Generative Adversarial Networks (GAN)**.

1 Generative Adversarial Networks

Generative adversarial networks (GANs) is a framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather that model G.

In this part, we first build a basic version of GANs that is suitable for modeling natural images, i.e., deep convolutional generative adversarial networks (DCGANs). After that, we implement an improved GAN to get a better performance, here we use the **Wasserstein GAN (WGAN)** and its improved version, **Wasserstein GAN with Gradient Penalty (WGAN-GP).**

We follow the below two architectures to build the generative model G and the discriminator model D.

Generative Model G

Layers	Hyper-parameters
Fully Connected Layer	hidden neuron number is $4 \times 4 \times 512$, followed by Batch normalization and ReLU.
Deconvolution 1	Kernel size = (5, 5, 256), stride = (2, 2), followed by Batch normalization and ReLU.
Deconvolution 2	Kernel size = (5, 5, 128), stride = (2, 2), followed by Batch normalization and ReLU.
Deconvolution 3	Kernel size = (5, 5, 64), stride = (2, 2), followed by Batch normalization and ReLU.
Deconvolution 4	Kernel size = (5, 5, 1), stride = (2, 2), followed by the activation function **Tanh**.

Discriminator Model D

Layers	Hyper-parameters
Convolution 1	Kernel size = (5, 5, 64), stride = (2, 2), followed by Leaky ReLU.
Convolution 2	Kernel size = (5, 5, 128), stride = (2, 2), followed by Batch normalization and Leaky ReLU.
Convolution 3	Kernel size = (5, 5, 256), stride = (2, 2), followed by Batch normalization and Leaky ReLU.
Convolution 4	Kernel size = (5, 5, 512), stride = (2, 2), followed by Batch normalization and Leaky ReLU.
Fully Connected Layer	hidden neuron number is 1.

In addition, we use the **Layer Normalization** to replace the original **Batch Normalization** in the discriminator model when implement the **WGAN-GP**.

Please feel free to check the ***Res*** folder to get more intuitive results of our performance.

1.1 DCGAN (CUFS dataset)

In this section, we build a simple DCGAN model to train the CUFS dataset. Here, we follow the strategies below:

- Set batch size equal to 64.
- Train model D once and G twice for one iteration.

- Set same learning rate for model D and G with value 0.0002.
- Set the max iteration equal to 5000.
- Generate a normal distribution as the sampled noise z (100-d).

Below shows the loss curve over training iterations.

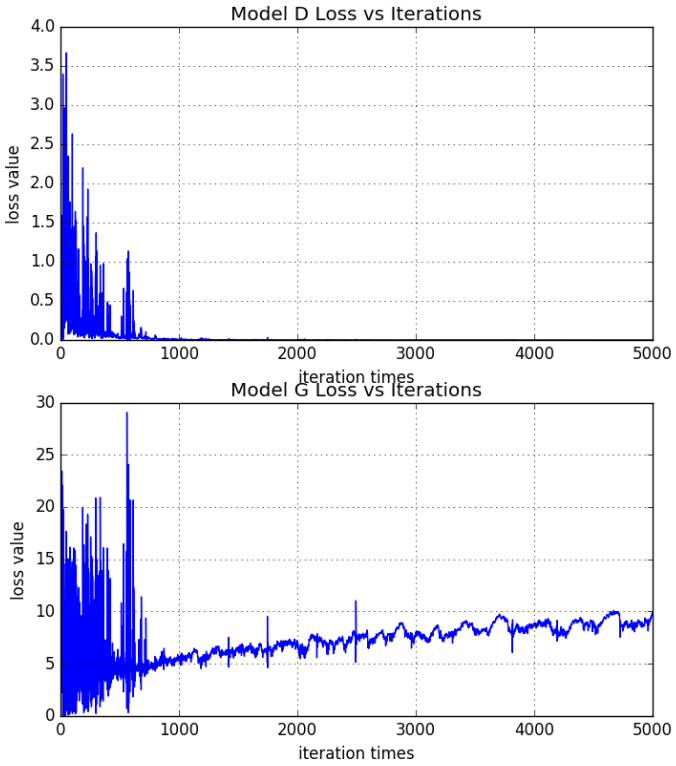
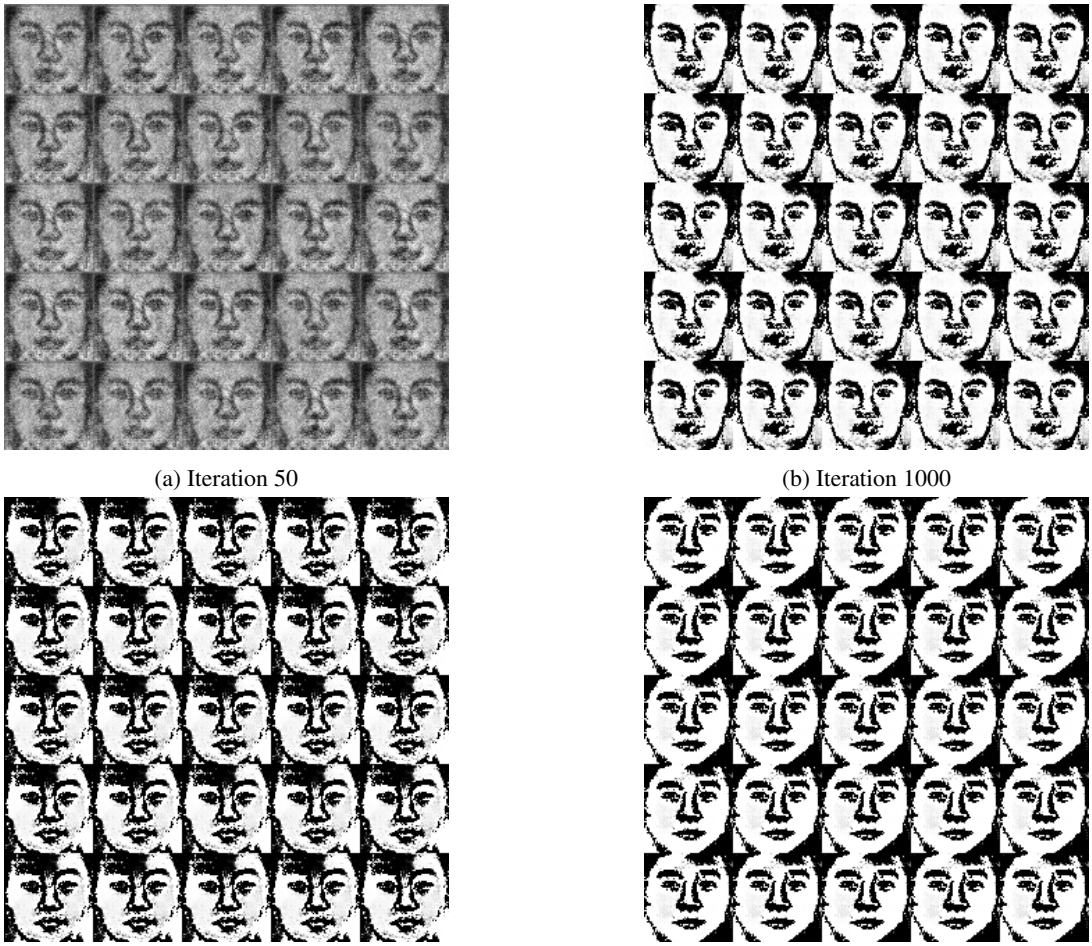


Figure 1: Training loss curve (DCGAN + CUFS)

Below show several generated sketches from random sampled z.



(a) Iteration 50 (b) Iteration 1000

(c) Iteration 2000 (d) Iteration 4500

Figure 2: Generated sketches from the random sampled z (DCGAN + CUFS)

Observation and Analysis: In this section, we use deep convolutional layers to build both generative model G and discriminator model G. Unlike VAE (Variational Auto-Encoders), GAN doesn't assume a single loss function, instead, the system makes the model G and D compete with each other such that both of them can become stronger over the training iterations.

More specifically, the generative model G aims to generate fake samples from random noise, then use those generated samples to fool the discriminator D:

$$L_G^{GAN} = E[\log(D(G(z)))]$$

For the discriminator model D, its goal is to distinguish the real and fake samples(obtained from model G), the loss is defined as below:

$$L_D^{GAN} = E[\log(D(x))] + E[\log(1 - D(G(z)))]$$

Generally, the discriminator model D plays a more powerful role than the generator G during the training, since the model G needs to learn some strategies trying to fool the discriminator, therefore, to some extents, the generator learns from the discriminator D.

Here, according to the result loss curve, there exists a lots of fluctuations, especially for the model G. That shows a typical competition game between two models. During the training process, both of them can learn better strategies to beat each other. Finally, two losses should decrease to some value interval (means they master better strategies) and fluctuate as well (means they are still competing with each other). The generated sketches show more representative results of DCGAN, with the iteration times accumulated, the samples become more and more similar to the real data.

However, there still exist some problems of DCGAN. The model instability still remains, we noticed as models are

trained longer they sometimes collapse a subset of filters to a single oscillating mode. Besides, DCGAN is harder for training than some standard GANs due to its deep convolutional layers, and sometimes, deeper and more complicated structure may not provide better performance, and even worse somehow. Those problems are highly likely caused by the instability of GAN architecture.

1.2 DCGAN (celebA dataset)

In this section, we build a simple DCGAN model to train the celebA dataset. Here, we follow the strategies below:

- Set batch size equal to 64.
- Train model D once and G twice for one iteration.
- Set same learning rate for model D and G with value 0.0002.
- Set the max iteration equal to 10000.
- Generate a normal distribution as the sampled noise z (100-d).

Below shows the loss curve over training iterations.

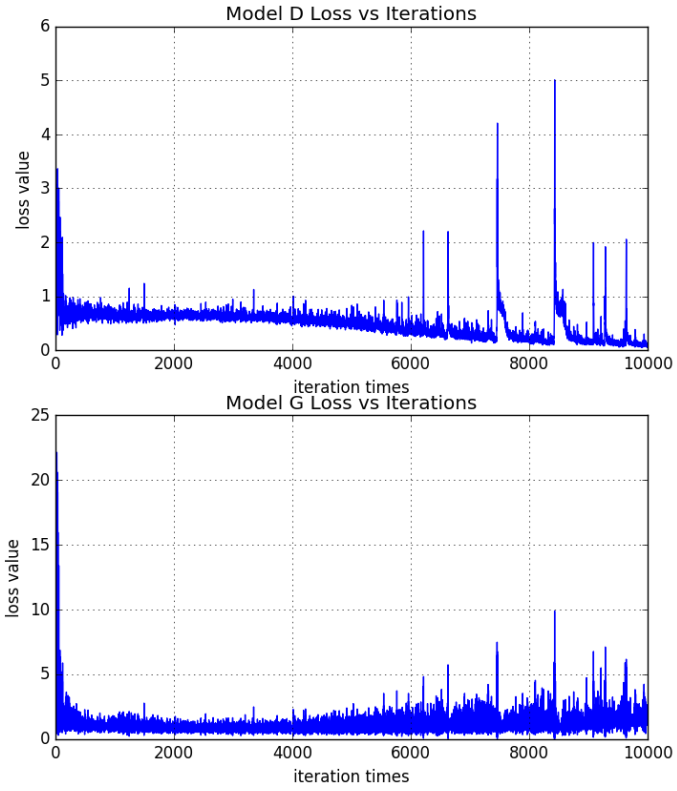
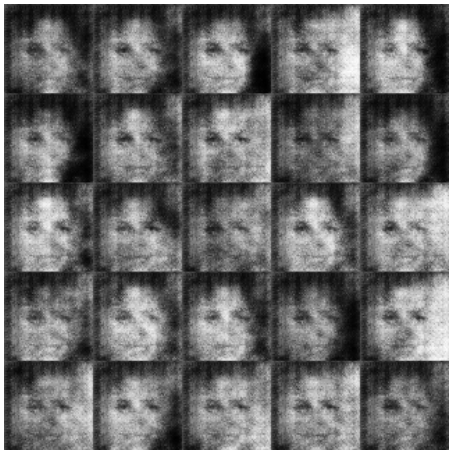
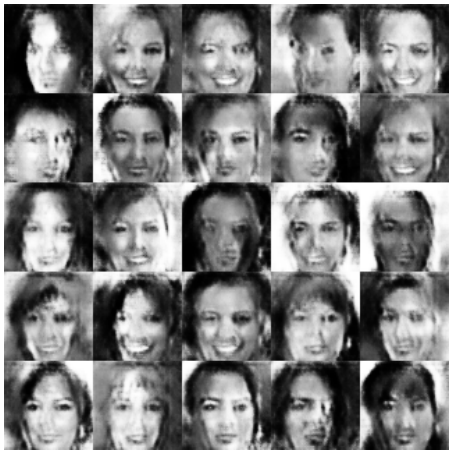


Figure 3: Training loss curve (DCGAN + celebA)

Below show several generated sketches from random sampled z.



(a) Iteration 100



(b) Iteration 1000

(c) Iteration 5500

(d) Iteration 10000

Figure 4: Generated sketches from the random sampled z (DCGAN + celebA)

Observation and Analysis: We follow the same DCGAN model as the previous one to train a different dataset, celebA. According to the loss curves, it becomes more reasonable than the CUFS one, both overall losses are decreasing with fluctuations locally. Finally, we can get better look generated sketches.

However, the instability problems still remain, over the iterations accumulation, those generated faces keep fluctuating a little bit, although the qualities make sense. Those problems are caused by the **loss function of the generator G**.

Recall the loss function of discriminator is

$$-E_{x\in P_r}[\log D(x)] - E_{x\in P_g}[\log(1-D(x))]$$

Here, P_r represents the real dataset while P_g represents the fake samples generated from the generator G. The main idea of discriminator loss function is trying to maximize the probability to detect real data while minimize the probability for fake samples. Then the author raises two types of loss functions of the generator G,

$$E_{x\in P_g}[\log(1-D(x))]$$

$$E_{x\in P_g}[\log(-D(x))]$$

We call the latter one as *"the -log D alternative"* or *"the - log D trick"*. However, both of them are not optimal and cause problems for the GAN training.

More specifically, the first loss function will lead to the **dying gradient** problem for the model G training due to the

Jensen-Shannon divergence influence. That is, after the model D training in each epoch which can get optimal discriminator, the gradient that leads to minimize the model G's loss becomes a constant ($\log 2$). Therefore, the reason that leads to the instability of GAN is clear:

- If train the discriminator too good, the gradient of model G will disappear.
- If the discriminator is not good, the gradient for the generator has large error.

It's pretty hard to handle how good the discriminator should be after each epoch training.

For the second loss function, it needs to deal with an opposite problem, minimize the KL divergence but maximize the JS divergence at the same time which is almost impossible. What's more, it introduces different penalties to the diversity and accuracy of the generated fake samples which leads to model jump into the **mode collapse** problem easily.

In order to deal with those drawbacks of original GAN, we can add some noises to two distributions, which can truly increase the performance and solve the instability problem. However, there still exists requirements to come up with the index to show the training process. Therefore, more improved and advanced GANs are necessary to be raised up.

1.3 Improved GAN: WGAN and WGAN-GP

In this section, we implement two improved version GANs, WGAN and WGAN-GP.

1.3.1 WGAN

GANs are powerful generative models, but still suffer from the training instability problem. The recently proposed Wasserstein GAN (WGAN) makes progress towards **stable** training of GANs.

The key idea of WGAN is introducing the **Wasserstein** (also called **Earth-Mover (EM)**) distance to replace original loss computation functions. The main advantage of W-distance compared with KL-divergence and JS-divergence is that, even two distributions have no overlap, it can still measure the similarity of them, such that it can provide the meaningful gradients for training. By introducing the **Lipschitz Constraints** and **Weight/parameters Clipping**, WGAN can optimize the model via minimizing the Wasserstein distance of the discriminator and generator distributions:

$$Loss_{discriminator} = E_{x \in P_g}[f_\omega(x)] - E_{x \in P_r}[f_\theta(x)]$$

$$Loss_{generator} = -E_{x \in P_g}[f_\omega(x)]$$

We follow the below strategies for the WGAN architecture implementation.

- Set batch size equal to 64.
- Set the number of critic **n_critic** equal to 5 and **weight clipping** value is 0.0015.
- Set same learning rate for model D and G with value 0.0015.
- Use **RMSProp or SGD** to replace the dynamic optimization algorithms (e.g. momentum and Adam).
- Set the max iteration equal to 5000 (CUFS) and 10000 (celebA).
- Generate a normal distribution as the sampled noise z (100-d).

Below show results of WGAN for both CUFS and celebA datasets respectively.

1. **WGAN on CUFS Dataset**

   Below shows the loss curve over training iterations.

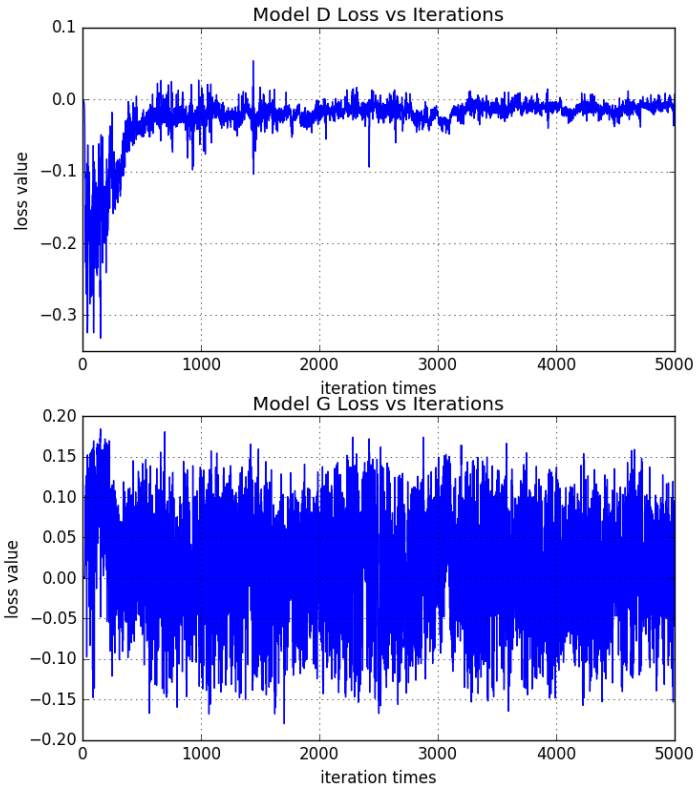
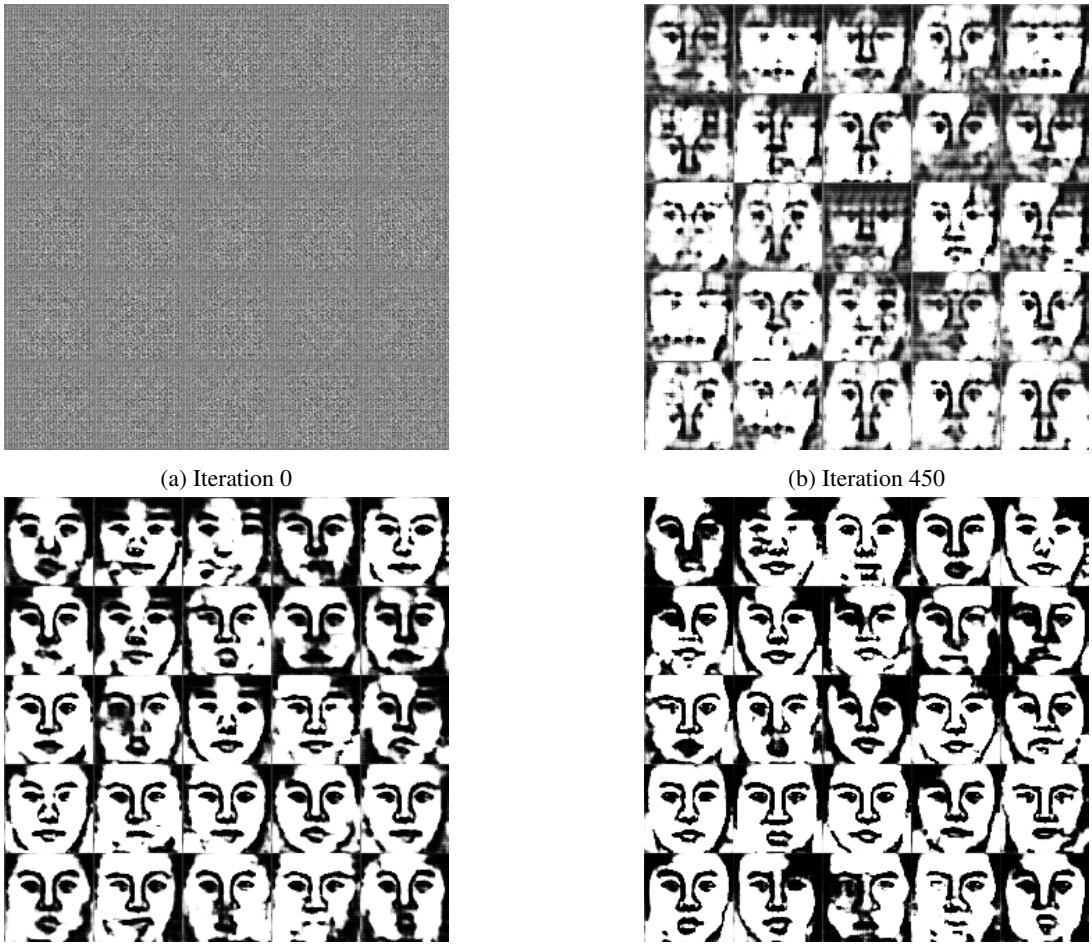


Figure 5: Training loss curve (WGAN + CUFS)

Below show several generated sketches from random sampled z.



(a) Iteration 0

(b) Iteration 450

(c) Iteration 2000

(d) Iteration 5000

Figure 6: Generated sketches from the random sampled z (WGAN + CUFS)

2. **WGAN on celebA Dataset**

   Below shows the loss curve over training iterations.

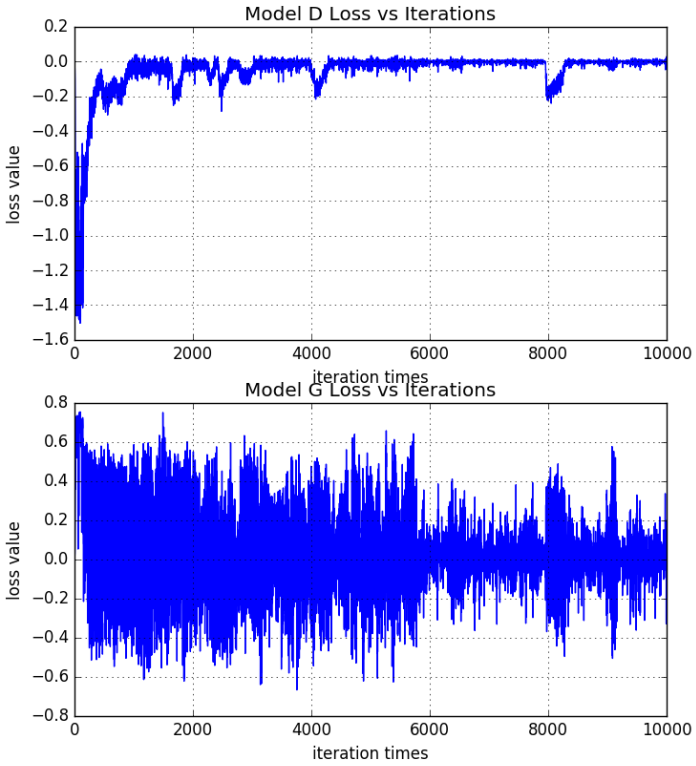
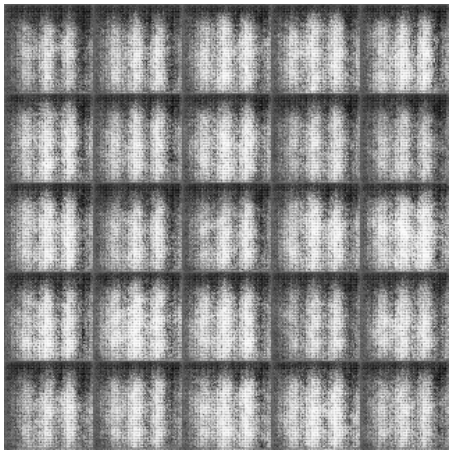


Figure 7: Training loss curve (WGAN + celebA)

Below show several generated sketches from random sampled z.



(a) Iteration 100

(b) Iteration 2000

(c) Iteration 7000

(d) Iteration 10000

Figure 8: Generated sketches from the random sampled z (WGAN + celebA)

Observations and Analysis: WGAN (Wassertein GAN) tells the truth that without special and complicated model construction and training strategies, it's also possible to train a good performance GAN. In WGAN, the task of model D is not trying to distinguish real and fake samples anymore, instead, it tries to fit the Wasserstein distance within the samples, that is, the classification problem is converted into a regression one. Meanwhile, the model G plays a role to reduce the Wasserstein distance.

More specifically, WGAN modifies on below several aspects:

- Remove the last **sigmoid** layer.
- Constraints the weights value of model D into the interval **[-c, c]**.
- Use SGD or RMSProp with some low learning rate to train the model.

However, according to our practical experiments and the obtained results(especially the celebA dataset), WGAN performs even worse. Actually, there still remain some instabilities problem in the WGAN. Those problems are often due to the use of weight clipping in WGAN to enforce a Lipschitz constraint on the critic, which can lead to pathological behavior.

More specifically, given the hard clipping weight value, the optimal strategy is letting all weights to two extreme boundaries value $-c$ and c which enforces the model to be a simple one, such that waste the powerful neural network nonlinear fitting ability, and then the back-propagated gradients also become worse. Therefore, that's why WGAN performs worse on the celebA dataset which contains more complicated sketches structures than CUFS.

Another main problem caused by the weight clipping strategy is that, it will highly likely cause the dying gradient (too small clipping) or gradient explosion problem (too large clipping). It's quite difficult to tune into a reasonable value.

1.3.2 WGAN-GP

Based on the WGAN, the improved version, WGAN-GP, is popular used nowadays. The new model proposes an alternative to clipping weights: penalize the norm of gradient of the critic with respect to its input. The updated WGAN performs better than standard WGAN and enables stable training of a wide variety of GAN architectures with almost no hyper-parameter tuning.

We follow the below strategies for the WGAN-GP architecture implementation.

- Set batch size equal to 64.
- Set the number of critic **n_critic** equal to 5.
- Set same learning rate for model D and G with value 0.0002.
- Set the max iteration equal to 5000 (CUFS) and 10000 (celebA).
- Generate a normal distribution as the sampled noise z (100-d).
- Use the **Layer Normalization** to replace the original **Batch Normalization** in the model D since WGAN-GP add the gradient penalty to each instance independently.

We train WGAN-GP for both CUFS and celebA datasets and compare the performance with DCGAN.

1. **WGAN-GP on CUFS Dataset**

   Below shows the loss curve over training iterations.

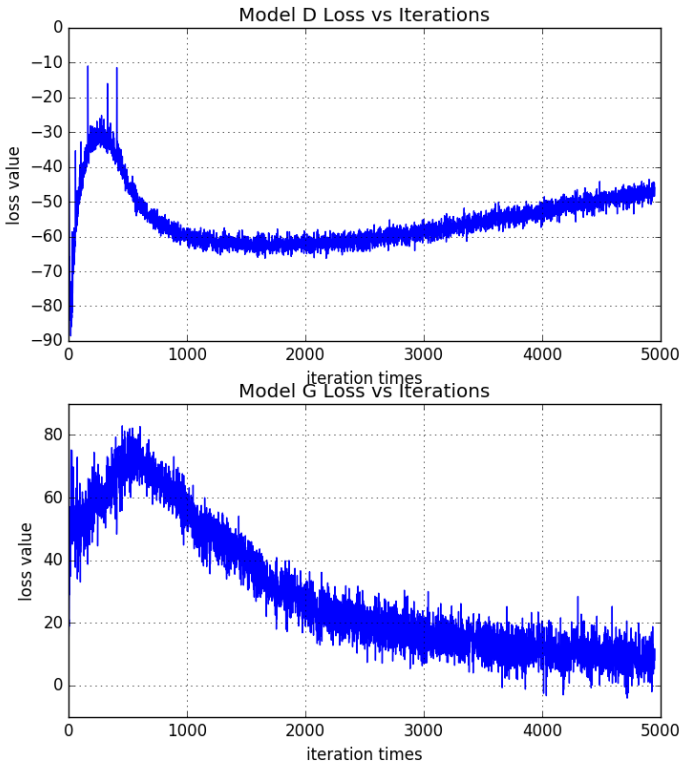


Figure 9: Training loss curve (WGAN-GP + CUFS)

Below show several generated sketches from random sampled z.

(a) Iteration 50

(b) Iteration 1000

(c) Iteration 3000

(d) Iteration 5000

Figure 10: Generated sketches from the random sampled z (WGAN-GP + CUFS)

2. **WGAN-GP on celebA Dataset**

Below shows the loss curve over training iterations.

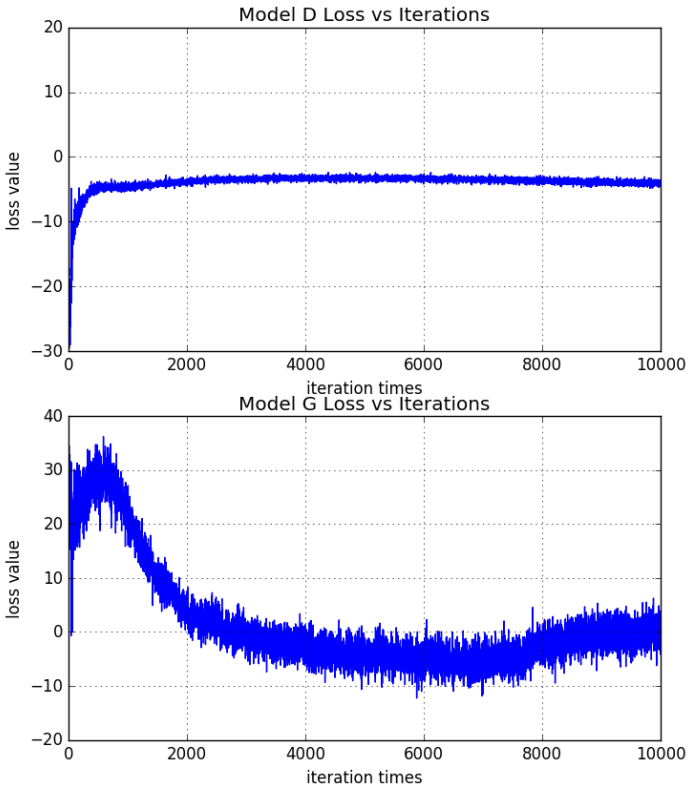
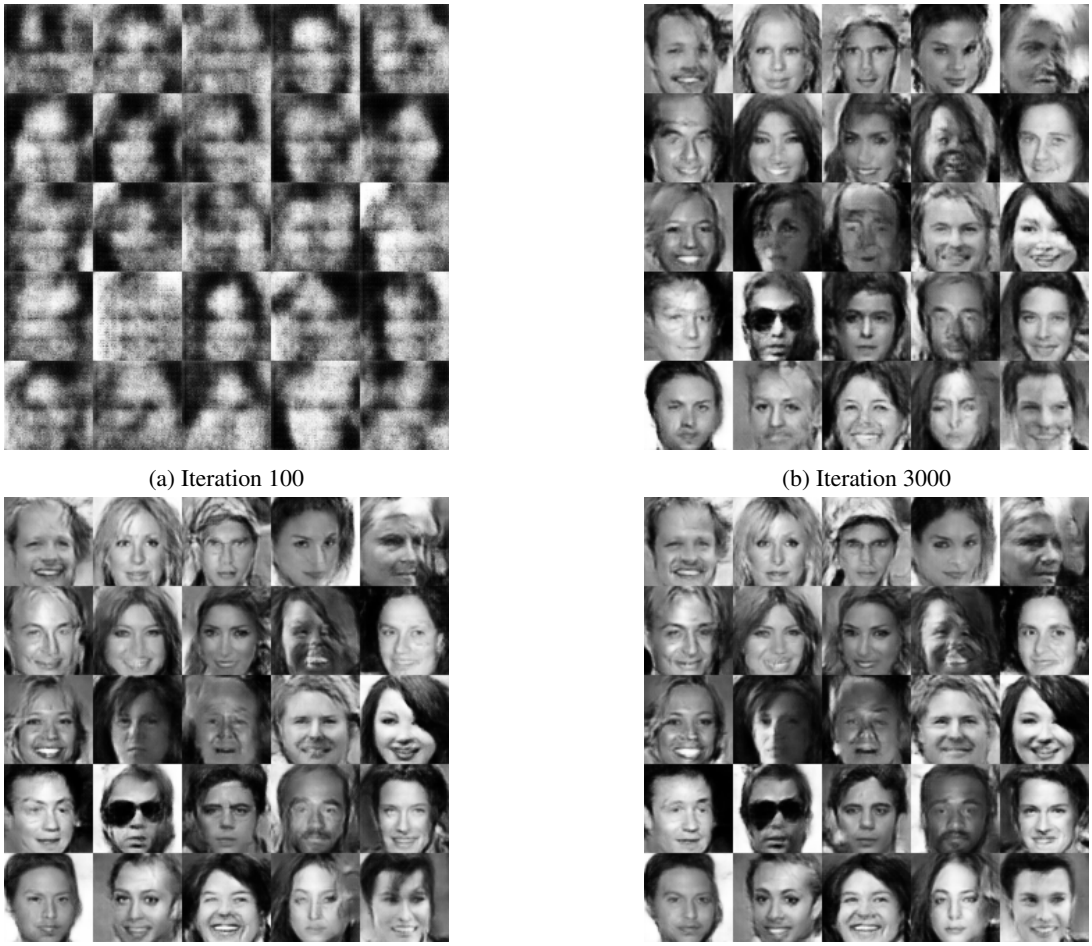


Figure 11: Training loss curve (WGAN-GP + celebA)

Below show several generated sketches from random sampled z.



(a) Iteration 100 (b) Iteration 3000

(c) Iteration 7000 (d) Iteration 10000

Figure 12: Generated sketches from the random sampled z (WGAN-GP + celebA)

Observations and Analysis: In WGAN, it introduces a weight clipping approach to add Lipschitz constraints to the critic efficiently. However, the clipping weights makes the optimization more difficult. Because with the weight clipping, most neural networks can only reach the max gradient norm value when learning the simple models. Therefore, the Lipschitz constraints with weight clipping will lead the critic to become the simpler function.

Due to the drawbacks of weight clipping, WGAN-GP removes the weight clipping of model D, instead, it uses the **Gradients Clipping**, also called **Gradient Penalty**, that is, set constraints to the gradient norm value of the critic function directly and get the new losses:

$$L_D^{WGAN\_GP} = L_D^{WGAN} + \lambda E[(|\nabla D(\alpha x - (1-\alpha G(z)))| - 1)^2] \tag{1}$$

$$L_G^{WGAN\_GP} = L_G^{WGAN}$$

Compared with the standard WGAN, the WGAN-GP can bring faster convergence speed and higher-qualified generated sketches. The stability of WGAN-GP model is also improved, and without any complicated hyper-parameters fine-tuning operation, it can succeed training models for multiple types of images and languages.

Accoridng to our practical experiments for both CUFS and celebA dataset, we find the performances are all improved with better look generated samples and loss convergences. For example, according to those gif files, WGAN-GP ones become more stable over the iterations accumulation than DCGAN, and the generated sketches are much clearer than WGAN, which proves the intuitive of WGAN-GP model.