Image Generation Based on Generative Adversarial Nets

Ke Ding

Abstract

To further research the image generation based on generative adversarial nets and dig out the key factor when generating images, we reproduce GAN, Wasserstein GAN, and Conditional GAN successively, and then adjust the hyper-parameters of WGAN. Experiments calculate the losses of objective function using different models that quantitatively measure the quality of image generation and visualize the generated data for qualitative evaluation. Among different kinds of hyper-parameters, we conclude that clipping parameter is the key factor that affects the quality of image generation. Our work additionally improves WGAN with the idea of gradient penalty and experiment to get a better result compared with original WGAN.

Keywords— Image generation, Generative model, Wasserstein GAN, Conditional GAN, Gradient penalty.

1. Introduction

Networks as a generator uses a simple distribution as input, and outputs another complex distribution. Generation tasks works well for the tasks need "creativity", which means the same input has different outputs, such as drawing, chatting robot and trajectory prediction, etc.

Generative adversarial nets are a kind of generative models via an adversarial process, where we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that judges the sample is real or fake. GAN can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition drives both teams to improve their methods until the counterfeits are indistinguishable. Since GAN's original work in 2014, millions of related works have been published subsequently.

In this report, we firstly implement **GAN**, **Wasserstein GAN (WGAN)**, and **Conditional GAN (CGAN)** to generate image on MNIST dataset. Among them, WGAN evaluates the difference between the generator's data and the real data using Wasserstein distance, rather than using Jensen-Shannon divergence in the original work. CGAN "manipulates" the output of generator with an extra priori input label, in which the discriminator D evaluates the input label and the data generated by D are matched or not.

Secondly, we adjust the hyper-parameters of WGAN, among which we analyze how the number of training step for discriminator per iteration, the lower and upper clip value, and the learning rate will affect the quality of image generation, and then summary the key factor.

Finally, we make an improvement on WGAN. Since the smoothing process of original WGAN is rough by simply restricting the parameters within a range between -c and c, we keep the gradient between real data and fake data close to 1.

In the last paragraph of Sec. 1, we summarize the crucial contribution in our work as below:

1) Reproducing GAN, WGAN, CGAN and generating images on MNIST dataset.

2) Analyzing the generation results of different hyper-parameters qualitatively and quantitatively.

3) Improving WGAN based on the idea of gradient penalty.

2. Methodology

2.1 Framework of GAN, WGAN, and CGAN

The framework of **GAN** model corresponds to a minimax 2-player game. It is most straightforward to apply when the models are both multiplayer perceptron. To learn the generator's distribution p_g over data x, we define a prior on input noise variables $p_z(z)$, then represent a mapping to data space as $G(z;\theta_g)$, where G is a differentiable function represented by a multiplayer perceptron with parameters θ_g. We also define a second multilayer perceptron $D(x;\theta_d)$ that outputs a single scalar. $D(x)$ represents the probability that x came from the data rather than p_g. We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize log(1 – D(G(z))). In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

In the view of math, the value of $arg\max_D V(D,G)$ in the above objective function is related to JS divergence. And when we train the network, a brief training process can be summarized as the pseudo code below:

```
Algorithm 1 Training process of GAN

1: Initialize $\theta$ for generator and discriminator.
2: In each training iteration do
3:     Fix generator G, and update discriminator D.
4:     Fix discriminator D, and update generator G.
```

The intuition of **WGAN** is that JS divergence of GAN is not suitable in most cases. For example, p_g and p_{dataa} are not overlapped. But JS divergence is always log2 if two distributions do not overlap, different "distances" between p_g and p_{data} will get equally bad results because binary classifier achieves 100% accuracy. So, we evaluate Wasserstein distance between p_g and p_{data}. The objective function is improved as below:

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim p(z)}[f_w(g_\theta(z)]$$

Additionally, we make a constraint that the trained weight should be within the range of -c and c, ensuring the training of *D* will converge. x

CGAN can generate MNIST digits conditioned on class labels both on generator and discriminator, which can be constructed by simply feeding the data. The discriminator in CGAN gets a high score only when the output *y* of generator is realistic and the prior label *x* and *y* are matched.

In the generator the prior input noise $p_z(z)$, and *y* are combined in joint hidden representation, and the adversarial training framework allows for considerable flexibility in how this hidden representation is composed. In the discriminator x and y are presented as inputs to a discriminative function, and the objective function of a 2-player minimax game shows as below:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))].$$

Fig 1 illustrates the structure of CGAN.

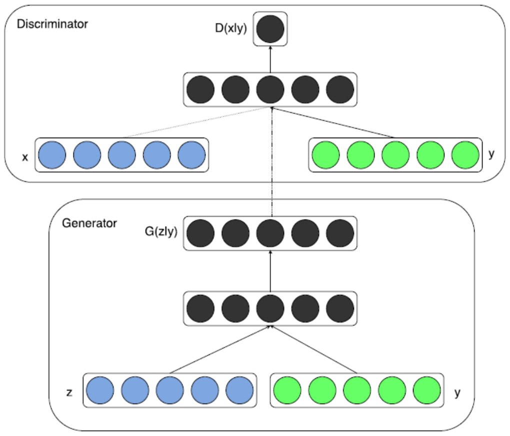

Figure 1. Structure of CGAN

2.2 Principle of network optimization

When we optimize WGAN, we mainly focus on 3 important parameters below:

***n*critic** stands for the number of training steps for discriminator per iteration.

c stands for the clipping parameter, which means the lower and upper bound for discriminator's weights. If w > *c*, then w = *c*; if w < -*c*, then w = -*c*.

α stands for the learning rate, which determines the step size at each iteration while moving toward a minimum of a loss function.

When we evaluate the results with different parameters, we evaluate both qualitatively and quantitatively.

2.3 Improvement of WGAN

The use of weight clipping in WGAN to enforce a Lipschitz constraint on the critic, which can lead to undesired behavior. So, we improve it by penalizing the norm of gradient of the critic with respect to its input and train with more stability.

We consider directly constraining the gradient norm of the critic's output with respect to its input. To circumvent tractability issues, we enforce a soft version of the constraints with a penalty on the gradient norm for random samples $\hat{x} \sim \mathrm{P}_{\hat{x}}$. The new objective function is as below:

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}\left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right]}_{\text{Our gradient penalty}}.$$

Generally, a detailed training process of Improved WGAN can be summarized as the pseudo code below:

```
Algorithm 2 Training process of WGAN with gradient penalty.

Input: Gradient penalty coefficient $\lambda$, number of critic iterations per generator iteration $n_{\text{critic}}$, batch size $m$, Adam hyper-parameters $\alpha$, $\beta_1$, $\beta_2$.
1: Initialize $\theta$ for generator and discriminator.
2: while $\theta$ has not converged do
3:     for $t = 1,\ldots, n_{\text{critic}}$ do
4:         for $i = 1,\ldots, m$ do
5:             Sample real data and latent variable.
6:             Update $\hat{x}$.
7:             Calculate $L$ according to the objective function.
8:         end for
9:         Update the critic parameters $\omega$.
10:    end for
11:    Sample a batch of latent variables.
12:    Update generator parameters $\theta$.
13: end while
```

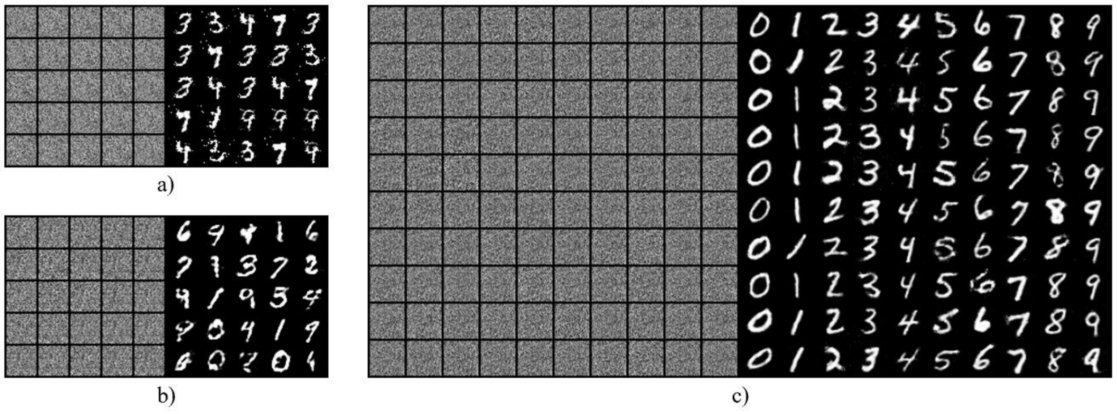

Figure 2. Visualization of the process generating MNIST digits using different models, where we input a random noise. a) GAN b) WGAN c) CGAN.

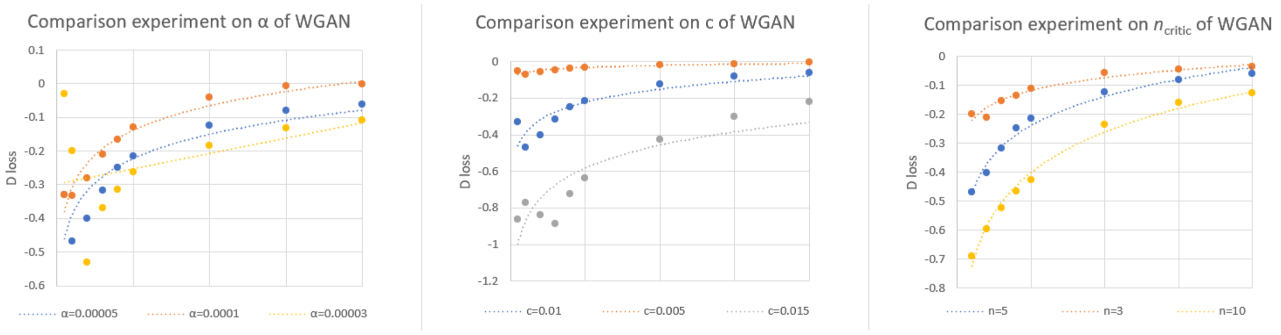

Figure 3. Contrast experiment of WGAN.

3. Experiments

Our work generates MNIST digits in Python 3.9 and PyTorch 1.10.1. When we trained GAN, WGAN, and CGAN, the generator nets use a mixture of activations, while the discriminator net uses maxout activations. We use noise as the input (prior label especially for CGAN) to only the bottommost layer of the generator.

Firstly, the results of GAN, WGAN, and CGAN are reported in Table 1, where the metrics is the loss of generator G loss and the loss of discriminator D loss at 200 epochs. As we see, G loss of GAN is relatively higher (3.698) because it doesn't converge. WGAN's losses are lower than the other two because it uses a different objective function that measures Wasserstein distance rather than JS divergence. Losses of CGAN is lower than GAN because of adding more prior information to make the objective function easier to converge.

Method	D loss	G loss
GAN	0.148	3.698
WGAN	-0.076	-0.210
CGAN	0.096	0.719

Table 1. Losses of GAN, WGAN, and CGAN

And then we visualize the results at 200 epochs, where we illustrate in Fig 2. As we see, all the 3 models generate digits from an original random noise as input. In Fig 2 a), GAN generates distinguishable digits but remains some noise at the background of the image. In Fig 2 b), the output of WGAN has no noise but some parts of the digits are not coherent and some parts are blurred. In Fig 2 c), CGAN outperforms the other two because we add a prior label, so that the machine knows to generate a known digit rather than a random digit.

Secondly, we adjust 3 different hyper-parameters listed in Sec 2.2 (increase or decrease) and the results are demonstrated in Table 2. We additionally draw the trending lines in Fig 3.

Method (WGAN)	D loss	G loss
n_{critic}=5, c=0.01, α=0.0005	-0.076	-0.210
n_{critic}=3, c=0.01, α=0.0005	-0.020	-0.149
n_{critic}=7, c=0.01, α=0.0005	-0.078	-3.570
n_{critic}=5, c=0.005, α=0.0005	-0.006	-0.010
n_{critic}=5, c=0.015, α=0.0005	-0.258	-0.987
n_{critic}=5, c=0.01, α=0.0003	-0.122	-0.260
n_{critic}=5, c=0.01, α=0.001	-0.001	0.012

Table 2. Losses of WGAN with different hyper-parameters

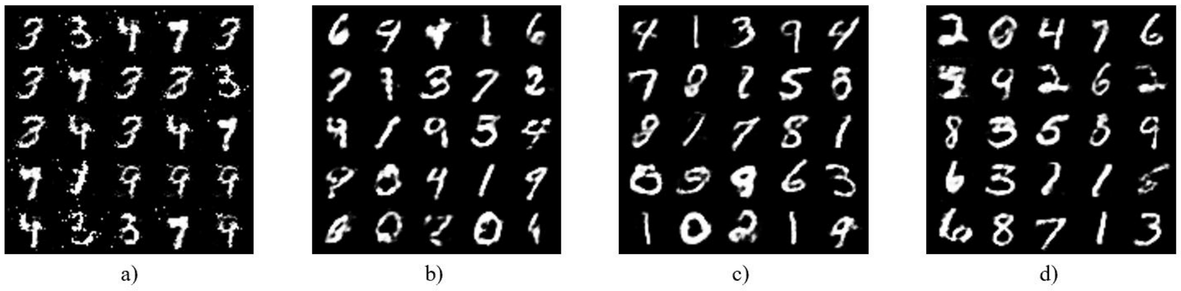

Figure 4. Visualization of Improved WGAN contrast to GAN and original WGAN. a) GAN b) original WGAN c) d) Improved WGAN.

As is shown above, we summary how the 3 hyper-parameters will affect D loss and G loss of the network:

For $\boldsymbol{n}_{\textbf{critic}}$, its increase and decrease symbolize the frequency the generator and discriminator will update. As the network are trained deeply, the update times will affect less for training parameters.

For $\boldsymbol{\alpha}$, its increase will accelerate the convergence but may cause the loss fluctuate within a range, and its decrease will slow the convergence and get worse results during the same number of training epochs. We see that choosing α as 0.0005 is relatively suitable, with lower α makes loss higher and higher α makes loss lower.

For $\boldsymbol{c}$, its increase and decrease change the restriction of the parameter's range. So, a lower c will make the network converge faster and will get a lower loss (upper for the opposite). As the core element in WGAN, c is the **key factor** affecting the quality of generated image.

Finally, we experiment on the improved WGAN and compare it with GAN and WGAN. The quantitative results evaluating on D loss and G loss are shown in Table 3 and the visualization results are shown in Fig 4.

Method	D loss	G loss
WGAN	-0.076	-0.210
Improved WGAN	-1.170	-2.664

Table 3. Losses of WGAN and Improved WGAN

As we see, although Improved WGAN performs worse at D loss and G loss in Table 3, Fig 4 demonstrates that Improved WGAN solve the remaining problem that GAN's noises and WGAN's incoherent lines to some extent. However, there remains some problems that the instability causes 2~3 generated digits out of 25 samples performs bad, which can be solved by increasing the number of training epochs. It means if we train more epochs, the generator and discriminator will "evolve" finer.

4. Conclusion

Our work researches image generation based on GAN, WGAN, and CGAN. We conclude that sometimes GAN generates images with noise and WGAN generates images with incoherent lines. Then we further analyze that clipping parameter is the key factor that affects the quality of image generation. Finally, we use gradient penalty to improve WGAN and get better results than original WGAN.

Our work finished the task well but the quality of generated digits can't be guaranteed, which is because we only train 200 epochs due to calculating resources and time. For future works possible, we admit many straightforward extensions:

1. Solve the remaining problem that 2~3 images out of 25 samples are damaged and the **network can be trained more precisely**.

2. Use generative adversarial nets for **more complicated scenarios**, such as datasets TFD, CIFAR-10, etc.

3. Try **different generative models** that catch on currently, such as Flow-based models, VAE models and diffusion models, etc.

5. References

[1] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y.. (2014). Generative Adversarial Networks.
[2] Arjovsky, M., Chintala, S., & Bottou, L.. (2017). Wasserstein GAN.
[3] Mirza, M., & Osindero, S.. (2014). Conditional Generative Adversarial Nets.
[4] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A.. (2017). Improved Training of Wasserstein GANs.