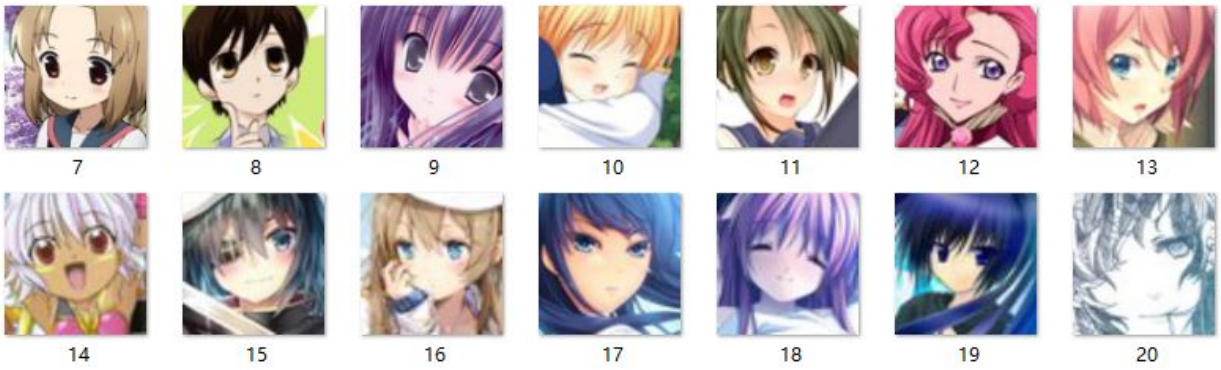
Individual Final Report - Hao Ning

Introduction

Generative Adversarial Network(GAN) is a hot topic in machine learning in recent years, it was invented by Ian Goodfellow and his colleagues in 2014[1]. It has many real life applications in Fashion, Art, Advertising, Science,Video games[2] (picture generation, image conversion, picture repair, video prediction, etc). Image generation is one of the amazing practices of GAN. In this project, a generator that produce various types of anime faces will be developed using Deep convolutional GAN, then the performance of the generator is optimized by Wasserstein GAN (WGAN) and the improved WGAN, which is the WGAN with Gradient Penalty (WGAN-GP) .The data for training are from these google drive: anime face, and extra data, image examples shown below:



All images are colored animation portraits with the same size of 96x96.

Description of Individual work

Main contribution:

- Research on improving GAN performance, using WGAN, WGAN-GP.
- WGAN model development using Pytorch
- Wasserstein distance related math explanation

Partial contribution:

- Load Data, Data prep, DCGAN, WGAN-GP code execution

[1] Goodfellow, Ian, etc. (2014). *Generative Adversarial Networks*
[2] Generative adversarial network wikipedia, https://en.wikipedia.org/wiki/Generative_adversarial_network

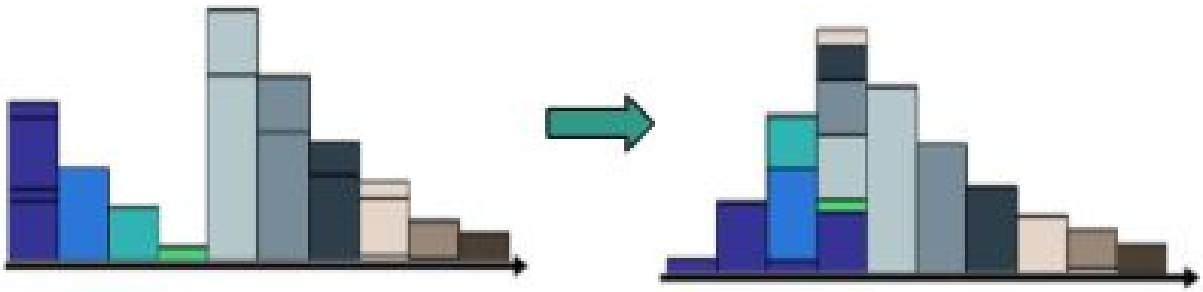
Use WGAN[3] to improve performance, the key is to change the loss function for both D and G.

Modified the DCGAN code for the implementation of WGAN. Also, the performance of training D different times each epoch are compared.

In GAN, the loss actually measures how well the generator fools the discriminator (since the output is a probability) rather than the measurement of image quality. The generator loss in GAN does not drop even when the image quality improves.

Wasserstein distance

Wasserstein Distance (or Earth Mover's distance) is a measure of the distance between two probability distributions. The best 'moving plan' is the minimum "cost" of turning one pile into the other between 2 distributions, as shown below.



JS convergence

Loss function of GAN is equivalent to JS convergence (when D at optimality). The problems in GAN can be explained with math.

First, what is the best value for D:

$$L(G,D) = \int_x \Big(p_r(x)\log(D(x)) + p_g(x)\log(1-D(x)) \Big) dx$$

$$\tilde{x} = D(x), A = p_r(x), B = p_g(x)$$

[3] https://github.com/eriklindernoren/PyTorch-GAN/blob/master/implementations/wgan/wgan.py

$$
\begin{aligned}
f(\tilde{x}) &= A log\tilde{x} + B log(1-\tilde{x}) \\
\frac{df(\tilde{x})}{d\tilde{x}} &= A\frac{1}{ln10}\frac{1}{\tilde{x}} - B\frac{1}{ln10}\frac{1}{1-\tilde{x}} \\
&= \frac{1}{ln10}(\frac{A}{\tilde{x}} - \frac{B}{1-\tilde{x}}) \\
&= \frac{1}{ln10}\frac{A-(A+B)\tilde{x}}{\tilde{x}(1-\tilde{x})}
\end{aligned}
$$

The best value of the discriminator is calculated by set the derivative to 0 :

$$D^*(x) = \tilde{x}^* = \frac{A}{A+B} = \frac{p_r(x)}{p_r(x)+p_g(x)} \in [0,1].$$

Once G is trained to optimality, Pg = Pr, thus, D = 0.5.

In other words, assuming that both D and G are optimized, discriminator can only flip a coin to tell the performance of the generator, although it's already generating samples as good as real ones.

In reality, Pg and Pr are always not overlapping for the following reasons:

1. Both Pg and Pr are low-dimensional manifold in high-dimensional space
2. Sampling

Now, the problem of JS divergence can be explained mathematically.

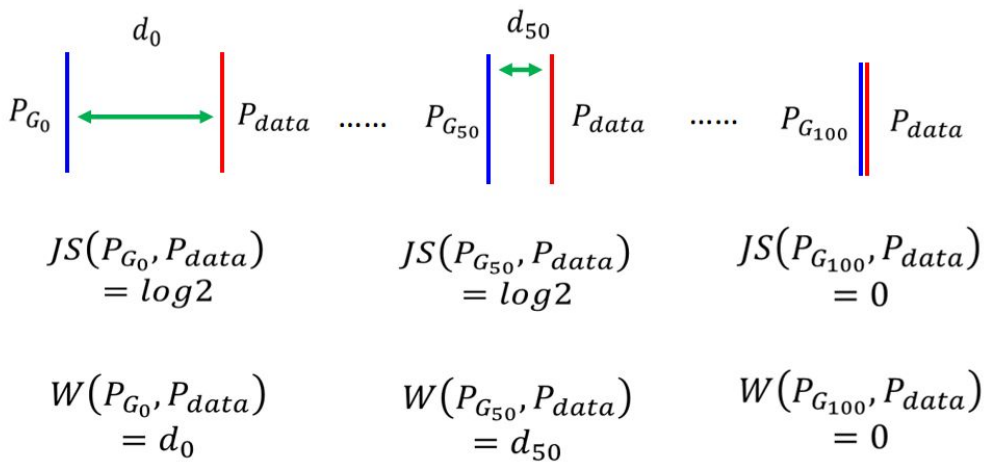
When 2 distribution are not overlapping, JS :

$$KL(P_1||P_2) = \mathbb{E}_{x\sim P_1}\log\frac{P_1}{P_2} \qquad JS(P_1||P_2) = \frac{1}{2}KL(P_1||\frac{P_1+P_2}{2}) + \frac{1}{2}KL(P_2||\frac{P_1+P_2}{2})$$

$$D_{JS}(\mathrm{P_r}, \mathrm{P_g}) = 1/2(\sum_{x=0,\, y\sim U(0,1)} 1\cdot\log\frac{1}{1/2} + \sum_{x=0,\, y\sim U(0,1)} 1\cdot\log\frac{1}{1/2}) = \log 2$$

The JS convergence is always log2, so there will be no gradient for optimization.

Comparing Wasserstein distance to JS convergence:



Wasserstein distance can effectively calculate the difference between Pg and Pr even when they are not overlapping.

To get Wasserstein distance, the objective function will be

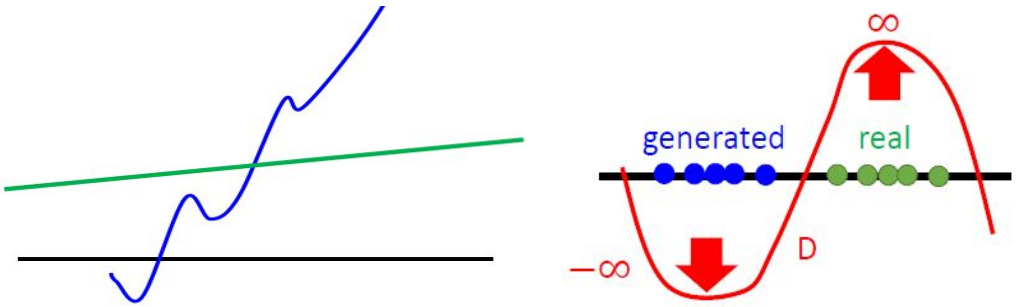
$$V(G,D) = \max_{D \in 1-Lipschitz} \{E_{x \sim P_{data}}[D(x)] - E_{x \sim P_G}[D(x)]\}$$

1-Lipschitz function is defined as:

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|$$

For 1-Lipschitz function, it should be continuously differentiable everywhere. The output change should be less or equal to the input change. As shown bottom left, the green line is a smooth function, while the blue line is not.

Since D has to be a smooth function, WGAN proposed that using weight clipping (force the parameters w between c and -c) to provide a constraint, otherwise, as shown bottom right, the output of real data will become +infinity, the generated data will be going -infinity, then D will not converge. Note that D with weight clipping here is not a strict 1-Lipschitz function, it's an 'engineering' approach (also for WGAN-GP).



WGAN algorithm[4] and Implementation

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Two significant improvement of WGAN:

- No sign of mode collapse in experiments
- The generator can still learn when the critic performs well

WGAN is still very similar to the discriminator D in DCGAN, but with a few adjustments. There's no sigmoid function at output of the discriminator, thus the output is a scalar score, that can be interpreted as how real the input images are, rather than a probability. Regarding the generator

[4] https://arxiv.org/pdf/1701.07875.pdf

G, we want to maximize the D's output for its fake instances. The changes compared to vanilla GAN are summarized as follows:

- Loss function, no log
- D: no sigmoid
- Clip the weight of D (-c,c), if w>c, w=c, if w<-c, w=-c
- Use RMSProp
- Train D more than G

	Discriminator	Generator
DCGAN	$\widetilde{V} = \frac{1}{m}\sum_{i=1}^{m} logD(x^i) + \frac{1}{m}\sum_{i=1}^{m}(1 - logD\left(G(z^i)\right))$	$\widetilde{V} = \frac{1}{m}\sum_{i=1}^{m}(1 - logD\left(G(z^i)\right))$
WGAN	$\widetilde{V} = \frac{1}{m}\sum_{i=1}^{m}(D(x^i) - D\left(G(z^i)\right))$	$\widetilde{V} = \frac{1}{m}\sum_{i=1}^{m} D\left(G(z^i)\right)$

Hyperparameters:

- Clip Weight at (-0.01,0.01), Batch size 128, Learning rate 0.0001

The code execution time is 4-10 hours depending on D training times (for each epoch, train D once/twice/4 times, G only once). WGAN Generated Images shown below:

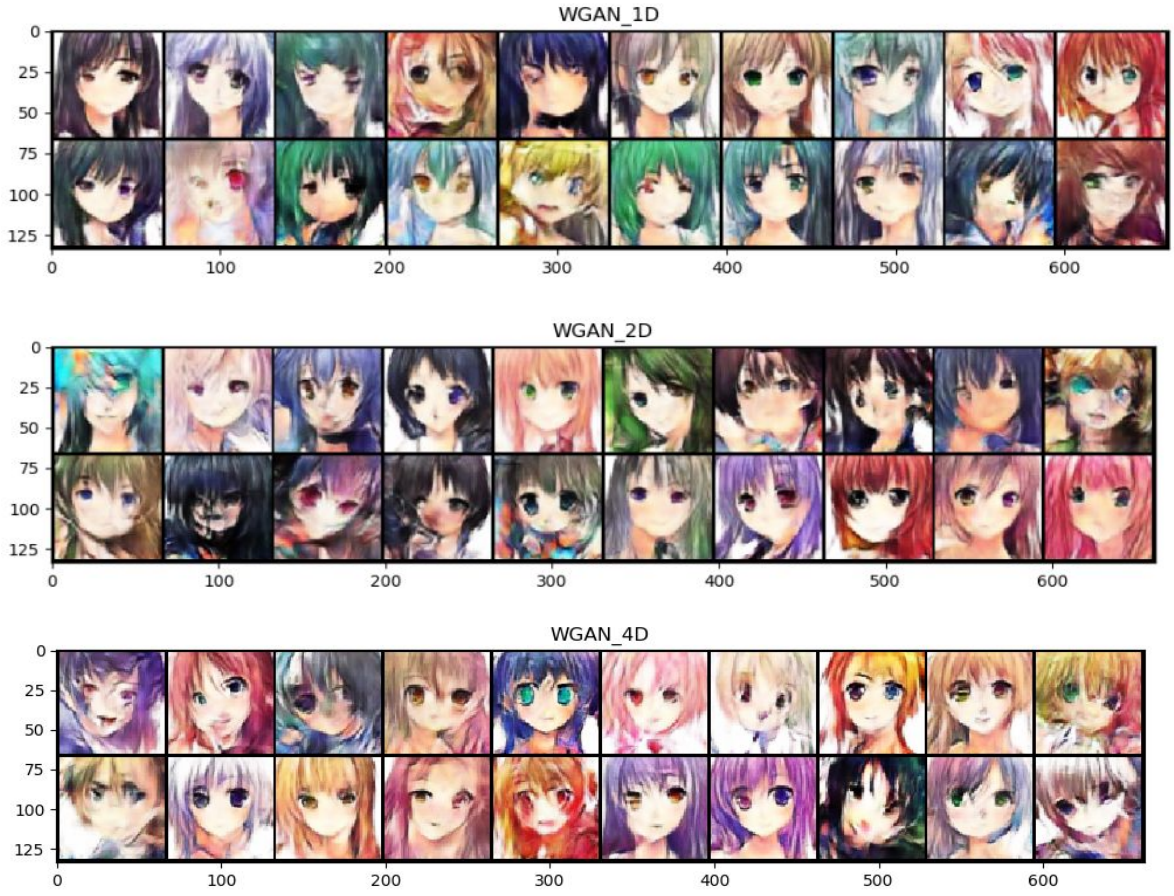
Left: Epoch 10 Right: Epoch 50

The image quality is improving and no mode collapse observed.

However, there're still some very bad quality images, probably due to the effectiveness of the weight clipping in this method.

Comparison of Train D more than G

From our observation for WGAN, there's no obvious differences when train D more than G, perhaps 4D is slightly better. This makes sense since weight clipping parameters have a great impact on the model performance.



Code

Main coding reference is from this Github, it has examples of all kinds of GAN.
https://github.com/eriklindernoren/PyTorch-GAN
The code is greatly modified for our own architecture and model.
Less than 20% is from or copied directly from the internet.

References.

1. Hongyi Lee, GAN 2018 https://www.youtube.com/watch?v=DQNNMiAP5Iw&list=PLJV_el3uVTsMq6JEFPW35BCiOQTsoqwNw
2. GANHACKS, https://github.com/soumith/ganhacks
3. PyTorch-GAN https://github.com/eriklindernoren/PyTorch-GAN
4. GAN — Wasserstein GAN & WGAN-GP https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490
5. From GAN to WGAN https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html
6. Wasserstein GAN https://arxiv.org/abs/1701.07875
7. Improved Training of Wasserstein GANs https://arxiv.org/abs/1704.00028