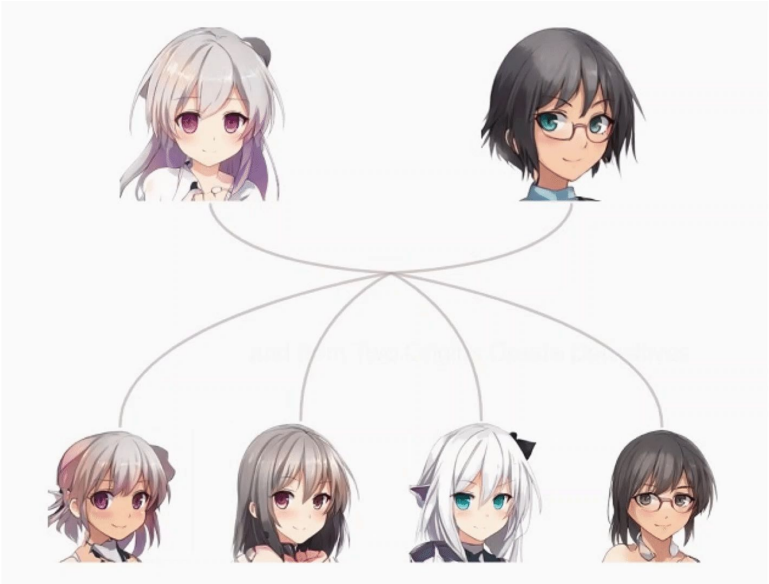
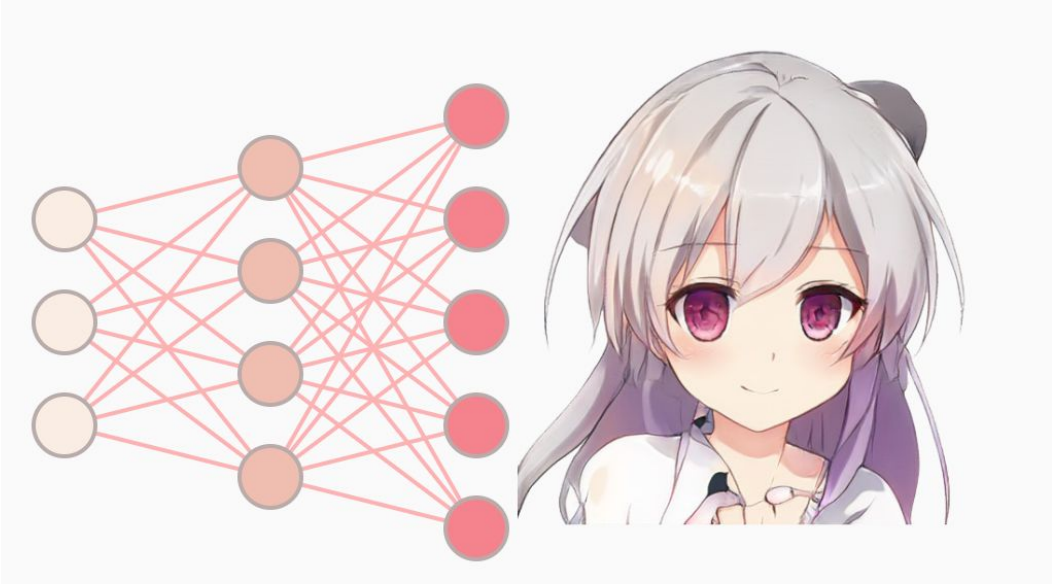
DATS6203 Machine Learning II Final Project
Anime Face Generator using Generative Adversarial Networks (GAN)



Group 2:
Chen Chen, Hao Ning, Hungchun Lin

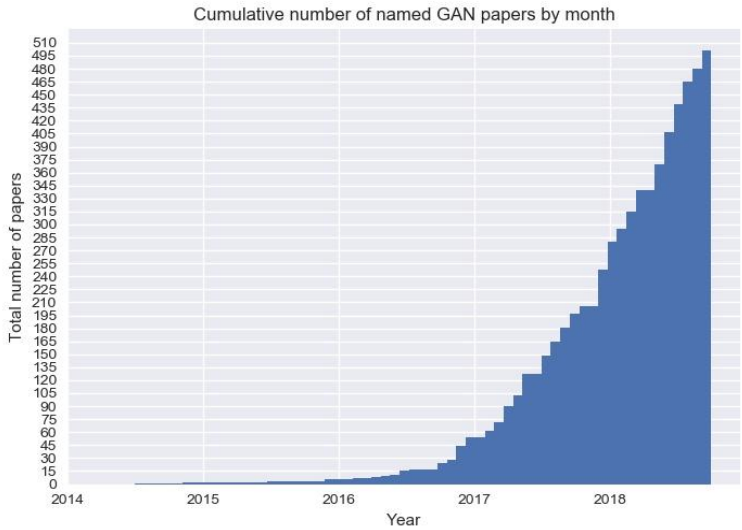
Outline

- Introduction
- Data Description
- GAN Theory/Architecture
- Model Description
  - DCGAN
  - WGAN
  - WGAN-GP
- Summary



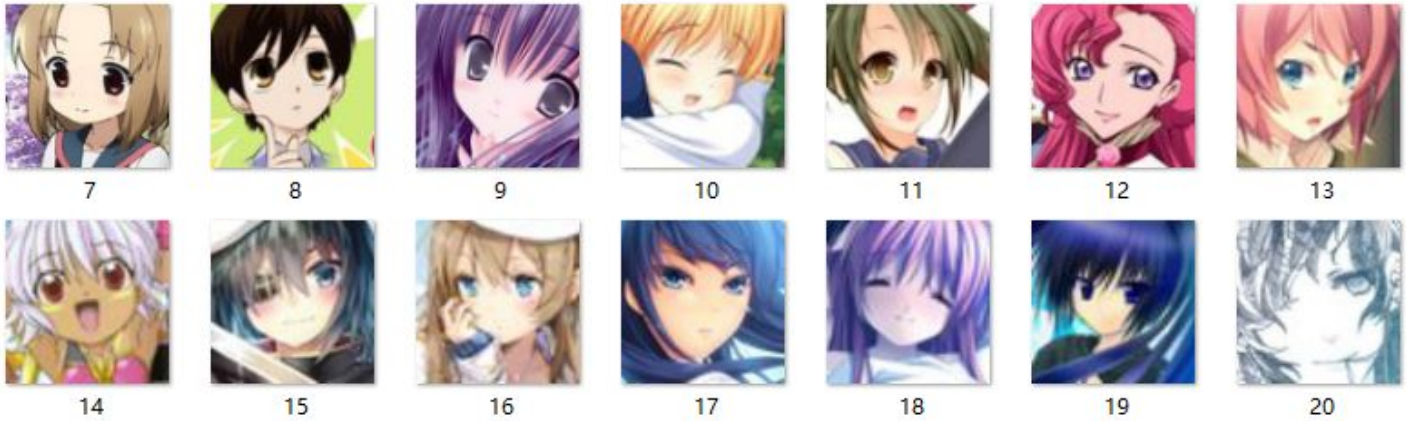
Introduction

- Generative Adversarial Network
  - Hot Topic in Machine Learning in recent years
- Applications in Fashion, Art, Advertising, Science,Video games, etc.
  - Final Fantasy VIII, Final Fantasy IX HD
  - Generate picture/character
  - Image conversion
  - Picture repair
  - Video prediction … ...



- Project Goal
  - Create a Generator that can produce various high quality anime face
- Challenges
  - Relatively new technology, training process, network performance, etc.

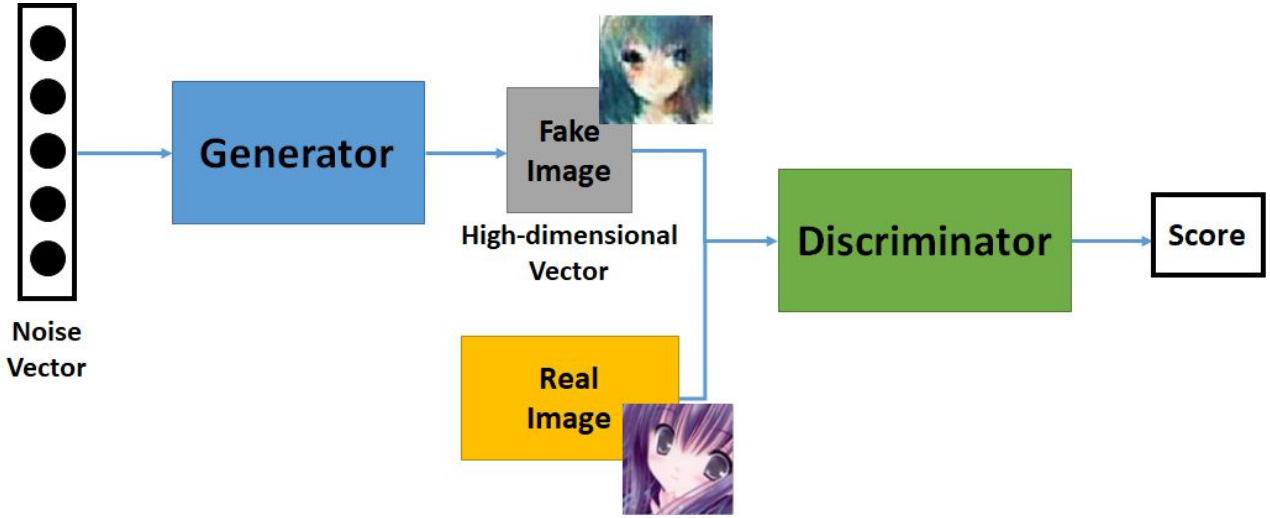
Data Description



- No. of Images of Animation portrait: 70,000+
- Image size: 96 x 96
- Data collected by University of Taiwan
  - AnimeDataset https://drive.google.com/drive/folders/1mCsY5LEsgCnc0Txv0rpAUhKVPWVkbw5l
  - Extra_data https://drive.google.com/file/d/1tpW7ZVNosXsIAWu8-f5EpwtF3ls3pb79
  - Web scraping Using https://github.com/nagadomi/lbpcascade_animeface

GAN Theory

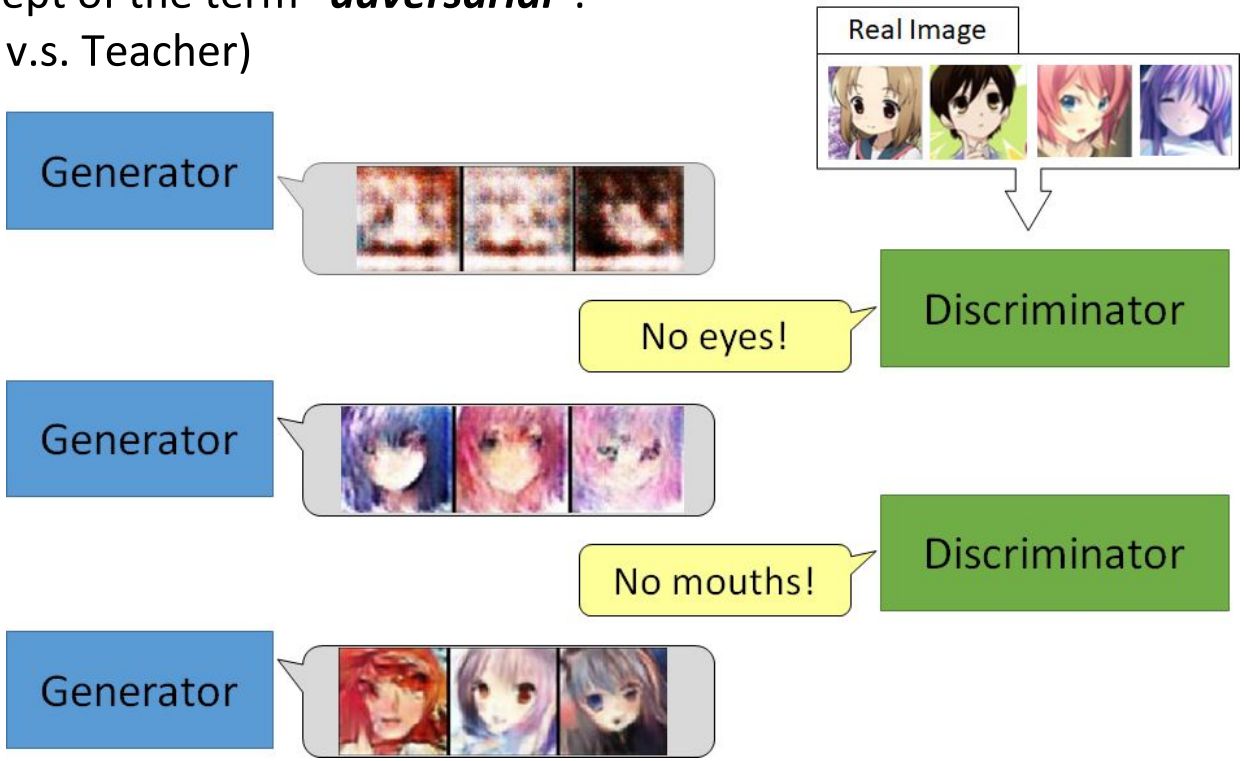
Basic Idea of GAN



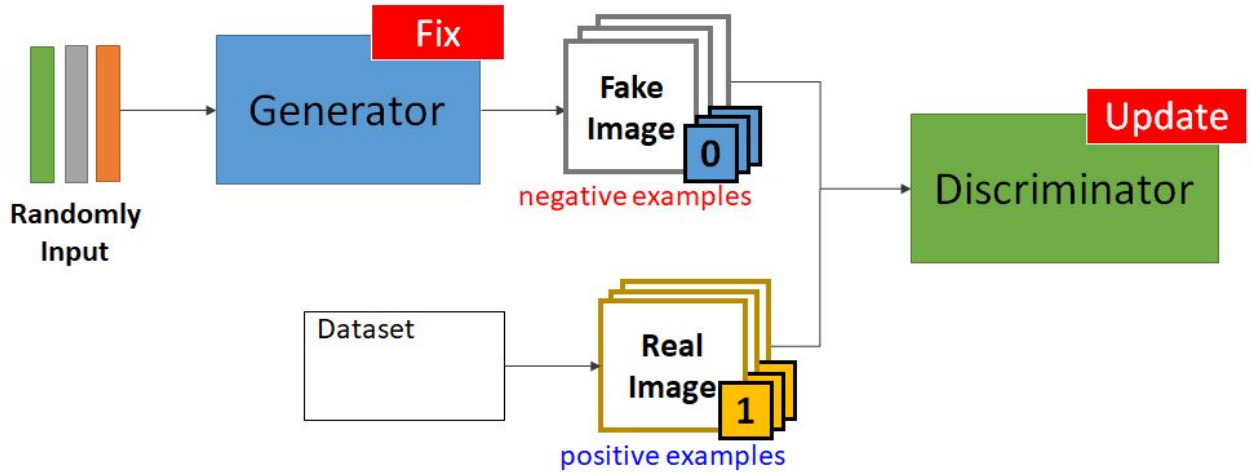
- GANs are basically made up of a system of two competing neural network models that compete with each other.
- The Generator generates fake samples of data, and tries to fool the Discriminator.
- The Discriminator tries to distinguish between the real and fake samples.
- The steps are repeated several times and in this, the Generator and Discriminator get better and better in their respective jobs after each repetition.

Basic Idea of GAN cont.

The concept of the term "***adversarial***".
(Student v.s. Teacher)

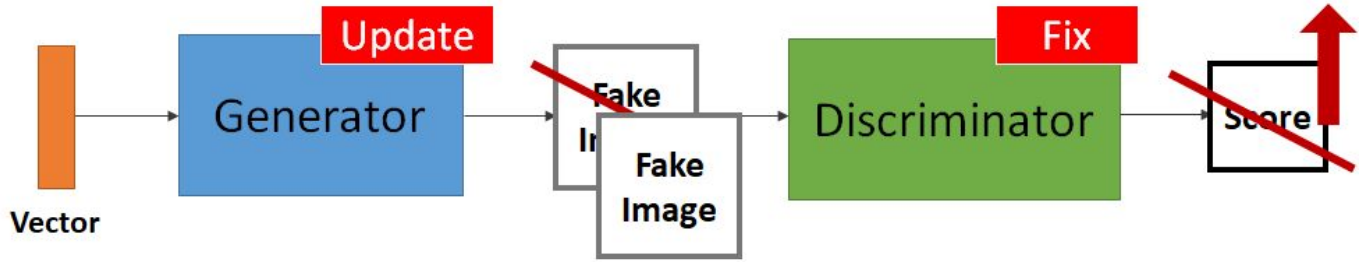


GAN - Discriminator Training



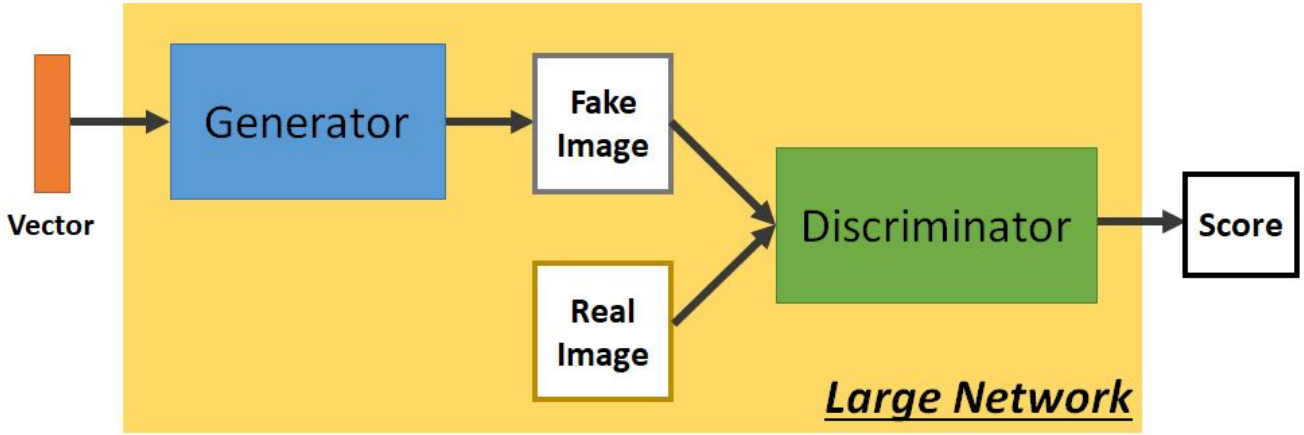
- Initialize Generator and Discriminator
- Fix Generator
- Update Discriminator
- Discriminator learns to assign high scores to real objects and low score to generated objects

GAN - Generator Training



- Initialize Generator and Discriminator
- Fix Discriminator
- Update Generator
- Generator learns to “fool” the discriminator

Structure of GAN



- Combine the Generator and Discriminator, it can be seen as a large Network
- A GAN is built by a Generator that is used for upsampling, and a Discriminator which is used for downsampling

GAN Math Theory

- x: original data
- z: noise or random input
- $\tilde{x}^i = \mathrm{G}(z^i)$: generated data

Learning D

- Update discriminator parameters θ_d to maximize
  - $\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} logD(x^i) + \sum_{i=1}^{m}(1 - logD\left(\mathrm{G}(z^i)\right))$
  - $\theta_d \leftarrow \theta_d + \eta\nabla\tilde{V}(\theta_d)$

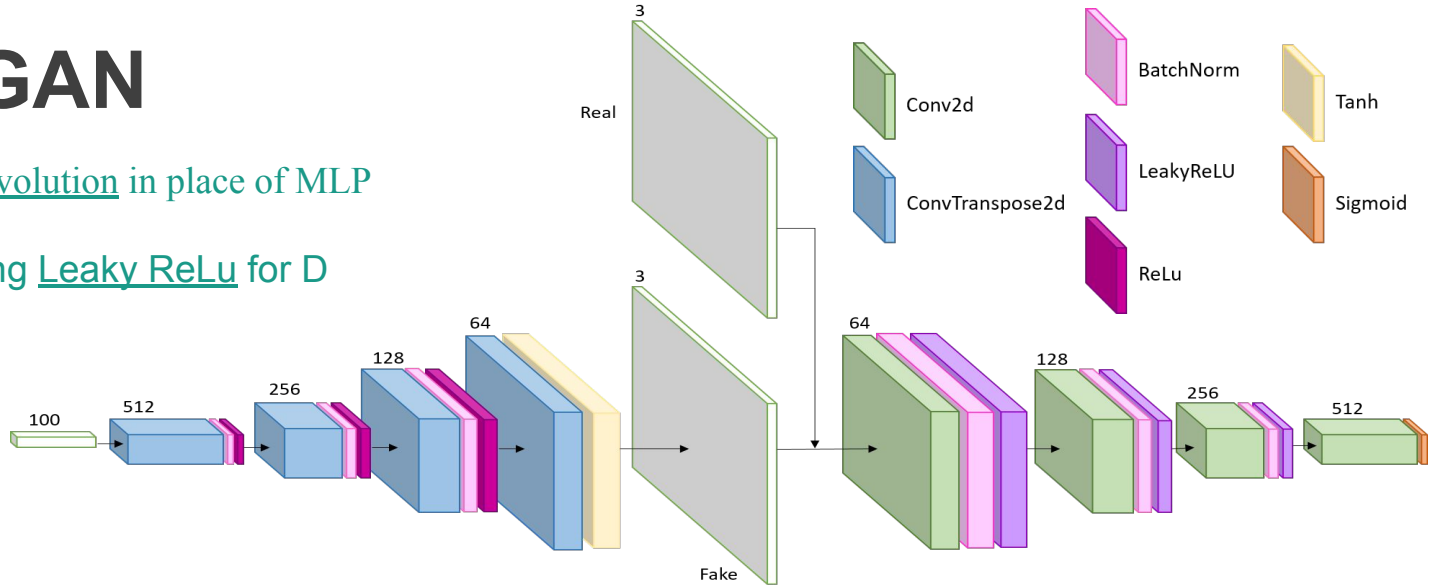
Learning G

- Update generator parameters θ_g to minimize
  - $\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} \log(1 - D\left(G(z^i)\right))$
  - $\theta_g \leftarrow \theta_g - \eta\nabla\tilde{V}(\theta_g)$

Model Description

DCGAN

- Convolution in place of MLP
- Using Leaky ReLu for D



Generator	Output size	Discriminator	Output size
Input noise vector: Z(100)		Input: Image(64*64*3)	
Linear(100) + BN + ReLu	(512,4,4)	Conv2d (3, (5,5), stride = 2) + BN + LeakyReLu	(64, 32, 32)
Conv2d_transpose(512, (5,5), stride = 2) + BN+ ReLu	(256,8,8)	Conv2d (64, (5,5), stride = 2) + BN + LeakyReLu	(128, 16, 16)
Conv2d_transpose(256, (5,5), stride = 2) + BN+ ReLu	(128,16,16)	Conv2d (128, (5,5), stride = 2) + BN + LeakyReLu	(256, 8, 8)
Conv2d_transpose(128, (5,5), stride = 2) + BN+ ReLu	(64,32,32)	Conv (256, (5,5), stride = 2) + BN + LeakyReLu	(512, 4, 4)
Conv2d_transpose(64, (5,5), stride = 2) + Tanh	(3,64,64)	Conv (512, (4,4), stride = 1) + Sigmoid	(1)

Mode Collapse

Epoch 25

Epoch 50

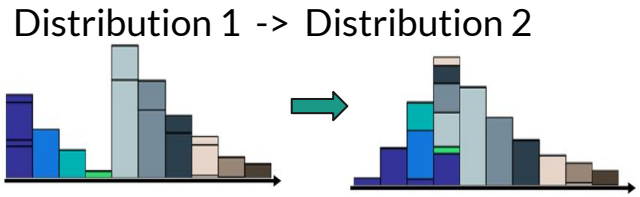
Problems of DCGAN

- Mode collapse
  - Generator only produces low-diversity output.
  - When input a different z, but G(z) does not change.

- Learning not stable
  - If the discriminator D gets stronger quickly, then the gradient of the loss function is 0, learning stopped. (D(x)= 1 , D(G(z)) =0), at generator G → log(1 — D(G(z))) = log(1–0) = 0)
  - If the discriminator D gets too weakly , then the generator G does not have good feedback so the loss represent nothing much.

WGAN Background

- Wasserstein distance as GAN loss function
  - Wasserstein(Earth Mover) Distance
    - Best moving plan, least distance (least energy cost)
    - Prevent vanishing gradients

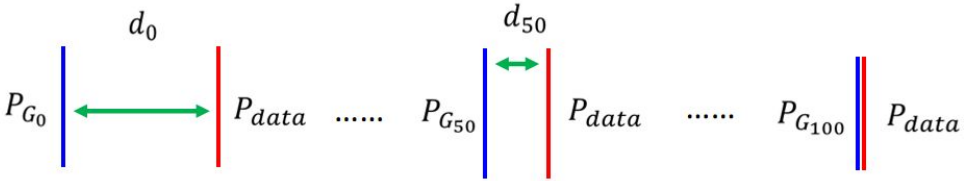


- Evaluate distance between real data and generated data distribution
  - Give a score instead of probability
- D Objective function D(x) - D(G(z))
  - Must be a smooth function
  - Maximize the the real fake difference
- G loss: D(G(z))
  - Maximize the D's output for its fake instances

Clip the weight to have a smooth function, differentiable

JS Convergence vs Wasserstein

- Loss function of GAN is equivalent to JS convergence(D at optimality)
- JS is always log2 if two distributions do not overlap (0 gradient)



$JS(P_{G_0}, P_{data}) = log2$

$JS(P_{G_{50}}, P_{data}) = log2$

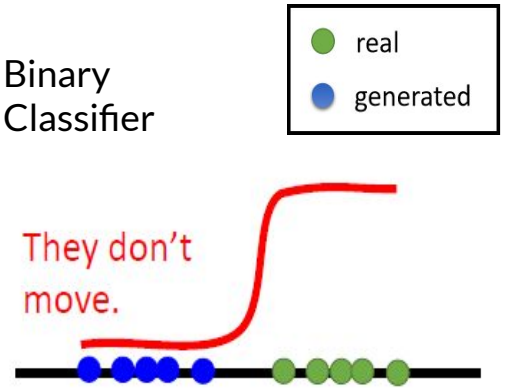
$JS(P_{G_{100}}, P_{data}) = 0$

$W(P_{G_0}, P_{data}) = d_0$

$W(P_{G_{50}}, P_{data}) = d_{50}$

$W(P_{G_{100}}, P_{data}) = 0$

Binary Classifier

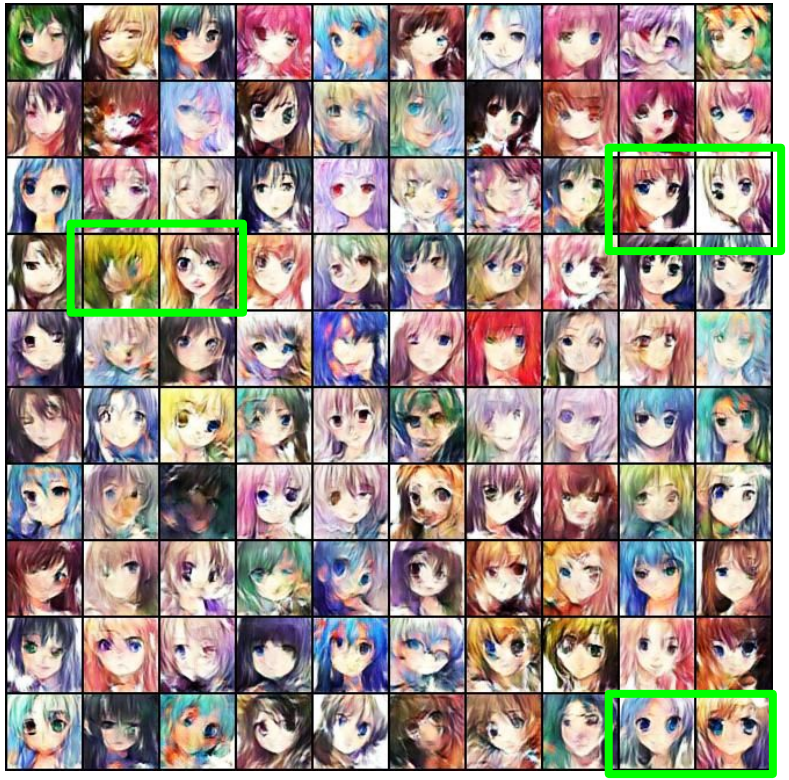


Implementation of WGAN

- Compared to Vanilla GAN
  - 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
- Training Process
  - Clip Weight at (-0.01,0.01)
  - Batch size 128
  - Learning rate 0.0001
  - 4-10 hours depends train D times

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

WGAN Generated Images

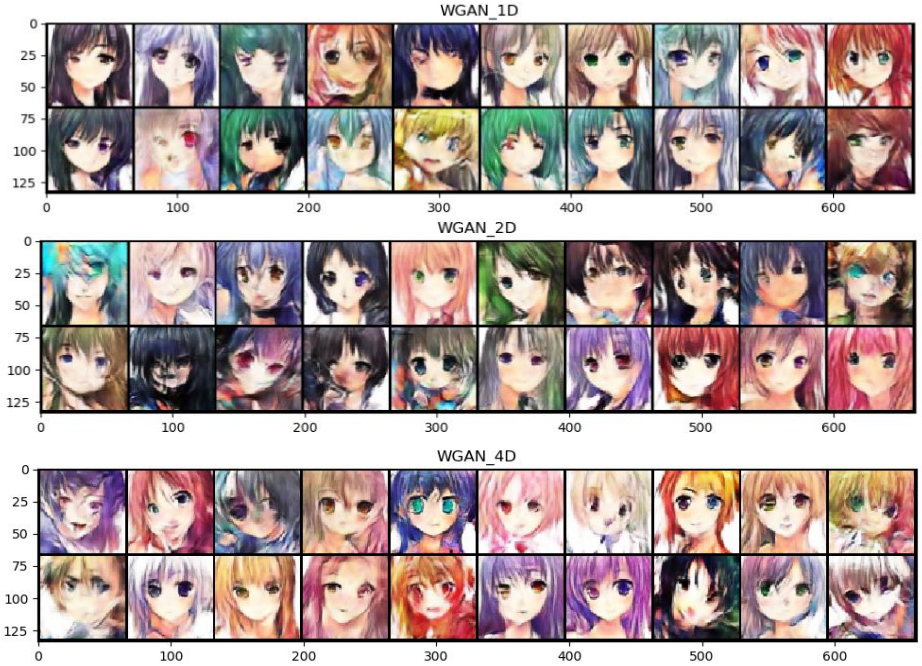


Epoch 10

Epoch 50

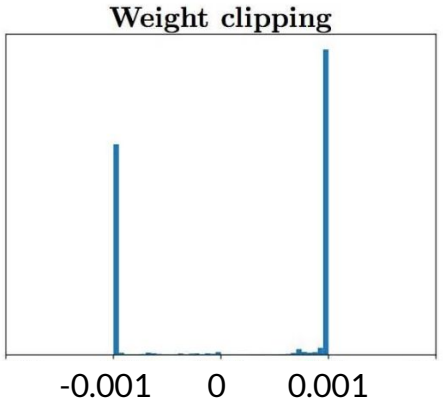
Train D more than G

- No obvious differences observed



Problems of WGAN

- The model is sensitive to clipping weight, in particular when the hyperparameter c is not tuned correctly.



- clipping is large, it can take a long time for any weights to reach their limit
- clipping is small, most of the weight are forced to the boundary value, this can easily lead to vanishing gradients

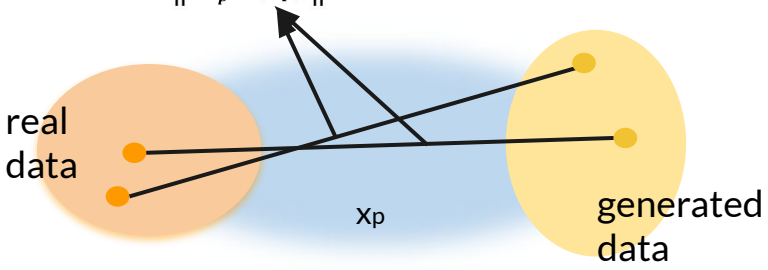
WGAN-GP

- Uses gradient penalty instead of the weight clipping

	Discriminator	Generator
WGAN	$\tilde{V} = \frac{1}{m}\sum_{i=1}^{m}(D(x^i) - D\left(G(z^i)\right))$ ***with weight clipping***	$\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} D\left(G(z^i)\right)$
WGAN-GP	$\tilde{V} = \frac{1}{m}\sum_{i=1}^{m}(D(x^i) - D\left(G(z^i)\right) +$ ***Gradient Penalty*** $)$	$\tilde{V} = \frac{1}{m}\sum_{i=1}^{m} D\left(G(z^i)\right)$

- Gradient penalty: penalizes the model if the gradient norm of discriminator moves away from its target norm value 1.

Gradient penalty: $\lambda(\|\nabla_{x_p} D(x_p)\| - 1)^2$



Implement of WGAN-GP

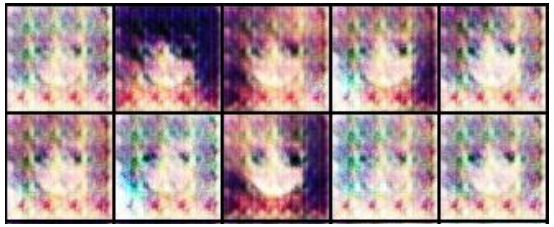
- Compared to WGAN
  - Remove batch-normalization in discriminator
  - No Clip the weight, use gradient penalty
  - Use Adam as optimizer

Generator	Output size	Discriminator	Output size
Input noise vector: Z(100)		Input: Image(64*64*3)	
Linear(100) + BN + ReLu	(512,4,4)	Conv2d (3, 64, (5,5), stride = 2) + LeakyReLu	(64, 32, 32)
Conv2d_transp(512, 256, (5,5), stride = 2) + BN+ ReLu	(256,8,8)	Conv2d (64, 128, (5,5), stride = 2) + LeakyReLu	(128, 16, 16)
Conv2d_transp(256, 128, (5,5), stride = 2) + BN+ ReLu	(128,16,16)	Conv2d (128, 256, (5,5), stride = 2) + LeakyReLu	(256, 8, 8)
Conv2d_transp(128, 64, (5,5), stride = 2) + BN+ ReLu	(64,32,32)	Conv (256, 512, (5,5), stride = 2) + LeakyReLu	(512, 4, 4)
Conv2d_transp(64, 3, (5,5), stride = 2) + Tanh	(3,64,64)	Conv (512, 1, (4,4))	1

Tricks to improve performance

--- increase λ

Epoch 10



Epoch 30

Epoch 50

λ= 0.5

λ= 5

Tricks to improve performance
--- train discriminator more

Epoch 10

Epoch 30

Epoch 50

Train D once , $\lambda = 5$

Train D twice, $\lambda = 5$

WGAN-GP Compare to WGAN

WGAN

Epoch 50

WGAN-GP

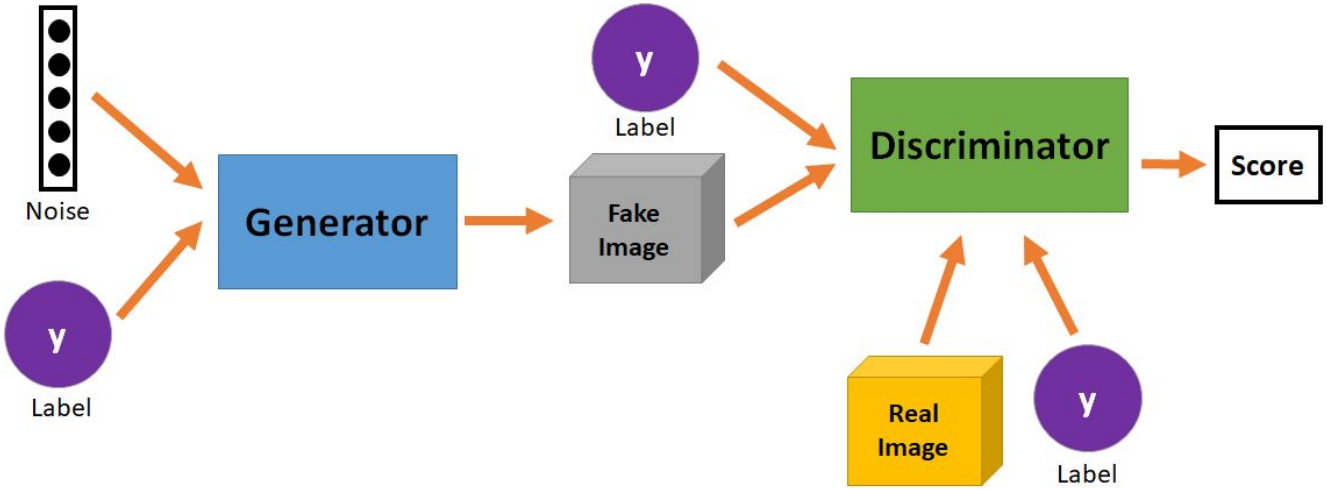
Epoch 50

Batch size: 128
Learning rate: 0.0001
Train D twice G once

Summary

- GAN theory, architecture tutorial
- Improve GAN performance
  - WGAN, WGAN-GP by changing loss function
    - No Mode Collapse
    - Better yield
    - WGAN-GP best quality and highest yield
  - Apply GAN tips such as train D more than G
- Future work
  - Further improve loss function, such as spectral normalization GAN
  - Conditional GAN (CGAN) shown next page

Future Work - CGAN



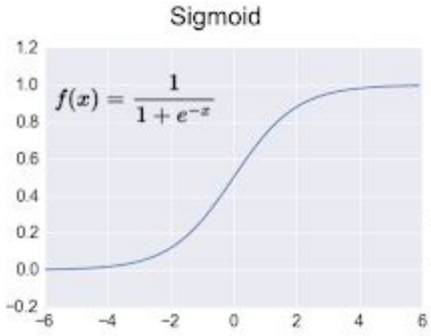
- The input is combined by some conditional parameters (y)
- Parameter y is added to both the Generator and Discriminator

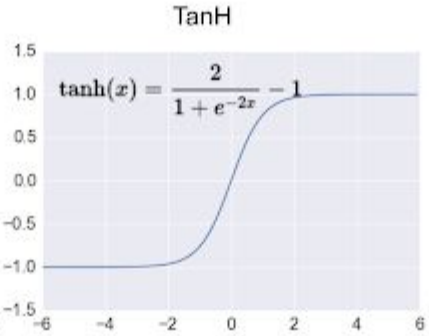
References

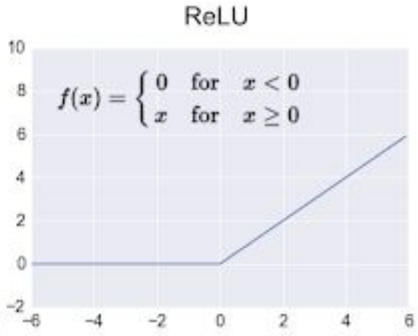
1. Hongyi Lee, GAN 2018 https://www.youtube.com/watch?v=DQNNMiAP5Iw&list=PLJV_el3uVTsMq6JEFPW35BCiOQTsoqwNw
2. DCGAN https://github.com/carpedm20/DCGAN-tensorflow
3. GANHACKS https://github.com/soumith/ganhacks
4. Wasserstein GAN https://arxiv.org/abs/1701.07875
5. Improved Training of Wasserstein GANs https://arxiv.org/abs/1704.00028
6. GAN — Wasserstein GAN & WGAN-GP https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490
7. From GAN to WGAN https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html

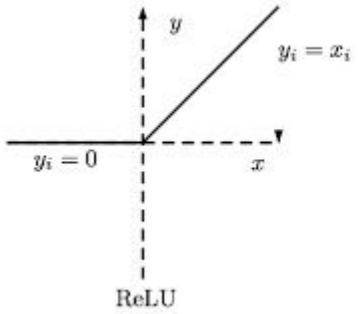

NO PAIN
NO GA N

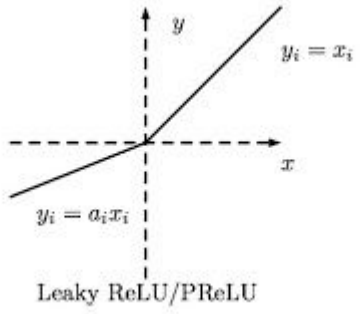
Appendix

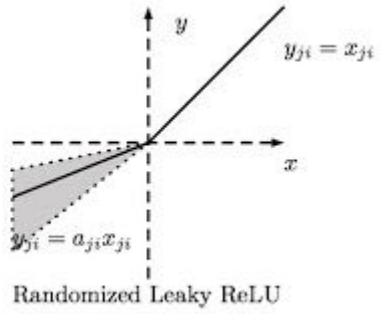

Sigmoid
$f(x) = \frac{1}{1+e^{-x}}$


TanH
$\tanh(x) = \frac{2}{1+e^{-2x}} - 1$


ReLU
$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$


y
$y_i = x_i$
$y_i = 0$
x
ReLU


y
$y_i = x_i$
x
$y_i = a_i x_i$
Leaky ReLU/PReLU


y
$y_{ji} = x_{ji}$
x
$y_{ji} = a_{ji} x_{ji}$
Randomized Leaky ReLU

JS Convergence

What is the best value for D

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

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

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

Thus, set $\frac{df(\tilde{x})}{d\tilde{x}}=0$, we get the best value of the discriminator:

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

Once the generator is trained to its optimality, pg gets very close to pr. When pg=pr, D(x) =0.5.
No judging, coin flipping!

When both G and D are at their optimal values, we have $p_g=p_r$ and $D^*(x)=1/2$ and the loss function becomes:

$$\begin{aligned}L(G,D^*) &= \int_x\Big(p_r(x)\log(D^*(x))+p_g(x)\log(1-D^*(x))\Big)dx\\ &= \log\frac{1}{2}\int_x p_r(x)dx+\log\frac{1}{2}\int_x p_g(x)dx\\ &= -2\log 2\end{aligned}$$

According to the formula listed in the previous section, JS divergence between p_r and p_g can be computed as:

$$\begin{aligned}D_{JS}(p_r\|p_g) =& \frac{1}{2}D_{KL}(p_r||\frac{p_r+p_g}{2})+\frac{1}{2}D_{KL}(p_g||\frac{p_r+p_g}{2})\\ =& \frac{1}{2}\Big(\log 2+\int_x p_r(x)\log\frac{p_r(x)}{p_r+p_g(x)}dx\Big)+\\ & \frac{1}{2}\Big(\log 2+\int_x p_g(x)\log\frac{p_g(x)}{p_r+p_g(x)}dx\Big)\\ =& \frac{1}{2}\Big(\log 4+L(G,D^*)\Big)\end{aligned}$$

Thus,

$$L(G,D^*)=2D_{JS}(p_r\|p_g)-2\log 2$$

$$KL(P_1||P_2)=\mathbb{E}_{x\sim P_1}\log\frac{P_1}{P_2}$$

$$JS(P_1||P_2)=\frac{1}{2}KL(P_1||\frac{P_1+P_2}{2})+\frac{1}{2}KL(P_2||\frac{P_1+P_2}{2})$$

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

W Distance

When treating x as the starting point and y as the destination, the total amount of dirt moved is $\gamma(x, y)$ and the travelling distance is $\|x - y\|$ and thus the cost is $\gamma(x, y) \cdot \|x - y\|$. The expected cost averaged across all the (x, y) pairs can be easily computed as:

$$\sum_{x,y} \gamma(x, y)\|x - y\| = \mathbb{E}_{x,y\sim\gamma}\|x - y\|$$

WGAN 1-Lipschitz function

- 1-Lipschitz function:
  - Output change is less or equal to the input change
  - Functions that are everywhere continuously differentiable is Lipschitz continuous
  - Weight clip/Gradient Penalty are both “engineering approach”, not a strict 1-Lipschitz

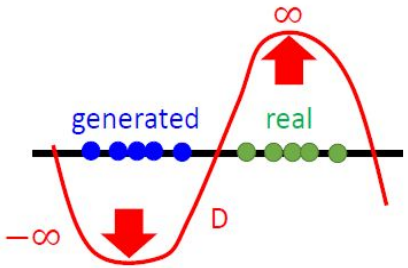
$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|$$

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

D has to be smooth enough.

Need a smooth function, need constraint

Without the constraint, the training of D will not converge.



WGAN Algorithm

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

WGAN-GP Algorithm

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Gradient penalty

A differentiable function f is 1-Lipschitz if and only if it has gradients with norm at most 1 everywhere.

f^* has gradient norm 1 almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$.

In specific, Appendix A in WGAN-GP paper proves that

if f^* is differentiable , $\pi(x = y) = 0$, and $x_t = (1-t)y$ with $0 \le t \le 1$,
it holds that $\mathbb{P}_{(x,y)\sim\pi}\left[\nabla f^*(x_t) = \frac{y - x_t}{\|y - x_t\|}\right] = 1.$

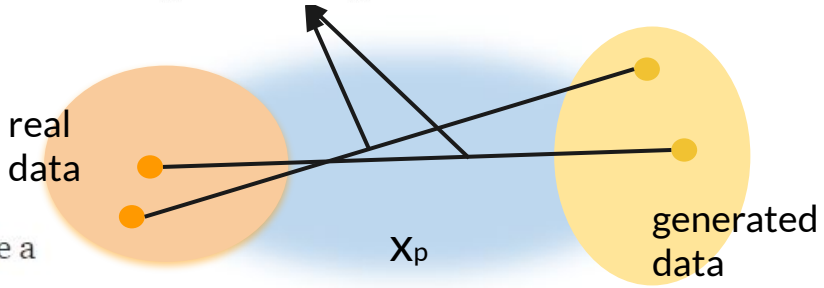
Source

Points interpolated between the real and generated data should have a gradient norm of 1 for f.

To calculate Wasserstein meet the requirement of 1 - Lipschitz function.

penalizes the model if the gradient norm moves away from its target norm value 1.

$$\lambda\left(\left\|\nabla_{x_p} D(x_p)\right\| - 1\right)^2$$



Batch normalization is **avoided** for the critic (discriminator). Batch normalization creates correlations between samples in the same batch. It impacts the effectiveness of the gradient penalty which is confirmed by experiments.