GANs (Generative Adversarial Networks)

By Yash Belhe, Hao Liang
Agenda

- Generative models
- Revisiting GANs
- WGAN
- WGAN-Gradient penalty (WGANGP)
  - Code walk through GANS, WGAN, WGANGP
- Cycle GAN
  - Code walk through Cycle GAN
- STAR GAN
  - Code walk through STAR GAN
Generative Models

Basic idea is to learn the underlying distribution of the data and generate more samples for the distribution.

Some examples of generative models

- Probabilistic Graphical Models
- Bayesian Networks
- Variational Autoencoder
- Generative Adversarial Networks
Generative Models

- Unknown distribution $P_r$ (r for real)
- Known distribution $P_\theta$
- Two approaches
  - Optimise $P_\theta$ to estimate $P_r$
  - Learn a function $g_\theta(Z)$ which transforms $Z$ into $P_\theta$
Approach 1: Optimise $P_\theta$ to estimate $P_r$

- Maximum Likelihood Estimation (MLE): 
  - This is same as minimizing the KL divergence
- Kullback-Leibler (KL) divergence: 
  $$KL(P\|Q) = \int_x \log \left( \frac{P(x)}{Q(x)} \right) P(x) dx$$

- Issue: Exploding of KL-divergence for zero values of $P_\theta$
  - Add random noise to $P_\theta$
Approach 2: Learn a function $g_\theta(z)$

- We learn a function $g_\theta(z)$ that transforms $z$ into $P_\theta$
  - $Z$ is a known distribution like Uniform or Gaussian
- We train $g_\theta$ by minimizing the distance between $g_\theta$ and $P_r$
- Any of the distance metrics like KL divergence, JS divergence or Earth Mover (EM) distance can be used.
Revisiting GANs

- GANs are generative models which try to understand underlying distribution to generate more sample.
- GANs typically have 2 networks trained in an adversarial fashion.
  - Generator
  - Discriminator
Revisiting GANs- Generative Network
Revisiting GANs - Generator + Discriminator
Revisiting GANs - training

\[
\max_D \min_G V(G, D) \]

How do we optimize this objective function?

\[
V(G, D) = \mathbb{E}_{p_{data}(x)} \log D(x) + \mathbb{E}_{p_g(x)} \log(1 - D(x))
\]
Revisiting GANs - training

\[
\max_D \min_G V(G, D)
\]

Optimization:
1. Fix generator, and update discriminator
2. Fix discriminator, and update generator
WGANs—Earth Mover Distance

Wasserstein distance: the minimum energy cost of moving and transforming a pile of dirt in the shape of one probability distribution to the shape of other distribution.

P and Q: 4 piles of dirt made up of 10 shovelfuls of dirt present.

- P1 = 3, P2 = 2, P3 = 1, P4 = 4
- Q1 = 1, Q2 = 2, Q3 = 4, Q4 = 3
- W = 5
WGANs-Objective function

- We train GANs using this Wasserstein distance.
- Discriminative is no more a direct critic. It is trained to estimate the Wasserstein distance between real and generated data.

\[ L_D = E_X D(X) - E_Z D(G(Z)) \]

- Lipschitz is clipped to 1 i.e. \( |f(x) - f(y)|/(x-y) \leq 1 \)
  - This bound on discriminator is not good, instead we clip the gradients.
WGAN-Gradient Penalty

- Bound on discriminator is not great and leads to poor discriminator.
- We can add the gradient penalty in the loss function making sure that the lipschitz is almost 1 everywhere.

\[ L_D = \mathbb{E}_X D(X) - \mathbb{E}_Z D(G(Z)) + \lambda \mathbb{E}_{X'} (\| \nabla D(X') \|_2 - 1)^2 \]

- We do not constraint the gradients everywhere.
  - We penalize where there is linear interpolation between real and fake data.
Code Walkthrough

GANs, WGAN-GP
Image translation

- Image-to-image translation involves generating a new synthetic version of a given image.
- Example: Changing a summer landscape --> winter landscape, blonde --> black hair, image --> painting.
- Data for such image translation is very limited or sometimes difficult to generate.
- 2 variants of GANs are used for this specific task.
  - Cycle GAN
  - STAR GAN
Cycle GANs

- Instead of a single Generator-Discriminator we have two Generators and discriminators.
  - One generator takes images from the first domain and outputs images from the second domain.
  - Discriminator models are used to determine how plausible generated images are and update the generator accordingly.
- The overall loss function for the cycle GAN is given below apart from the standard objective we have an added cycle-consistency loss.

\[
\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) \\
+ \mathcal{L}_{GAN}(F, D_X, Y, X) \\
+ \lambda \mathcal{L}_{cyc}(G, F),
\]
Cycle GAN

Cycle-consistency loss: 
\[ L_{cyc}(G, F) = E_{x \sim p_{data}(x)}[\|F(G(x)) - x\|_1] + E_{y \sim p_{data}(y)}[\|G(F(y)) - y\|_1]. \]
Application: Style Transfer

Example of Style Transfer from Famous Painters to Photographs of Landscapes.
Taken from: Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks.
Application: Object Transfiguration
Star GAN (Unified GAN for Multi-Domain I2I translation)

- Star GAN helps us to generate images in target domain given an input and target domain.
  - Image of a man and target domain is gender.
  - Image of a person and target domain is age.
- We train the generator-discriminator in adversarial fashion with an added auxiliary classifier.
- Along with normal adversarial loss this loss is added while training the generator and discriminator.
Star GAN - Generator

- Generator have 3 objectives:
  - Tries to generate realistic images
  - The weights of generator are adjusted so that the generated images are classified as target domain by the discriminator.
  - Construct original image from the fake image given the original label domain label.

Objective function:

$$\mathcal{L}_G = \mathcal{L}_{adv} + \lambda_{cls} \mathcal{L}_{cls}^f + \lambda_{rec} \mathcal{L}_{rec}$$
Star GAN - Discriminator

- Discriminator has 2 objectives:
  - Whether the image is fake or real
  - What is the domain in which the image belongs.
- If the generator is able to generate fool the discriminator then discriminator would predict the target domain and we stop training.

Objective function:

$$\mathcal{L}_D = -\mathcal{L}_{adv} + \lambda_{cls} \mathcal{L}^{r}_{cls}$$
Applications
Thank You!
Thank You!

Slow and steady wins the race is a lie, so pace up: Amit
Code Walkthrough

Cycle GAN and STAR GAN
References

- https://arxiv.org/abs/1703.10593 (Cycle GAN)
- https://arxiv.org/abs/1711.09020 (Star GAN)
- https://machinelearningmastery.com/what-is-cyclegan/
- https://towardsdatascience.com/stargan-image-to-image-translation-44d4230fbb48
- Lecture notes of 11-777
GANs - Code Walkthrough

Yash Belhe, Hao Liang
GAN Loss Function

Some Notation:

$p(x)$ – The distribution over all possible real images that we want to model
GAN Loss Function

Some Notation:

\( p(x) \) – The distribution over all possible real images that we want to model

\( p(z) \) – The distribution over the generator's input e.g. \( U[0,1]^N \) if \( z \in \mathbb{R}^N \)
GAN Loss Function

Some Notation:

\( p(x) \) – The distribution over all possible real images that we want to model

\( p(z) \) – The distribution over the generator's input e.g. \( U[0,1]^N \) if \( z \in \mathbb{R}^N \)

\( G \) – Generator, output is an image \( G(z) \)
GAN Loss Function

Some Notation:

$p(x)$ – The distribution over all possible real images that we want to model

$p(z)$ – The distribution over the generator's input e.g $U[0,1]^N$ if $z \in \mathbb{R}^N$

$G$ – Generator, output is an image $G(z)$

$D$ – Discriminator, output is the probability that the image is real $D(x) \in [0,1]$
GAN Loss Function

Some Notation:

$p(x)$ – The distribution over all possible real images that we want to model

$p(z)$ – The distribution over the generator's input e.g $U[0,1]^N$ if $z \in \mathbb{R}^N$

$G$ – Generator, output is an image $G(z)$

$D$ – Discriminator, output is the probability that the image is real $D(x) \in [0,1]$

Real Image Label - 1

Fake Image Label - 0
GAN Loss Function

Some Notation:

\( p(x) \) – The distribution over all possible real images that we want to model

\( p(z) \) – The distribution over the generator's input e.g. \( U[0,1]^N \) if \( z \in \mathbb{R}^N \)

\( G \) – Generator, output is an image \( G(z) \)

\( D \) – Discriminator, output is the probability that the image is real \( D(x) \in [0,1] \)

Real Image Label - 1

Fake Image Label - 0

\[ \mathcal{L}_{GAN} = \min_G \max_D \mathbb{E}_{x \sim p(x)}[\log(D(x))] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))] \]
GAN Loss Function

\[ \mathcal{L}_{GAN} = \min_G \max_D \mathbb{E}_{x \sim p(x)}[\log(D(x))] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))] \]
GAN Loss Function

\[ \mathcal{L}_{GAN} = \min_G \max_D \mathbb{E}_{x \sim p(x)}[\log(D(x))] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))] \]

We estimate the expectation by an average over samples
GAN Loss Function

\[ \mathcal{L}_{GAN} = \min_G \max_D \mathbb{E}_{x \sim p(x)}[\log(D(x))] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))] \]

We estimate the expectation by an average over samples

Let \( \mathcal{X} \) be a minibatch of samples drawn from \( p(x) \), \(|\mathcal{X}| = N\)

Let \( Z \) be a minibatch of samples drawn from \( p(z) \), \(|Z| = N\)
GAN Loss Function

\[ \mathcal{L}_{GAN} = \min_G \max_D \mathbb{E}_{x \sim p(x)}[\log(D(x))] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z))个交易]) \]

We estimate the expectation by an average over samples

Let \( \mathcal{X} \) be a minibatch of samples drawn from \( p(x) \), \( |\mathcal{X}| = N \)

Let \( Z \) be a minibatch of samples drawn from \( p(z) \), \( |Z| = N \)

\[ \mathcal{L}_{GAN} = \min_G \max_D \frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) + \frac{1}{N} \sum_{z \in Z} \log(1 - D(G(z))) \]
Discriminator Loss

\[ \mathcal{L}_D = - \min_D \frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) + \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \]
Discriminator Loss

\[ \mathcal{L}_D = -\min_D \frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) + \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \]

\[ -\frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) \]
Discriminator Loss

\[
\mathcal{L}_D = - \min_D \frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) + \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z)))
\]

\[
- \frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) \quad \text{cross-entropy loss between the predicted labels } D(x) \text{ and real labels i.e 1}
\]
Discriminator Loss

\[ \mathcal{L}_D = - \min_D \frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) + \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \]

- \frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) \quad \text{cross-entropy loss between the predicted labels } D(x) \text{ and real labels i.e } 1

- \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z)))
Discriminator Loss

\[ \mathcal{L}_D = - \min_D \frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) + \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \]

\[ -\frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) \quad \text{cross-entropy loss between the predicted labels } D(x) \text{ and real labels i.e } 1 \]

\[ -\frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \quad \text{cross-entropy loss between the predicted labels } D(G(z)) \text{ and fake labels i.e } 0 \]
Discriminator Loss

\[ \mathcal{L}_D = -\min_D \frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) + \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \]

\[ -\frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) \quad \text{cross-entropy loss between the predicted labels D(x) and real labels i.e 1} \]

\[ -\frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \quad \text{cross-entropy loss between the predicted labels D(G(z)) and fake labels i.e 0} \]

D_real_loss = bce_loss(D(x), torch.ones(batch_size))
D_fake_loss = bce_loss(D(G(z)), torch.zeros(batch_size))
Generator Loss

$$
\mathcal{L}_{G_{sat}} = - \max_G \frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) + \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z)))
$$
Generator Loss

\[ \mathcal{L}_{G_{sat}} = - \max_G \frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) + \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \]

\[ \mathcal{L}_{G_{sat}} = - \max_G \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \]
Generator Loss

\[ \mathcal{L}_{G_{sat}} = - \max_G \frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) + \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \]

\[ \mathcal{L}_{G_{sat}} = - \max_G \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \]

\[ \mathcal{L}_{G_{sat}} = - \min_G \left[ - \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \right] \]
Generator Loss

\[ \mathcal{L}_{G_{sat}} = -\max_{G} \left\{ \frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) + \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \right\} \]

\[ \mathcal{L}_{G_{sat}} = -\max_{G} \left\{ \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \right\} \]

\[ \mathcal{L}_{G_{sat}} = -\min_{G} \left[ -\frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \right] \]

-ve cross-entropy loss between the predicted labels \(D(G(z))\) and fake labels i.e 0
Generator Loss

\[ \mathcal{L}_{G_{sat}} = - \max_G \frac{1}{N} \sum_{x \in \mathcal{X}} \log(D(x)) + \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \]

\[ \mathcal{L}_{G_{sat}} = - \max_G \frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \]

\[ \mathcal{L}_{G_{sat}} = - \min_G \left[ -\frac{1}{N} \sum_{z \in \mathcal{Z}} \log(1 - D(G(z))) \right] \]

-ve cross-entropy loss between the predicted labels \( D(G(z)) \) and fake labels i.e 0

\[
G\_loss = -\text{bce\_loss}(D(G(z)), \text{torch\_zeros}(\text{batch\_size}))
\]
Generator Loss

\[ \mathcal{L}_{G_{sat}} = - \min_G \left[ -\frac{1}{N} \sum_{z \in Z} \log(1 - D(G(z))) \right] \] -ve cross-entropy loss between the predicted labels \( D(G(z)) \) and fake labels i.e 0

\[ \text{G_loss} = -\text{bce_loss}(D(G(z)), \text{torch.zeros(batch_size)}) \]
Generator Loss

\[ \mathcal{L}_{G_{sat}} = - \min_G \left[ -\frac{1}{N} \sum_{z \in Z} \log(1 - D(G(z))) \right] \]

-ve cross-entropy loss between the predicted labels D(G(z)) and fake labels i.e 0

\[ G_{loss} = -\text{bce_loss}(D(G(z)), \text{torch.zeros}(\text{batch_size})) \]

- D(G(z)) -> 0, when the discriminator is confident that G(z) is fake
- Often happens during the beginning of training
- Empirically this means that the gradients received by G vanish
Generator Loss

\[ \mathcal{L}_{G_{sat}} = -\min_G \left[ -\frac{1}{N} \sum_{z \in Z} \log(1 - D(G(z))) \right] \] -ve cross-entropy loss between the predicted labels \( D(G(z)) \) and fake labels i.e 0

\[ \mathcal{L}_{G_{no\_sat}} = -\min_G \left[ -\frac{1}{N} \sum_{z \in Z} - \log(D(G(z))) \right] \]

\( G_{loss} = -\text{bce\_loss}(D(G(z)), \text{torch\_zeros}(\text{batch\_size})) \)

- \( D(G(z)) \rightarrow 0 \), when the discriminator is confident that \( G(z) \) is fake
- Often happens during the beginning of training
- Empirically this means that the gradients received by \( G \) vanish
Generator Loss

\[ \mathcal{L}_{G_{\text{sat}}} = -\min_G \left[ -\frac{1}{N}\sum_{z \in Z} \log(1 - D(G(z))) \right] \]

-ve cross-entropy loss between the predicted labels \(D(G(z))\) and fake labels i.e 0

\(G\_\text{loss} = -\text{bce\_loss}(D(G(z)), \text{torch\_zeros}(\text{batch\_size}))\)

- \(D(G(z)) \rightarrow 0\), when the discriminator is confident that \(G(z)\) is fake
- Often happens during the beginning of training
- Empirically this means that the gradients received by \(G\) vanish

\[ \mathcal{L}_{G_{\text{no\_sat}}} = -\min_G \left[ -\frac{1}{N}\sum_{z \in Z} - \log(D(G(z))) \right] \]

\[ \mathcal{L}_{G_{\text{no\_sat}}} = -\min_G \frac{1}{N}\sum_{z \in Z} \log(D(G(z))) \]
Generator Loss

\[ \mathcal{L}_{G_{\text{sat}}} = - \min_G \left[ -\frac{1}{N} \sum_{z \in Z} \log(1 - D(G(z))) \right] \]

-ve cross-entropy loss between the predicted labels \( D(G(z)) \) and fake labels i.e 0

\[ \text{G_loss} = -\text{bce_loss}(D(G(z)), \text{torch.zeros(batch_size)}) \]

- \( D(G(z)) \) -> 0, when the discriminator is confident that \( G(z) \) is fake
- Often happens during the beginning of training
- Empirically this means that the gradients received by \( G \) vanish

\[ \mathcal{L}_{G_{\text{no sat}}} = - \min_G \left[ -\frac{1}{N} \sum_{z \in Z} - \log(D(G(z))) \right] \]

cross-entropy loss between the predicted labels \( D(G(z)) \) and real labels i.e 1
Generator Loss

\[ \mathcal{L}_{G_{\text{sat}}} = - \min_G \left[ - \frac{1}{N} \sum_{z \in Z} \log(1 - D(G(z))) \right] \text{-ve cross-entropy loss between the predicted labels D(G(z)) and fake labels i.e 0} \]

\[ \text{G_loss} = -\text{bce_loss}(D(G(z)), \text{torch.zeros(batch_size)}) \]

- D(G(z)) -> 0, when the discriminator is confident that G(z) is fake
- Often happens during the beginning of training
- Empirically this means that the gradients received by G vanish

\[ \mathcal{L}_{G_{\text{no_sat}}} = - \min_G \left[ - \frac{1}{N} \sum_{z \in Z} \log(D(G(z))) \right] \text{cross-entropy loss between the predicted labels D(G(z)) and real labels i.e 1} \]

\[ \text{G_loss} = \text{bce_loss}(D(G(z)), \text{torch.ones(batch_size)}) \]
Rough Code Implementation

(full code link)

```python
G = generator()
D = discriminator()

bce_loss = nn.BCELoss()
D_optimizer = optim.Adam(D.parameters())
G_optimizer = optim.Adam(G.parameters())

z = get_noise()
x = get_real()

D_real_loss = bce_loss(D(x), torch.ones(batch_size))
D_fake_loss = bce_loss(D(G(z)), torch.zeros(batch_size))

D_loss = D_real_loss + D_fake_loss
D_loss.backward()
D_optimizer.step()

G_loss = bce_loss(D(G(z)), torch.ones(batch_size))
G_loss.backward()
G_optimizer.step()
```
W-GAN

\[ \mathcal{L}_{W-GAN} = \min_G \max_D \mathbb{E}_{x \sim p(x)}[D(x)] - \mathbb{E}_{z \sim p(z)}[D(G(z))] \]

Where \( \|D\|_L \leq K \), i.e D is K-Lipschitz Continuous
W-GAN

\[ \mathcal{L}_{WGAN} = \min_G \max_D \mathbb{E}_{x \sim p(x)}[D(x)] - \mathbb{E}_{z \sim p(z)}[D(G(z))] \]

Where \( \|D\|_L \leq K \), i.e. \( D \) is \( K \)-Lipschitz Continuous

- Measures the Wasserstein/ Earth Mover Distance between two distributions
How To Enforce K-Lipschitz Continuity for the Discriminator?

- Heuristic: Clip each weight $w$ of the discriminator s.t $|w| < c$
How To Enforce K-Lipschitz Continuity for the Discriminator?

• Heuristic: Clip each weight $w$ of the discriminator s.t $|w| < c$

• Is this a good way of maintaining Lipschitz Continuity - No
How To Enforce K-Lipschitz Continuity for the Discriminator?

- Heuristic: Clip each weight $w$ of the discriminator s.t $|w| < c$

- Is this a good way of maintaining Lipschitz Continuity - No

- Does it work? Somewhat
W-GAN Discriminator Loss

\[ \mathcal{L}_D = \max_D \mathbb{E}_{x \sim p_r}[D(x)] - \mathbb{E}_{z \sim p_z}[D(G(z))] \]
W-GAN Discriminator Loss

\[ \mathcal{L}_D = \max_D \mathbb{E}_{x \sim p_r}[D(x)] - \mathbb{E}_{z \sim p_r(z)}[D(G(z))] \]

\[ \mathcal{L}_D = \max_D \frac{1}{N} \sum_{x \in \mathcal{X}} D(x) - \frac{1}{N} \sum_{z \in \mathcal{Z}} D(G(z)) \]
W-GAN Discriminator Loss

\[ \mathcal{L}_D = \max_D \mathbb{E}_{x \sim p_r}[D(x)] - \mathbb{E}_{z \sim p_r(z)}[D(G(z))] \]

\[ \mathcal{L}_D = \max_D \frac{1}{N} \sum_{x \in \mathcal{X}} D(x) - \frac{1}{N} \sum_{z \in \mathcal{Z}} D(G(z)) \]

\[ \mathcal{L}_D = \min_D \left[ -\frac{1}{N} \sum_{x \in \mathcal{X}} D(x) + \frac{1}{N} \sum_{z \in \mathcal{Z}} D(G(z)) \right] \]
W-GAN Discriminator Loss

\[ \mathcal{L}_D = \max_D \mathbb{E}_{x \sim p_r}[D(x)] - \mathbb{E}_{z \sim p(z)}[D(G(z))] \]

\[ \mathcal{L}_D = \max_D \frac{1}{N} \sum_{x \in \mathcal{X}} D(x) - \frac{1}{N} \sum_{z \in \mathcal{Z}} D(G(z)) \]

\[ \mathcal{L}_D = \min_D \left[ -\frac{1}{N} \sum_{x \in \mathcal{X}} D(x) + \frac{1}{N} \sum_{z \in \mathcal{Z}} D(G(z)) \right] \]

\[ D\_loss = -D(x).\text{mean}() + D(G(z)).\text{mean}() \]
W-GAN Discriminator Loss

\[ \mathcal{L}_D = \max_D \mathbb{E}_{x \sim p_r}[D(x)] - \mathbb{E}_{z \sim p_r(z)}[D(G(z))] \]

\[ \mathcal{L}_D = \max_D \frac{1}{N} \sum_{x \in \mathcal{X}} D(x) - \frac{1}{N} \sum_{z \in \mathcal{Z}} D(G(z)) \]

\[ \mathcal{L}_D = \min_D \left[ -\frac{1}{N} \sum_{x \in \mathcal{X}} D(x) + \frac{1}{N} \sum_{z \in \mathcal{Z}} D(G(z)) \right] \]

\[ \text{D_loss} = -D(x).\text{mean}() + D(G(z)).\text{mean}() \]

For Lipschitz Continuity:
W-GAN Discriminator Loss

\[
\mathcal{L}_D = \max_D \mathbb{E}_{x \sim p_r}[D(x)] - \mathbb{E}_{z \sim p_r(z)}[D(G(z))]
\]

\[
\mathcal{L}_D = \max_D \frac{1}{N} \sum_{x \in \mathcal{X}} D(x) - \frac{1}{N} \sum_{z \in \mathcal{Z}} D(G(z))
\]

\[
\mathcal{L}_D = \min_D \left[ -\frac{1}{N} \sum_{x \in \mathcal{X}} D(x) + \frac{1}{N} \sum_{z \in \mathcal{Z}} D(G(z)) \right]
\]

\[
D\_loss = -D(x).\text{mean()} + D(G(z)).\text{mean()}
\]

For Lipschitz Continuity:

```python
for p in D.parameters():
    p.data.clamp_(-c, c)
```
W-GAN Generator Loss

$$\mathcal{L}_D = \min_G \mathbb{E}_{x \sim p_r}[D(x)] - \mathbb{E}_{z \sim p_r(z)}[D(G(z))]$$
W-GAN Generator Loss

\[ \mathcal{L}_D = \min_G \mathbb{E}_{x \sim p_r}[D(x)] - \mathbb{E}_{z \sim p_r(z)}[D(G(z))] \]

\[ \mathcal{L}_D = \min_G - \frac{1}{N} \sum_{z \in Z} D(G(z)) \]
W-GAN Generator Loss

\[ \mathcal{L}_D = \min_G \mathbb{E}_{x \sim p_r}[D(x)] - \mathbb{E}_{z \sim p_z}[D(G(z))] \]

\[ \mathcal{L}_D = \min_G - \frac{1}{N} \sum_{z \in Z} D(G(z)) \]

\[ G_{\text{loss}} = -D(G(z)).\text{mean}() \]
G = generator()
D = discriminator()

c = 0.01  #Some small number

D_optimizer = optim.Adam(D.parameters())
G_optimizer = optim.Adam(G.parameters())

z = get_noise()
x = get_real()

D_loss = -D(x).mean() + D(G(z)).mean()
D_loss.backward()
D_optimizer.step()

for p in D.parameters():
    p.data.clamp_(-c, c)

G_loss = -D(G(z)).mean()
G_loss.backward()
G_optimizer.step()