Machine Learning

Lecture 11: Generative Adversarial Networks (II)

Nevin L. Zhang
lzhang@cse.ust.hk

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

This set of notes is based on internet resources and references listed at the end.

Outline

Outline

Outline

Outline

GAN solves a minimax game

- The objective function $\frac{1}{m}\sum_{i=1}^{m}\log D(\mathbf{x}^{(i)}) + \frac{1}{m}\sum_{i=1}^{m}\log(1 - D(g(\mathbf{z}^{(i)})))$ used in the GAN algorithm is an approximation of

$$\begin{aligned} V(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d) &= \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}\log D(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\log(1 - D(g(\mathbf{z}))) \\ &= \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}\log D(\mathbf{x}) + \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}\log(1 - D(\mathbf{x})) \end{aligned}$$

  We will use the latter in the analysis. Sometimes, we also write it as $V(G, D)$

- The discriminator wants to maximize V and the generator want to minimize V. So, we have a minmax game

$$\boldsymbol{\theta}_g^* = \arg\min_{\boldsymbol{\theta}_g}\max_{\boldsymbol{\theta}_d} V(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d)$$

Optimal Discriminator for Fixed Generator

- Assume that both G and D has sufficient capacity so that they can represent any function from $\mathbf{z}$ to $\mathbf{x}$ and any function from $\mathbf{x}$ to $[0, 1]$.
- **Theorem** (Goodfellow et al 2014): For fixed G, the optimal discriminator is

$$D^*(\mathbf{x}) = \frac{p_r(\mathbf{x})}{p_r(\mathbf{x}) + p_g(\mathbf{x})}$$

- **Proof**:

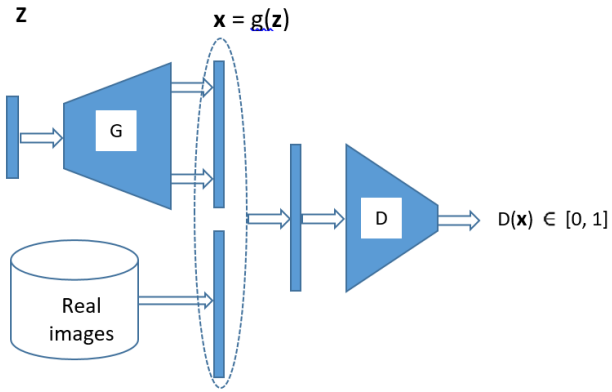
$$\begin{aligned} V(G, D) &= \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})} \log(1 - D(\mathbf{x})) \\ &= \int [p_r(\mathbf{x}) \log D(\mathbf{x}) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x}))]\, d\mathbf{x} \\ &\leq \int [p_r(\mathbf{x}) \log D^*(\mathbf{x}) + p_g(\mathbf{x}) \log(1 - D^*(\mathbf{x}))]\, d\mathbf{x} \end{aligned}$$

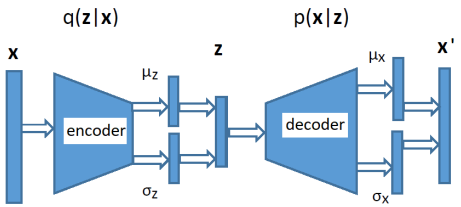
The last step follows by applying Gibbs' inequality to the integrand.

GAN and Jensen-Shannon Divergence

$$
\begin{aligned}
V(G) &= \max_D V(G,D) \\
&= \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})} \log \frac{p_r(\mathbf{x})}{p_r(\mathbf{x})+p_g(\mathbf{x})} + \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})} \log(1 - \frac{p_r(\mathbf{x})}{p_r(\mathbf{x})+p_g(\mathbf{x})}) \\
&= \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})} \log \frac{p_r(\mathbf{x})}{(p_r(\mathbf{x})+p_g(\mathbf{x}))/2} + \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})} \log(\frac{p_g(\mathbf{x})}{(p_r(\mathbf{x})+p_g(\mathbf{x}))/2}) - \log 4 \\
&= -\log 4 + 2JS(p_r||p_g)
\end{aligned}
$$

GAN and Jensen-Shannon Divergence





The above analysis leads to the following view on GAN:

- The objective of GAN is to learn the parameters of the generator so as to minimize the JS divergence $JS(p_r||p_g)$ between the real data distribution p_r and the generator distribution p_g.
- The introduction of a discriminator is to approximate the JS divergence.

Recall that the objective of VAE is to learn parameters of the decoder by minimizing the KL divergence $KL(p_r||p_g)$ between p_r and p_g (called the decoder distribution in VAE context). The introduction of an encoder $q(\mathbf{z}|\mathbf{x})$ is to provide an approximation (an upper bound) for the KL divergence

Why GAN generates more realistic images than VAE?

- VAE minimize the KL divergence

$$KL(p_r||p_g) = \int p_r(\mathbf{x}) \log \frac{p_r(\mathbf{x})}{p_g(\mathbf{x})} d\mathbf{x}$$

  - KL receives large contributions from areas of data space where $p_g(\mathbf{x})$ is small and $p_r(\mathbf{x})$ is large: High cost for not covering parts of data. Minimizing KL avoids such areas.
  - KL receives small contributions from areas of data space where $p_r(\mathbf{x})$ is small and $p_g(\mathbf{x})$ is large: Low cost for generating fake looking images. Minimizing KL does not avoid such areas. This is why VAE generates blurry images.

- If $KL(p_g||p_r)$ is used instead, the story will be reversed. The generator distribution might not covering parts of the data points, and hence will not generate images similar to those in that areas. This is called mode dropping

Why GAN generates more realistic images than VAE?

- GAN minimize the JS divergence

$$JS(p_r||p_g) = \frac{1}{2}KL(p_r||p_a) + \frac{1}{2}KL(p_g||p_a)$$

  where $p_a = (p_r + p_g)/2$.

- $JS(p_r||p_g)$ is a middle ground between $KL(p_r||p_g)$ and $KL(p_q||p_r)$. Hence, it gives high cost for generating fake looking images.
- This is one reason why GAN generates more realistically looking images than VAE.

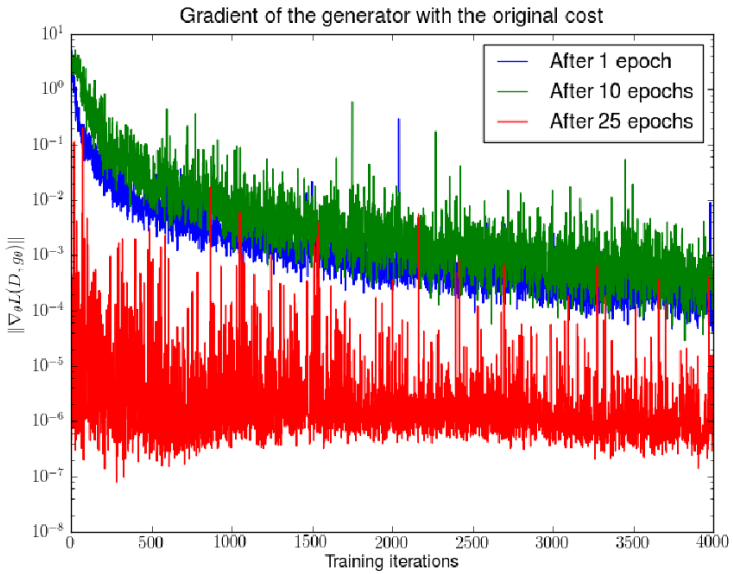
Two Cost Functions for Generator

- The original cost function for the generator is: $\mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})} \log(1 - D(\mathbf{x}))$, i.e, expected log-probability that the discriminator being correct.
- In practice, an alternative cost function is used: $-\mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}[\log D(\mathbf{x})]$, i.e, negation of expected log-probability that the discriminator being mistaken.
- This is why:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_g} \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})} \log(1 - D(\mathbf{x})) &= \nabla_{\boldsymbol{\theta}_g} \int p_g(\mathbf{x}) \log(1 - D(\mathbf{x})) d\mathbf{x} \\ &= \int \log(1 - D(\mathbf{x})) \nabla_{\boldsymbol{\theta}_g} p_g(\mathbf{x}) d\mathbf{x}\end{aligned}$$

  - At the beginning, the generator is poor. The discriminator can easily tell fakes images from real ones, i.e. $D(\mathbf{x}) = 0$ for all $\mathbf{x} \sim p_g(\mathbf{x})$.
  - Hence, the gradient for generator is 0. The generator does not get information on how to improve its parameters at a time it needs such information the most.

Gradient of Original Generator Cost Function



- How the gradient of the original cost function change as we train the discriminator to optimum.
- Arjovsky et al. ICLR 2017: " First, we trained a DCGAN for 1, 10 and 25 epochs. Then, with the generator fixed we train a discriminator from scratch and measure the gradients with the original cost function. We see the gradient norms decay quickly, in the best case 5 orders of magnitude after 4000 discriminator iterations. Note the logarithmic scale "

Two Cost Functions for Generator

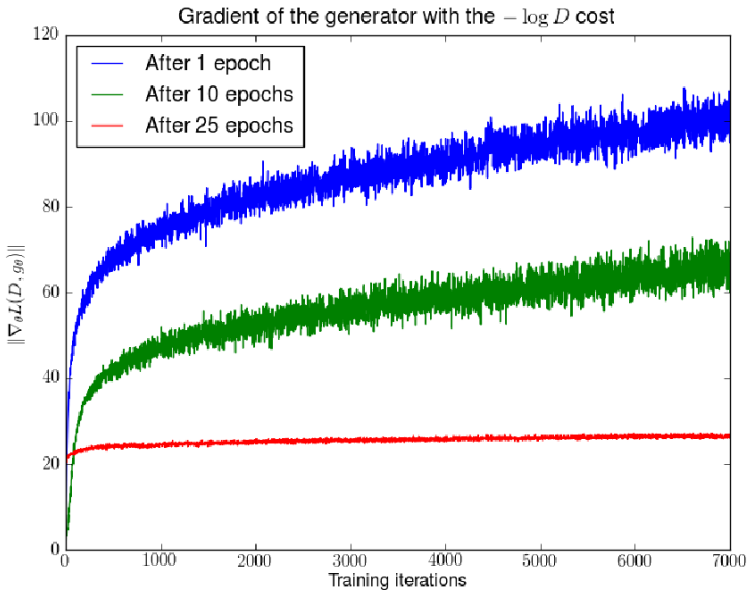
- The vanishing gradient problem is avoided if we use the alternative cost function $\mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}[-\log D(\mathbf{x})]$ because

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_g}\mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}[-\log D(\mathbf{x})] &= \nabla_{\boldsymbol{\theta}_g}\int p_g(\mathbf{x})[-\log D(\mathbf{x})]d\mathbf{x}\\ &= \int[-\log D(\mathbf{x})]\nabla_{\boldsymbol{\theta}_g}p_g(\mathbf{x})d\mathbf{x}\end{aligned}$$

  The gradient for the generator is no longer zero, because $D(\mathbf{x})$ is not close to 1 for $\mathbf{x}\sim p_g(\mathbf{x})$.

- However, when the discriminator become good ($D(\mathbf{x})\approx 0$ for $\mathbf{x}\sim p_g(\mathbf{x})$), the gradient become very large (note $\log 0 = -\infty$), making the training unstable.

Gradient of the Alternative Generator Cost Function



- How the gradient of the alternative cost function change as we train the discriminator to optimum.
- With a poor generator (the one after only one epoch of training), the gradient of the alternative cost function goes to infinite.
- The general trend is the same with a better generator (the ones after 10, or 25 epochs of training), except now the gradient increase at a slower pace.

Difficulties with GAN

- The original generator cost function leads to slow training, and the alternative cost function leads to unstable training.
- In practice, those difficulties are dealt with by not training the discriminator to optimum, not even close to optimum.
- Next, we will analyze the reason for the difficulties, and introduce a better method.

Core Reason for the Difficulties with GAN

- Let $\mathbf{x}$ be the vector that represents a real image. Let d_r be its dimensionality.
- If the image is 256×256, then $d_r = 65,536$.
- New consider randomly generating a vector of $65,536$ dimensions. What is the probability that it corresponds to a real image?
- The probability is extremely small.
- Hence, the support $supp(p_r) = \{\mathbf{x}|p_r(\mathbf{x}) > 0\}$ of the real data distribution is an extremely small subset of $\mathbb{R}^{d_r}$.

A Side Note

- Some claims that $supp(p_r)$ is a manifold of dimensionality lower than d_r.
- This is not true.
  - Let $\mathbf{x} \in supp(p_r)$, i.e., it corresponds to a real image.
  - Then any member of the following set should also be a realistically looking image:

$$\{\mathbf{x} + 10^{-10}\boldsymbol{\epsilon} | \boldsymbol{\epsilon} \in [-1, 1]^{d_r}\}$$

  - Such tiny "bubbles" are of dimensionality d_r, and they are subsets of $supp(p_r)$.
  - So, $supp(p_r)$ is of dimensionality d_r.

Core Reason for the Difficulties with GAN

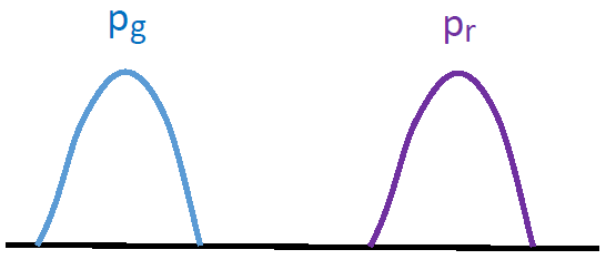
- Now, let d_z be the dimensionality of $\mathbf{z}$. It is much smaller than d_r.
- The support of the generator distribution

$$supp(p_g) = \{\mathbf{x}|p_g(\mathbf{x}) > 0\} = \{g(\mathbf{z})|\mathbf{z} \in [0,1]^{d_z}\}$$

  is of dimensionality d_z.
- It is a manifold of dimensionality (d_z) lower than d_r, hence measure-zero subset of $\mathbb{R}^{d_r}$.

Core Reason for the Difficulties with GAN

- Because both $supp(p_r)$ and $supp(p_g)$ are tiny subsets of $\mathbb{R}^{d_r}$, the probability that they intersect is negligible.
- This is like randomly drop a coin ($supp(p_r)$) and a needle ($supp(p_g)$) in a soccer field, and ask about the probability that the coin rests on top of the needle or vice versa.
- So, it is almost always the case that $supp(p_r) \cap supp(p_g) = \emptyset$



Core Reason for the Difficulties with GAN

- In this case, $p_r(\mathbf{x}) = 0$ for all $\mathbf{x} \in supp(p_g)$, and $p_g(\mathbf{x}) = 0$ for all $\mathbf{x} \in supp(p_r)$.
- Recall that $p_a = (p_r + p_g)/2$. We have

$$\begin{aligned} KL(p_r||p_a) &= \int p_r(\mathbf{x}) \log \frac{p_r(\mathbf{x})}{p_a(\mathbf{x})} d\mathbf{x} = \int_{supp(p_r)} p_r(\mathbf{x}) \log \frac{p_r(\mathbf{x})}{p_a(\mathbf{x})} d\mathbf{x} \\ &= \int_{supp(p_r)} p_r(\mathbf{x}) \log \frac{p_r(\mathbf{x})}{p_r(\mathbf{x})/2} d\mathbf{x} = \log 2 \\ KL(p_g||p_a) &= \log 2 \text{ by symmetry} \end{aligned}$$

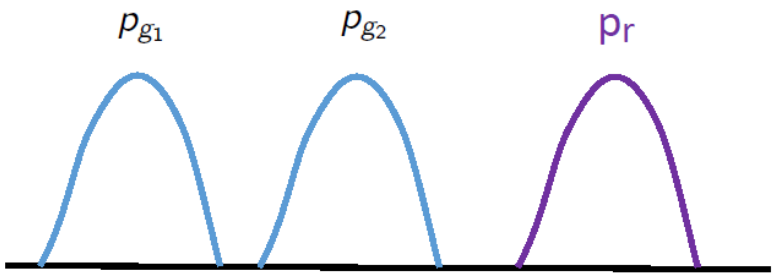
- Hence

$$\begin{aligned} JS(p_r||p_g) &= \frac{1}{2} KL(p_r||p_a) + \frac{1}{2} KL(p_g||p_a) = \log 2 \\ \nabla_{\boldsymbol{\theta}_g} JS &= 0.^1 \end{aligned}$$

[1] This is another explanation for the fact that if we train the discriminator to optimum, the generator would have 0 gradient.

Core Reason for the Difficulties with GAN

- So, $JS(p_r||p_g)$ is almost always $\log 2$.
- As such, JS is not a desirable cost function to train the generator.
- In the following figure, p_{g_2} is closer to p_r than p_{g_1}. Intuitively, we would think g_2 is a better generator. However, they are the same according to JS.

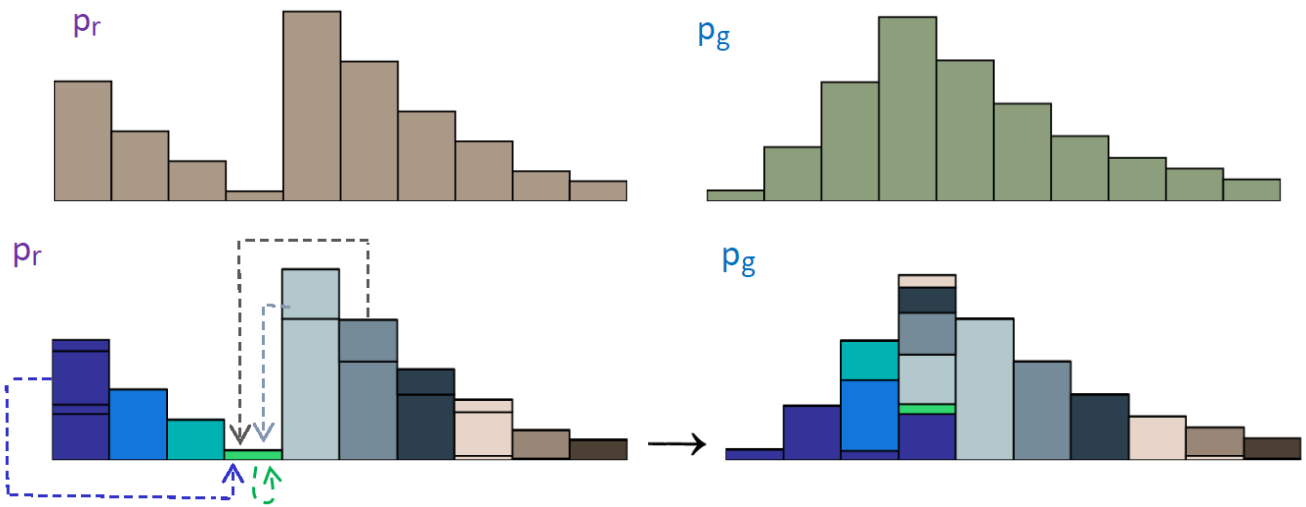


- So, we need a better way to measure the divergence between distributions.

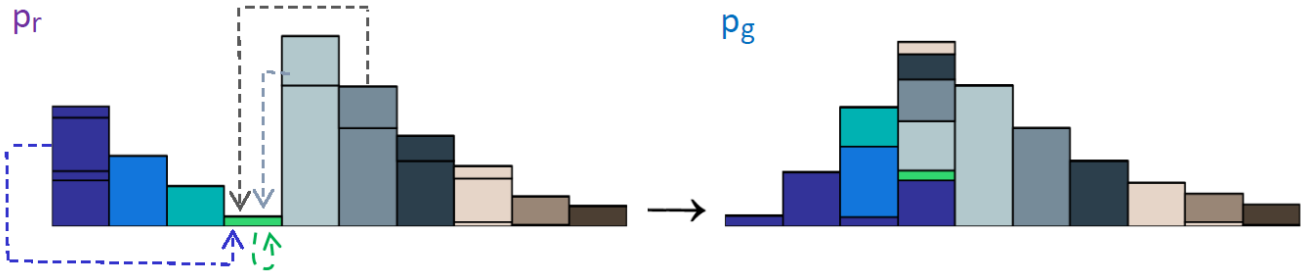
Outline

Earth Mover's Distance or Wasserstein Distance

- The **Earth Mover's Distance (EMD)** or **Wasserstein Distance** is a measure of the distance between two distributions.
- In the discrete case, a distribution can be depicted as a histogram, which can be viewed as heap of dirt.
- The EMD is the minimal total amount work to transform one distribution to another.

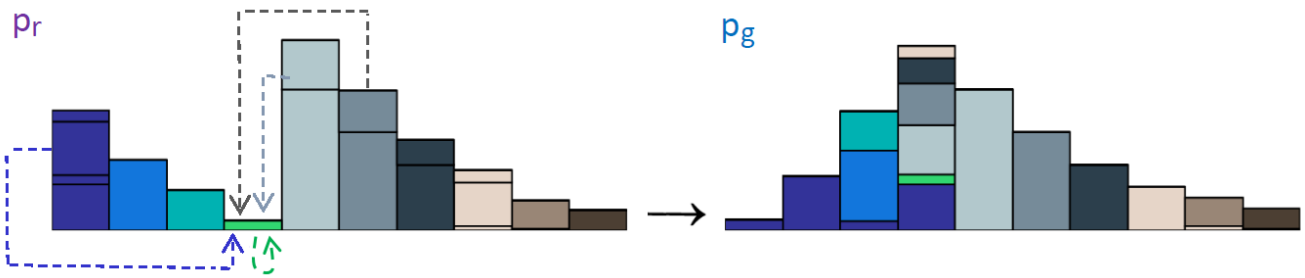


Earth Mover's Distance or Wasserstein Distance



- A transport plan can be described using a function $\gamma(\mathbf{x}, \mathbf{y})$, which stands for the amount of dirt to transport from location $\mathbf{x}$ to location $\mathbf{y}$. It satisfies three conditions:
  - $0 \leq \gamma(\mathbf{x}, \mathbf{y})$.
  - $\sum_{\mathbf{y}} \gamma(\mathbf{x}, \mathbf{y}) = p_r(\mathbf{x})$: All the dirt at location $\mathbf{x}$ is moved to some of the locations, possibly the location $\mathbf{x}$ itself.
  - $\sum_{\mathbf{x}} \gamma(\mathbf{x}, \mathbf{y}) = p_g(\mathbf{y})$: All the dirt at location $\mathbf{y}$ after the move is from some of the locations, possibly thelocation $\mathbf{y}$ itself.
- γ is a distribution over pairs of locations: $\sum_{\mathbf{x},\mathbf{y}} \gamma(\mathbf{x}, \mathbf{y}) = 1$.

Earth Mover's Distance or Wasserstein Distance



- The total cost of a particular transport plan γ is:

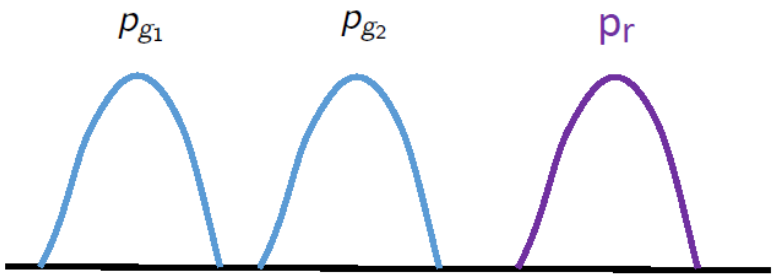
$$\sum_{\mathbf{x},\mathbf{y}} \gamma(\mathbf{x},\mathbf{y})||\mathbf{x}-\mathbf{y}|| = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\gamma}||\mathbf{x}-\mathbf{y}||$$

- The earth mover's distance (EMD) between the two distribution is the minimal cost:

$$EMD(p_r, p_g) = \min_{\gamma} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\gamma}||\mathbf{x}-\mathbf{y}||$$

- The definition also applies to the continuous case, except in this case, summation needs to be replaced by integration.

Earth Mover's Distance or Wasserstein Distance



- It is clear that, in this example,

$$EMD(p_{g_2}, p_r) \leq EMD(p_{g_1}, p_r)$$

- Recall that

$$JS(p_{g_2}||p_r) = JS(p_{g_1}||p_r)$$

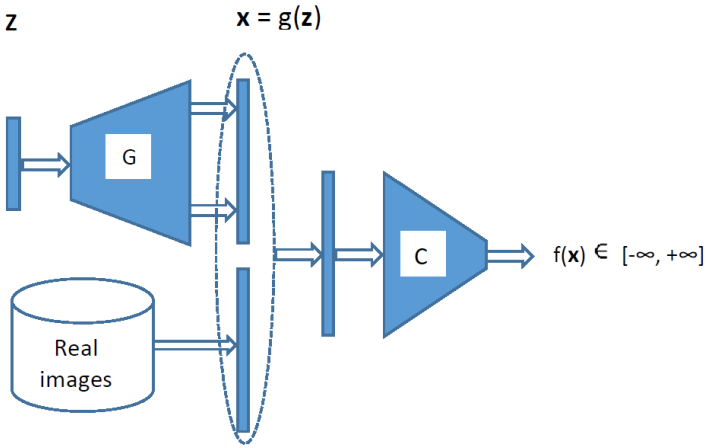
- Obviously, EMD is a better measure of distance between distributions than JS.

Wasserstein GAN (WGAN)

- We would like to learn the generator by minimizing the EM distance

$$\arg\min_{\theta} EMD(p_r, p_\theta)$$

- Computing the EM distance is a difficult optimization problem itself.
- We will use a neural network called a **critic** to provide an approximation of the EM distance.
- This leads to **Wasserstein GAN (WGAN)**. (Arjovsky et al 2017). (When talking about WGAN, we adopt some of the notations from Arjovsky).



Lipschitz Functions

- Let K be a positive real number. A real-valued function $f(\mathbf{x})$ is K-**Lipschitz** if
$$|f(\mathbf{x}) - f(\mathbf{y})| \leq K||\mathbf{x} - \mathbf{y}|| \text{ for all } \mathbf{x} \text{ and } \mathbf{y}$$
- If f is differentiable, then it is K-Lipschitz if and only if its gradient norm is bounded by K everywhere:
$$||\nabla_{\mathbf{x}} f(\mathbf{x})|| \leq K \text{ for all } \mathbf{x}$$
- Intuitively, a K-Lipschitz function is one that does not change too fast with its arguments.

Kantorovich-Rubinstein Duality

- Let $\mathcal{F}_{K-L}$ be the set of all K-Lipschitz functions.
- It has been shown (Villani 2009) that

$$EMD(p_r, p_\theta) = \max_{f \in \mathcal{F}_{1-L}} (\mathbb{E}_{\mathbf{x} \sim p_r}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_\theta}[f(\mathbf{x})])$$

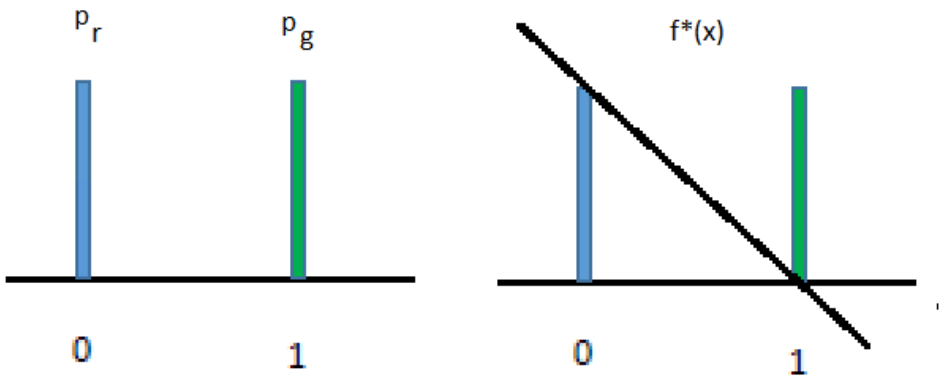
  This is called **Kantorovich-Rubinstein Duality**.

- Furthermore, define

$$EMD_K(p_r, p_\theta) = \max_{f \in \mathcal{F}_{K-L}} (\mathbb{E}_{\mathbf{x} \sim p_r}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_\theta}[f(\mathbf{x})])$$

  Then, $$EMD_K(p_r, p_\theta) = K \cdot EMD(p_r, p_\theta)$$

Kantorovich-Rubinstein Duality:Example



- In this example, $EMD(p_r, p_\theta) = 1$.
- Let us consider only linear functions $f(x) = ax + b$

$$\begin{aligned} EMD(p_r, p_\theta) &= \max_{f \in \mathcal{F}_{1-L}} (\mathbb{E}_{\mathbf{x} \sim p_r}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_\theta}[f(\mathbf{x})]) \\ &\geq \max_{f = ax+b : |a| \leq 1} (f(0) - f(1) \\ &= \max_{|a| \leq 1} (-a) = 1. \end{aligned}$$

The maximum is attained at $a^* = -1$, or $f^*(x) = -x$. (This does not prove the KR duality, but it gives us an idea why it is true.)

Wasserstein GAN (WGAN)

- Suppose the maximum is attained for some f^*, i.e.,

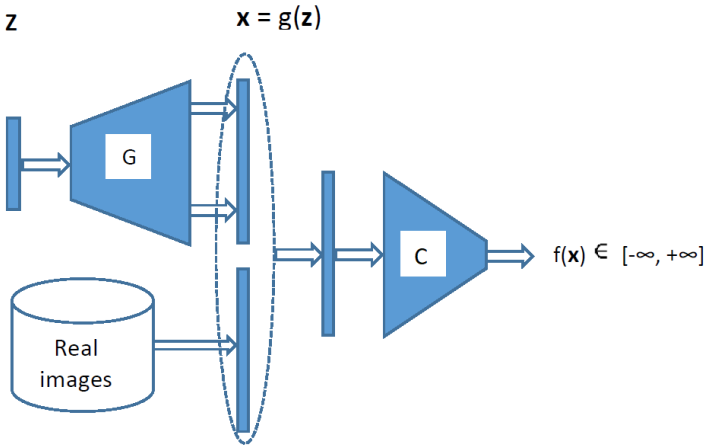
$$\max_{f \in \mathcal{F}_{K-L}} (\mathbb{E}_{\mathbf{x} \sim p_r}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_{\boldsymbol{\theta}}}[f(\mathbf{x})]) = \mathbb{E}_{\mathbf{x} \sim p_r}[f^*(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_{\boldsymbol{\theta}}}[f^*(\mathbf{x})]$$

- Then we can get gradient for the generator as follows:

$$\nabla_{\boldsymbol{\theta}} EMD(p_r, p_{\boldsymbol{\theta}}) = -\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim p_{\boldsymbol{\theta}}}[f^*(\mathbf{x})]$$

- But, how to find f^*?

Wasserstein GAN (WGAN)



- To get an approximation of f^*, we create a neural network with weights $\mathbf{w}$ that maps $\mathbf{x}$ to $\mathbb{R}$, and restrict each weight to be from an interval $[-c, c]$.
- The network is called a **critic**. It is trained to give high values to samples from P_r and low values to samples from p_g, so as to maximize the following **critic objective function**:

$$L_c = \mathbb{E}_{\mathbf{x}\sim p_r}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim p_\theta}[f(\mathbf{x})]$$

- The restriction on the weights is to ensure the function f represented by the critic is K-Lipschitz. The restriction is called **weight clipping**.

Wasserstein GAN (WGAN)

- After learning a good critic f^*, we learn the generator to produce samples that the critic considers good, i.e.,

$$\arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim p_{\boldsymbol{\theta}}}[f^*(\mathbf{x})] = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim p_{\boldsymbol{\theta}}}[-f^*(\mathbf{x})]$$

- We do this using gradient descent, where the gradient is:

$$\nabla_{\boldsymbol{\theta}} EMD(p_r, p_{\boldsymbol{\theta}}) = -\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim p_{\boldsymbol{\theta}}}[f^*(\mathbf{x})]$$

The WGAN Algorithm

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

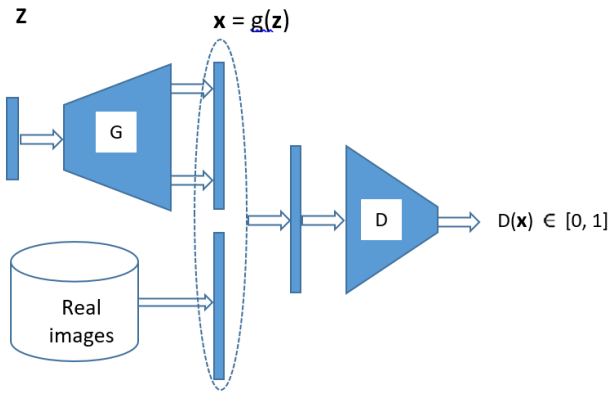
Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

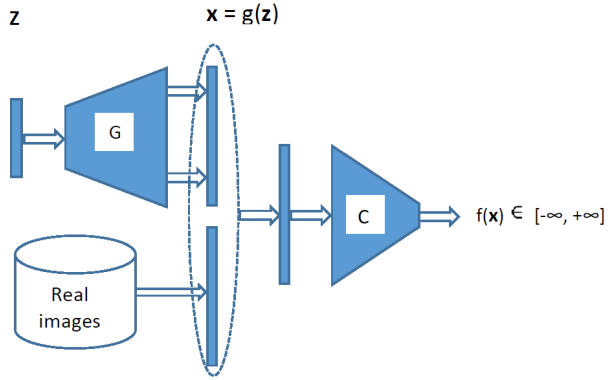
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

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

Adam found not stable. So, RMSProp is used.

Comparison with GAN





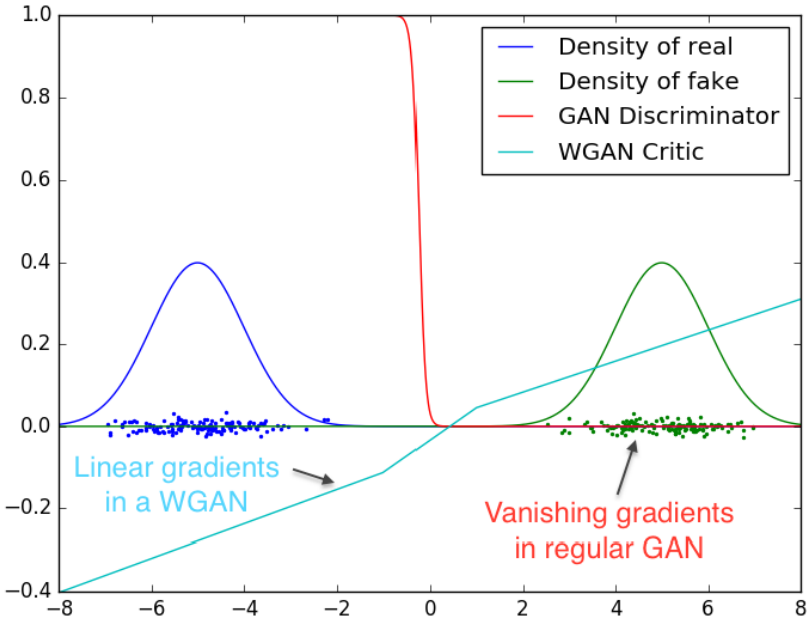
- In GAN, the discriminator is trained by maximizing the following

$$\mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})} \log(1 - D(\mathbf{x}))$$

  We want it to give high probabilities to real images and low probabilities to fake images so that we can classify the images correctly.

- In WGAN, we cant the critic to give high values to real images and low values to fake images. The task is evaluation, not classification. This is why it is called a critic.

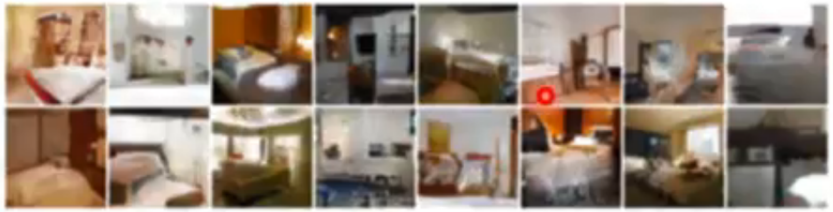
Comparison with GAN



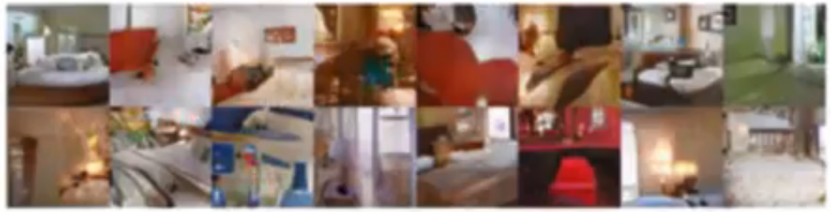
- The WGAN critic curve tells us how good the fake green points are, and how to improve the generator so as to generate points closer to the real data points.
- The GAN discriminator correctly classify green points as fake, but it does not tell us how to improve the generator so as to generate points closer to the real data points.

Empirical Comparison of GAN and WGAN

CNN generator:



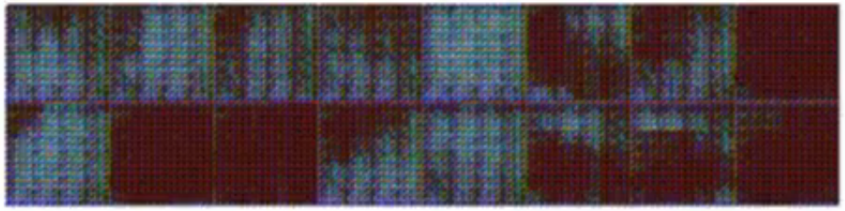
W-GAN



GAN

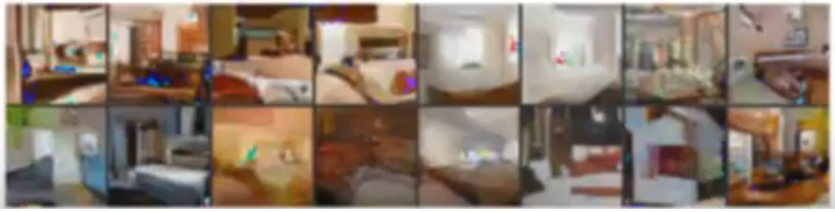
CNN generator (no batch normalization, bad structure):

W-GAN

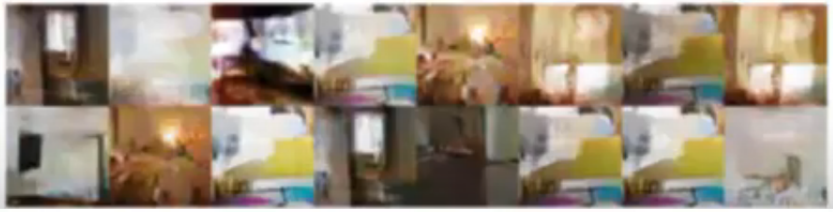


GAN

MLP generator:



W-GAN



GAN

Model Collapse

- GAN is known to suffer from the **model collapse** problem: It tends to produce samples with low variety. (See bottom right of previous page)
- WGAN does not have the problem.

WGAN with Gradient Penalty

- Weight clipping is a simplistic way to enforce the Lipschitz constraint. There is room for improvement
- One improvement is proposed by Gulrajani et al (2017). It is called **gradient penalty**.

- The notations used by Gulrajani et al are somewhat different from what we used so far
  - Function represented by the critic network is denoted as $D(\mathbf{x})$ instead of $f(\mathbf{x})$.
  - The critic is trained to minimize

$$L_c = \mathbb{E}_{\mathbf{x}\sim p_g}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim p_r}[D(\mathbf{x})]$$

    where $D(\mathbf{x})$ is a 1-Lipschitz function. Early, we talked about training the critic to maximize a objective function that is the negation of the L_c function defined here.

WGAN with Gradient Penalty

- Recall that a function is 1-Lipschitz if its gradient norm is upper bounded by 1 everywhere.
- On the other hand, we want the gradient norm of the critic to be as large as possible, so that the generator distribution p_g can approach the real data distribution p_r as fast as possible.
- So, Gulrajani et al (2017) propose the following new critic objective function

$$L_c = \mathbb{E}_{\mathbf{x}\sim p_g}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim p_r}[D(\mathbf{x})] + \lambda \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[(||\nabla_{\mathbf{x}} D(\mathbf{x})||_2 - 1)^2]$$

- The blue term is called the **gradient penalty** term.
  - It encourages the critic gradient at some sampled points to be one.
  - Such points are obtained by: Sample a point $\mathbf{x}$ from p_r, sample a point $\tilde{\mathbf{x}}$ from p_g, randomly pick a point on the line between $\mathbf{x}$ and $\tilde{\mathbf{x}}$

WGAN with Gradient Penalty

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

WGAN-GP is used in Progressive GAN, StyleGAN, and StarGAN.

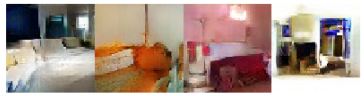
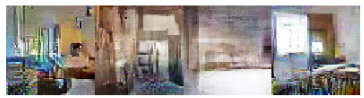
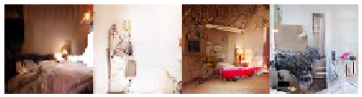
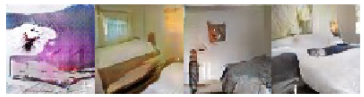
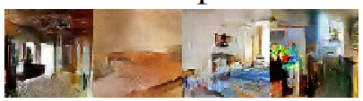
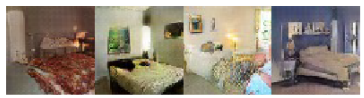
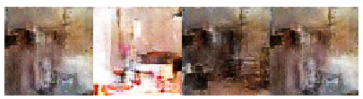
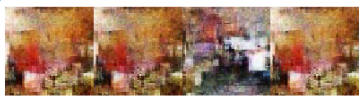
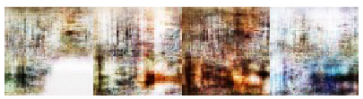
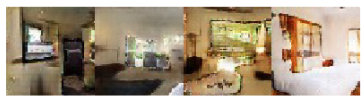
DCGAN	LSGAN	WGAN (clipping)	WGAN-GP (ours)
Baseline (G: DCGAN, D: DCGAN)			
			
G: No BN and a constant number of filters, D: DCGAN			
			
G: 4-layer 512-dim ReLU MLP, D: DCGAN			
			
No normalization in either G or D			
			
Gated multiplicative nonlinearities everywhere in G and D			
			
tanh nonlinearities everywhere in G and D			
			
101-layer ResNet G and D			
			

Figure 2: Different GAN architectures trained with different methods. We only succeeded in training every architecture with a shared set of hyperparameters using WGAN-GP.

References

- Generative adversarial nets [Ian Goodfellow et al. NIPS 2014]
- Generative adversarial networks [Ian Goodfellow et al. NIPS 2016 Tutorial]
- Unsupervised representation learning with deep convolutional generative adversarial networks [Alec Radford et al. ICLR 2016]
- Towards principled methods for training generative adversarial networks [Martin Arjovsky et al. ICLR 2017]
- Wasserstein gan [Martin Arjovsky et al.2017]
- Improved training of wasserstein gans [Ishaan Gulrajani et al.2017]
- Hung-Yi Lee (2017). Improving GAN. `https://www.youtube.com/watch?v=KSN4QYgAtao`
- Vincent Herrmann. Wasserstein GAN and the Kantorovich-Rubinstein Duality. `https://vincentherrmann.github.io/blog/wasserstein/`
- Cedric Villani (2009). Optimal Transport: Old and New. Grundlehren der mathematischen Wissenschaften. Springer, Berlin.