Generative Adversarial Networks

Chunpai Wang

September 24, 2018

1 Goal of GAN

The goal of GAN is to train a generator G such that the discriminator D cannot discriminate the real samples and the samples generated by G. The ultimate goal is to approximate the real distribution.

2 Algorithm and Architecture

Algorithm 1: Generative Adversarial Networks.

Initialize generator G and discriminator D ;
for *number of training iterations* **do**
  (Train Discriminator D, used to evaluate the JS-Divergence)
  for k *steps* **do**
    Fix generator and sample minibatch of m noise sample $\{z^{(1)}, ..., z^{(m)}\}$ from noise prior $P_G(z)$.
    Sample minibatch of m examples $\{x^{(1)}, ..., x^{(m)}\}$ from real distribution $P_{data}(x)$.
    Discriminator learns to assign high scores to real objects and low scores to generated objects, by updating the parameters in D with stochastic gradient ascent.

$$\theta_D = \theta_D + \nabla_{\theta_D} \frac{1}{m} \sum_{i=1}^{m} \Big[\log D(x^{(i)}) + \log(1 - D(G(z^{(i)}))) \Big] \tag{1}$$

  end
  (Train Generator G, same as minimize the JS-Divergence)
  # Fix discriminator D, and update generator G. Generator learns to "fool" the discriminator,
  # such that discriminator will give high score to the objects generated by G;
  Sample minibatch of m noise samples $\{z^{(1)}, ..., z^{(m)}\}$ from noise prior $p_G(z)$.
  Update the generator by descending its stochastic gradient:

$$\theta_G = \theta_G - \nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} \Big[\log(1 - D(G(z^{(i)}))) \Big] \tag{2}$$

end

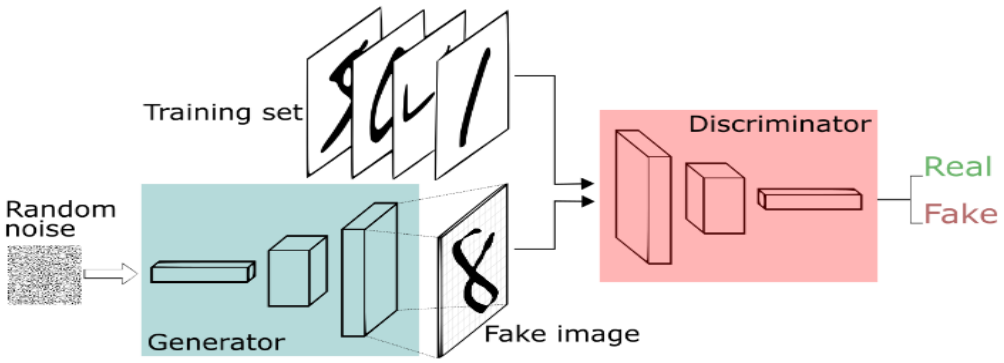

Figure 1: Caption

In summary, the discriminator D and generator G play the following two player minimax game with value

function $V(G, D)$:

$$\min_G \max_D V(D, G) = \min_G \max_D E_{x \sim P_{data}(x)} [\log D(x)] + E_{z \sim p_z(z)} [\log(1 - D(G(z)))] \tag{3}$$

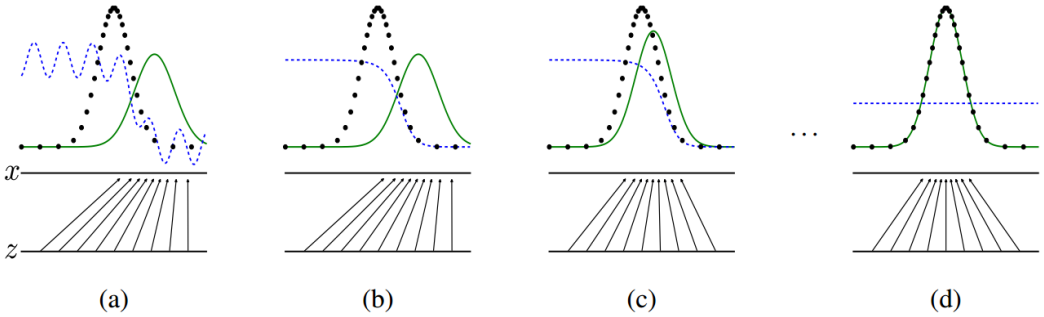

Figure 2: Training of GAN. Black dot line represents the real data distribution $P_{data}(x)$ and green line represents the generated distribution $p_G(z)$, where $p(z)$ denotes the prior of x and the up-ward arrow means the mapping $x = G(z)$. Blue dash line represents the discriminator D. Note that, the z is uniformly distributed, but after mapping $G(z)$ is not uniformly distributed.

If we don't use uniformly distributed z, what will happen ? It does not matter, since the neural network can still find the mapping $x = G(z)$.

Now, let's see how can we play this minimax game. Once G is fixed, we want to maximize the value function such that discriminator D will give high value to real data and low value to generated data. Thus, the optimal D^* is

$$D^* = \arg\max_D V(D, G = g) \tag{4}$$

Now when discriminator is fixed, and we want to make the generator generate a data such that the discriminator will give high value on it, and the optimal G is

$$G^* = \arg\min_G V(G, D = d) = \arg\min_G E_{z \sim p_z(z)} [\log(1 - D(G(z)))] \tag{5}$$

The remaining questions are

- how can we obtain those optimal D and G in each iteration ?
- how can we prove it will converge to $P_{data} = P_G$?

3 Basic Theory

3.1 MLE and KL-Divergence

One interesting thing is MLE can be interpreted as minimizing the KL-divergence from the true distribution to the estimated distribution. Our ultimate goal is to find the distribution $P_{data}(x)$, which we cannot directly to get it, but we can sample from it. We would like to use a parameterized distribution $P_G(x; \theta)$ such that it is close to $P_{data}(x)$. That is, we sample m data from the real distribution $P_{data}(x)$, we would like to find the parameter θ such that the likelihood is maximized

$$\theta^* = \arg\max_\theta P(x^1, ..., x^m; \theta) = \arg\max_\theta \prod_{i=1}^m P_G(x^{(i)}; \theta) \tag{6}$$

Note that, the data are sampled from the real distribution $P_{data}(x)$, thus we have

$$\arg\max_\theta \prod_{i=1}^m P_G(x^{(i)}; \theta) = \arg\max_\theta \log \prod_{i=1}^m P_G(x^{(i)}; \theta) \tag{7}$$

$$= \arg\max_\theta \sum_{i=1}^m \log P_G(x^{(i)}; \theta) \tag{8}$$

$$= \arg\max_\theta \frac{1}{m} \sum_{i=1}^m \log P_G(x^{(i)}; \theta) \tag{9}$$

By the law of large number, sample averages converge almost surely to expectation, we have

$$\arg\max_{\theta} \frac{1}{m} \sum_{i=1}^{m} \log P_G(x^{(i)}; \theta) \approx E_{x \sim P_{data}} [\log P_G(x; \theta)] = \sum \frac{1}{N} \log P_G(x^i; \theta) \tag{10}$$

since we assume we sample x from P_{data} uniformly, and $p(x_i) = \frac{1}{N}$, where N denotes by the total number of samples. Hence, we have

$$\begin{aligned}
\arg\max_{\theta} \frac{1}{m} \sum_{i=1}^{m} \log P_G(x^{(i)}; \theta) &\approx \arg\max_{\theta} E_{x \sim P_{data}} [\log P_G(x; \theta)] && (11)\\
&= \arg\max_{\theta} \int_x P_{data}(x) \log P_G(x; \theta) && (12)\\
&= \arg\max_{\theta} \int_x P_{data}(x) \log P_G(x; \theta) dx - \arg\max_{\theta} \int_x P_{data}(x) \log P_{data}(x) dx && (13)\\
&= \arg\max_{\theta} \int_x P_{data}(x) \left[\log P_G(x; \theta) - \log P_{data}(x)\right] dx && (14)\\
&= \arg\max_{\theta} - \int_x P_{data}(x) \left[\log P_{data}(x) - \log p_G(x; \theta)\right] dx && (15)\\
&= \arg\min_{\theta} \int_x P_{data}(x) \left[\log P_{data}(x) - \log p_G(x; \theta)\right] dx && (16)\\
&= \arg\min_{\theta} KL\left(P_{data} \| p_G\right) && (17)
\end{aligned}$$

Traditional approaches to generative modeling relied on maximizing likelihood, or equivalently minimizing the KL divergence between our unknown data distribution P_{data} and our generator's distribution P_G, which parameterized by θ. But, some issues of KL-divergence are:

- If $P_{data}(x) > P_G(x)$, then x is a point with higher probability of coming from the data than being a generated sample. When $P_{data}(x) > 0$ but $P_G(x) \to 0$, the integral inside the KL-divergence grows quickly to infinity, meaning that this cost function assigns an extremely high cost to a generator's distribution not covering parts of the data. Hence, the generator will generate images nearly same as the training data, which is so-called "mode-dropping".
- If $P_{data}(x) < P_G(x)$, then x is a point with lower probability of coming from the data than being a generated sample. When $P_{data}(x) \to 0$ and $p_G(x) > 0$, the value inside the KL-divergence goes to 0, meaning that this cost function will pay extremely low cost for generating fake looking samples. Thus, the generator will output an image that does not look real.

Thus, minimizing KL-divergence is not a desirable objective to approximate the real distribution and generate fake samples. The question is if there is an existing metric that does not have the shortcoming of KL-divergence ? If not, can we create one ?

3.2 How to Define A General $P_G(x)$? Why Discriminator ?

The question rises from the minimizing the divergence between two distributions, where sometimes we restrict the parameter θ. GAN exploits a more general distribution $P_G(x)$, which G is a neural network and the network defines the probability distribution P_G.

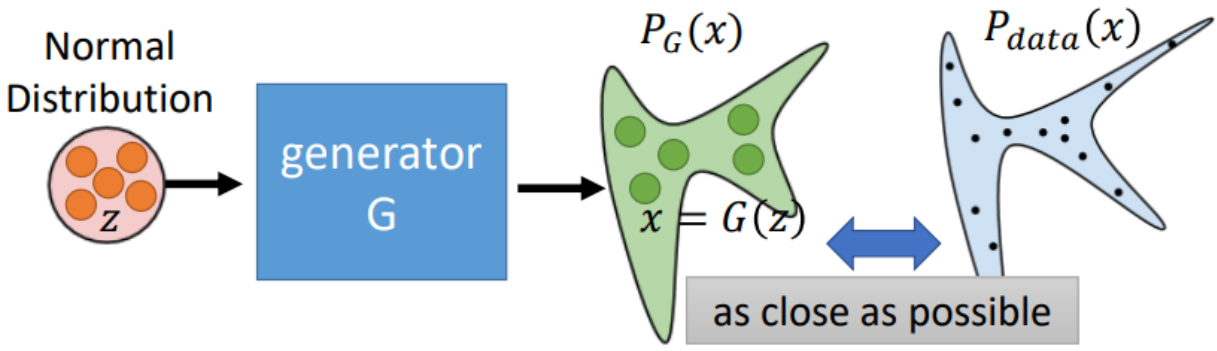

Figure 3: The purpose of generator in GAN

In GAN, the optimal $G^* = \arg\min_G Div(P_{data}, P_G)$, where Div denotes a kind of divergence which measures the difference of two distributions (not limited to KL-divergence). We will see later that a so-called JS-divergence can do a better job than KL-divergence.

One problem here is how can we find the optimal G^* in the situation that we don't know the both distribution P_G and P_{data}? If we don't know the distributions, how can we minimize the divergence of two distributions.

Although we do not know the distributions of P_G and P_{data}, but we can sample from them. Then, we can use a discriminator D to evaluate the divergence between these two distributions, where it will give high value to samples from P_{data} and low value to samples from P_G, and the objective of D is to maximize the value function (when G is fixed) as follows

$$V(G, D) = E_{x \sim P_{data}}[\log D(x)] + E_{x \sim P_G}[\log(1 - D(x))] \tag{18}$$

and

$$D^* = \arg\max_D V(D, G) \tag{19}$$

The interesting part is, the objective function (that maximizing the value function $\arg\max_D V(D, G)$) is exactly same as training a binary classifier. For example, when the samples A from P_{data} and samples B from P_G are close and merge to each other, then the binary classifier is hard to classify these A and B classes, thus the loss of training will be always high. It also means the objective value $V(G, D)$ is always low. We can also interpret that the divergence of these two classes is small. In other words, if a discriminator can easily classify two classes, then it means the divergence of two classes is large. Therefore, we can use the discriminator to evaluate the divergence of two distributions, and $G^* = \arg\min_G Div(P_{data}, P_G)$. We do not need a metric with analytic form to measure two distributions any longer, where two distributions are unknown !!!

3.3 Jensen-Shannon Divergence

We will see that the maximum objective value $\max V(G, D)$ with G fixed, is related to JS-Divergence:

$$JSD(P\|Q) = \frac{1}{2}KL(P\|M) + \frac{1}{2}KL(Q\|M) \tag{20}$$

where $M = \frac{1}{2}(P + Q)$.

$$\begin{aligned} V(G, D) &= E_{x \sim P_{data}}[\log D(x)] + E_{x \sim P_G}[\log(1 - D(x))] && (21)\\ &= \int_x P_{data}(x) \log D(x) dx + \int_x P_G(x)[\log(1 - D(x))] dx && (22)\\ &= \int_x \{P_{data}(x) \log D(x) + P_G(x)[\log(1 - D(x))]\} \, dx && (23) \end{aligned}$$

Assume $D(x)$ is extremely powerful and can be any function, to maximize the $V(G, D)$, the optimal D^* maximizes the value in terms of every possible x

$$P_{data}(x) \log D(x) + P_G(x)[\log(1 - D(x))] \tag{24}$$

Now, given a x, to find the optimal D^*, we can set the derivative of above formula w.r.t D, and set it to 0. Based on the calculus, for $f(D) = a \cdot log(D) + b \cdot \log(1 - D)$

$$\frac{df(D)}{dD} = a * \frac{1}{D} + b * \frac{1}{1 - D} * (-1) = 0 \tag{25}$$

$$D^* = \frac{a}{a + b} = \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} = D^*_G(x) \tag{26}$$

and the corresponding optimal value is

$$\max_D V(G,D) = V(G,D^*) \tag{27}$$

$$= E_{x\sim P_{data}}[\log D^*(x)] + E_{x\sim P_G}[\log(1-D^*(x))] \tag{28}$$

$$= E_{x\sim P_{data}}[\log \frac{P_{data}(x)}{P_{data}(x)+P_G(x)}] + E_{x\sim P_G}[\log \frac{P_G(x)}{P_{data}(x)+P_G(x)}] \tag{29}$$

$$= \int_x P_{data}(x)[\log \frac{P_{data}(x)}{P_{data}(x)+P_G(x)}]dx + \int_x P_G(x)[\log \frac{P_G(x)}{P_{data}(x)+P_G(x)}]dx \tag{30}$$

$$= \int_x P_{data}(x)[\log \frac{\frac{1}{2}P_{data}(x)}{\frac{1}{2}(P_{data}(x)+P_G(x))}]dx + \int_x P_G(x)[\log \frac{\frac{1}{2}P_G(x)}{\frac{1}{2}(P_{data}(x)+P_G(x))}]dx \tag{31}$$

$$= 2\log\frac{1}{2} + \int_x P_{data}(x)[\log \frac{P_{data}(x)}{\frac{1}{2}(P_{data}(x)+P_G(x))}]dx + \int_x P_G(x)[\log \frac{P_G(x)}{\frac{1}{2}(P_{data}(x)+P_G(x))}]dx \tag{32}$$

$$= 2\log\frac{1}{2} + KL(P_{data}\|\frac{P_{data}+P_G}{2}) + KL(P_G\|\frac{P_{data}+P_G}{2}) \tag{33}$$

$$= 2\log\frac{1}{2} + 2\cdot JSD(P_{data}\|P_G) \tag{34}$$

$$= -2\log(2) + 2\cdot JSD(P_{data}\|P_G) \tag{35}$$

We will find that the main issue of GAN is existing in this equality. Theoretically, we would like to train the discriminator to optimum, and evaluate the JS-Divergence, then use gradient descent to minimize it. However, in practices we observe gradient vanishing. One fact is, the images is a low-dimensional manifold in a high dimensional space. The overlap of generated samples and true samples can be neglected [5], and the JS-Divergence is always 0 ?

3.4 Combine G^* and D^*

Recall that

$$D^* = \arg\max_D V(G,D) \tag{36}$$

and

$$G^* = \arg\min_G Div(P_G, P_{data}) \tag{37}$$

Since the Div denotes by any divergence, and the optimal discriminator is related to JS-divergence, thus we can replace the Div with JS-divergence

$$G^* = \arg\min_G \max_D V(G,D) \tag{38}$$

The figure below shows an example to solve this minmax problem.

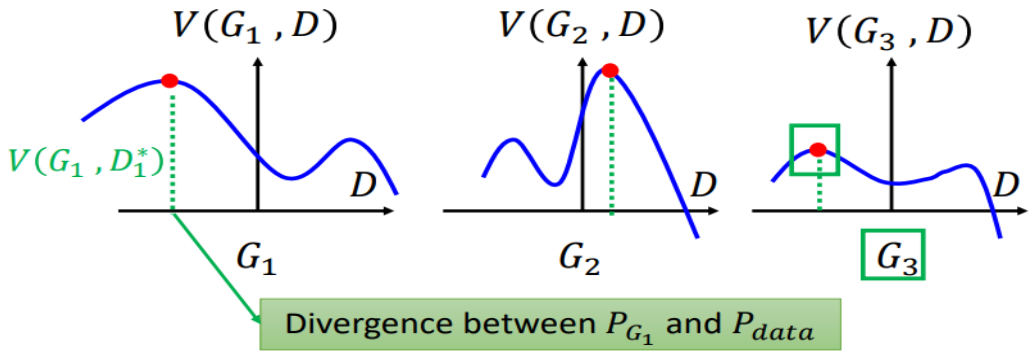

Figure 4: Assume there are only 3 generators in the generator space. For each generator, we can find the optimal D^* that maximize the value V, and then choose the generator with minimum value among those $V(G,D^*)$. In this example, $G3$ is the optimal generator.

We can see that the global minimum of the virtual training criterion $C(G)$ is achieved if and only if $P_G = P_{data}$. At that point, $C(G)$ achieves the value $-2\log 2$. Since the JS-divergence is always non-negative, and zero if and only if they $P_G = P_{data}$. We can see that the algorithm to solve $\min_G \max_D V(D,G)$ is actually the algorithm 1. In algorithm, the steps of training the discriminator is actually to evaluate the JS-divergence, and the steps of training the generator is to find the optimal G that minimize the JS-divergence.

3.5 Convergence of Algorithm 1

What algorithm 1 does is to solve the problem

$$G^* = \arg\min_G \max_D V(G, D)$$

which can be interpreted as to minimize the JS-divergence between P_{data} and P_G.

Proposition 1. *If G and D have enough capacity, and at each step of algorithm 1, the discriminator is allowed to reach its optimum given G, and P_G is updated so as to improve the criterion*

$$E_{x\sim P_{data}}[\log D^*(x)] + E_{x\sim P_G}[\log(1 - D^*(x))]$$

then P_G converges to P_{data}.

Key points: What is the meaning of "G and D have enough capacity" ? Should the discriminator reach its optimum given G ?

Proof. Now assume the discriminator can reach its optimum given G, and we denote the optimal value $L(G) = \max_D V(G, D)$. In addition, P_G is updated as

$$\theta_G = \theta_G - \eta \frac{\partial L(G)}{\partial \theta_G} \tag{39}$$

Now we would like to compute the derivative of $L(G) = \max_D V(G, D)$ w.r.t θ_G. Note that $V(G, D)$ is convex in P_G, because the expectation of convex function is convex, namely

$$\mathbb{E}_{z\sim P_z(z)}[\log(1 - D(G(z)))] \tag{40}$$

is convex. The $\max_D V(G, D)$ is pointwise supremum, which is also convex with unique global optima. However, the update rule does not necessarily decrease the JS-Divergence. Because when we update the θ_{G_0} to θ_{G_1}, we need to exploit the discriminator to compute the new JS-Divergence between $P_{G_1}(x)$ and $P_{data}(x)$. Thus, the algorithm is not decreasing the JS-Divergence between $P_{G_0}(x)$ and $P_{data}(x)$ any more. Thus, in the algorithm only does one-step update on the θ_G , assuming it will not change the JS-divergence too much (see figure), and the algorithm will converge.

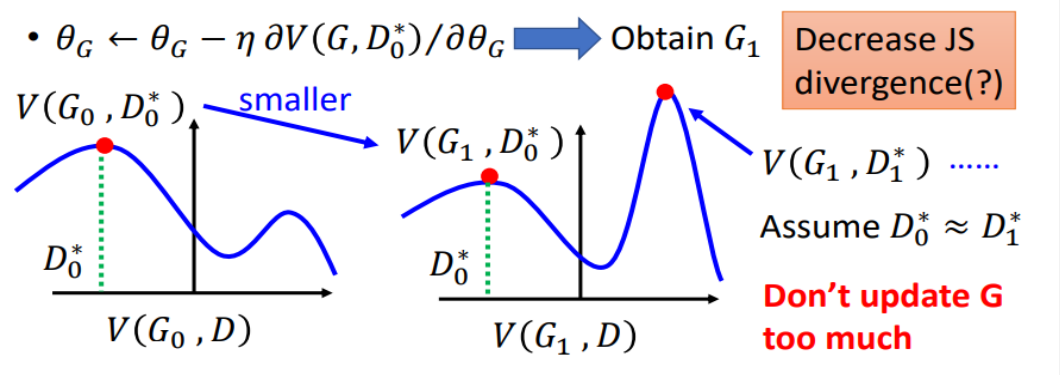

Figure 5: Potential Issue in GAN.

□

4 Implementation Issues of GANs

- In practice, we must implement the game using an iterative numerical approach. Optimizing D to completion in the inner loop of training is computationally prohibitive, and on finite datasets would result in overfitting. Instead, we alternative between k steps of optimizing D and one step of optimizing G. This results in D being maintained near its optimal solution, so long as G changes slowly enough. We cannot train the discriminator too well, otherwise the gradient vanishes. However, this theoretically is not evaluating the JS-Divergence any longer. The main reason is, the overlap of generated samples and true samples can be neglected, which leads to JS-divergence equals to $\log 2$, for any x there are 4 possibilities

- $P_{data}(x) = 0$ and $P_G = 0$
- $P_{data}(x) \neq 0$ and $P_G \neq 0$
- $P_{data}(x) = 0$ and $P_G \neq 0$
- $P_{data}(x) \neq 0$ and $P_G = 0$

For the first case, we cannot compute the JS-Divergence; for second case, since the overlap is 0, its contribution to JS-Divergence is also 0; for third and fourth case, the JS-Divergence is

$$JSD(P_{data}\|P_G) = \frac{1}{2}KL(P_{data}\|\frac{P_{data}+P_G}{2}) + \frac{1}{2}KL(P_G\|\frac{P_{data}+P_G}{2}) = \log 2 \tag{41}$$

- In practice, equation 3 may not provide sufficient gradient for G to learn well. Early in learning when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, $\log(1 - D(G(z)))$ saturates, which means the gradient is very small. Rather than training G to minimize $E[\log(1 - D(G(z)))]$ we can train G to minimize $E[-\log D(G(z))]$. This objective function results in the same fixed point of the dynamics of G and D but provides much stronger gradients early in learning [6]. However, if we use the alternative $-\log(D)$ trick, we will still see mode collapse and unstable gradient. The reason is if we replace the $\log(1 - D(G(z)))$ with $\log(-D(G(z)))$, the formula (27) will becomes

$$KL(P_{data}\|P_G) - 2 \cdot JS(P_{data}\|P_G) \tag{42}$$

5 Structure Learning and GAN

Structure learning can be viewed as an extreme one-shot or zero-shot learning. If we consider each possible output as a class, because the output space is huge, most classes do not have any training data, and machine has to create new stuff during testing. Machine has to learn to do planning:

- Machine generates objects component-by-component, but it should have a big picture in its mind.
- Because the output components have dependency, they should be considered globally.

There are two main-stream approaches to solve the structure learning problem:

- Button-up. Learn to generate the object at the component level, which generate the value pixel-by-pixel, and it will lose global dependency.
- Top-down. Evaluating the whole object, and find the best one.

Here, we may view the button-up as generator, and top-down as discriminator.

References

[1] Short Summation of GANs https://zhuanlan.zhihu.com/p/34916654

[2] A Tutorial on Generative Adversarial Networks https://www.jiqizhixin.com/articles/2017-10-1-1

[3] GAN-Why it is so hard to train Generative Adversarial Networks! https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

[4] https://zhuanlan.zhihu.com/p/39719663

[5] Arjovsky, Martin, and Léon Bottou. "Towards principled methods for training generative adversarial networks." arXiv preprint arXiv:1701.04862 (2017).

[6] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014.