# Generative Adversarial Networks

Amaires@June 2024

Generative Adversarial Networks (GAN) are an approach to generative artificial intelligence. It is the first known model to produce new photorealistic images automatically.

## 1 Original GAN

With data samples $\dot{x}$ that follow a certain unknown distribution $p(x)$, the idea is to have a neural network generator $G_\theta$, parameterized by $\theta$, which transforms samples $z \sim N(0, I)$ to $\tilde{x}$ that follows $p(x)$. Figure 1 depicts the architecture. In order to train $G_\theta$, some optimization objective is needed to give $G_\theta$ feedback about whether $\tilde{x}$ really looks like it is drawn from $p(x)$. A binary descriminator/classifier $D_\phi$ that returns the probability that $\tilde{x}$ follows $p(x)$ would serve the purpose. Figure 2 shows the architecture we have so far. The introduction of $D_\phi$ merely defers the responsibility of guiding $G_\theta$. How do we train $D_\phi$? We could feed it both real samples $(x, y) = (\dot{x}, 1)$ and artificial samples $(x, y) = (\tilde{x}, 0)$ created by $G_\theta$. Figure 3 shows the complete GAN architecture.

The loss function $\mathscr{L}$ of $D_\phi$ is the common binary cross entropy function, shown below (see `amaires.github.io/OptimizationObjective/` for a refresher):

$$\mathscr{L} = \min_\theta -E_{x,y\sim p(x,y)}(y \log D_\phi(x) + (1-y)\log(1 - D_\phi(x)) \tag{1}$$

Remove the negation in (1) and rewrite it in conditional expectation form:

$$\begin{aligned}\mathscr{L} &= \max_\phi E_{y\sim p(y)} E_{x\sim p(x|y)}(y \log D_\phi(x) + (1-y)\log(1 - D_\phi(x)) \\ &= \max_\phi [\Pr(y=1) E_{x\sim p(x|y=1)} \log D_\phi(x) + \Pr(y=0) E_{x\sim p(x|y=0)} \log(1 - D_\phi(x)]\end{aligned}$$

If the same number of real samples $\dot{x}$ and artificial samples $\tilde{x}$ are fed to $D_\phi$ in each batch/mini-batch, $Pr(y=1) = Pr(y=0) = \frac{1}{2}$. Also note that $x \sim p(x|y=1)$ is the same as $\dot{x} \sim p(\dot{x})$ and $x \sim p(x|y=0)$ is the same as $\tilde{x} \sim p_\theta(\tilde{x}|z)$, $\mathscr{L}$ can be further written:

$$\begin{aligned}\mathscr{L} &= \max_\phi [\frac{1}{2} E_{\dot{x}\sim p(\dot{x}))} \log D_\phi(\dot{x}) + \frac{1}{2} E_{\tilde{x}\sim p_\theta(\tilde{x})} \log(1 - D_\phi(\tilde{x})] \\ L &= \max_\phi [\frac{1}{2} E_{\dot{x}\sim p(\dot{x}))} \log D_\phi(\dot{x}) + \frac{1}{2} E_{z\sim N(0,I)} E_{\tilde{x}\sim p_\theta(\tilde{x}|z)} \log(1 - D_\phi(\tilde{x}))]\end{aligned}$$

After removing the constant $\frac{1}{2}$, if for every sample $z$, only one sample is drawn for $\tilde{x}$, which is what $G_\theta$does, $\mathscr{L}$ becomes

$$\mathscr{L} = \max_\phi [E_{\dot{x}\sim p(\dot{x}))} \log D_\phi(\dot{x}) + E_{z\sim N(0,I)} \log(1 - D_\phi(G_\theta(z))]$$

Note that the objective is parameterized by both $\theta$ and $\phi$, but it is maximized only in terms of $\phi$, the paramters for the discriminator. It is also worth noting that $D_\phi$ gives the probability if a sample follows $p(x)$, but not $p(x)$ itself. Also, $G_\theta$ generates a sample $\tilde{x}$, not $p_\theta(\tilde{x})$.

The process so far optimizes $\phi$ to make $D_\phi$ better at telling real samples apart from samples created by a fixed $G_\theta$. For each batc/mini-batch of samples, $\phi$ takes a gradient *ascent* step. This process does not seem to improve $G_\theta$ whatsoever. If $\theta$ also takes a gradient ascent step using $\nabla_\theta \mathscr{L}$, it also makes $D_\phi$ better. In this case, it is probably equivalent to making samples created by $G_\theta$ more obviously fake, the opposite of what we want. The solution to this last piece of the GAN puzzle is to update $\theta$ with a gradient *descent* step. More formally, our objective function is

$$\mathscr{L} = \min_\theta \max_\phi [E_{\dot{x}\sim p(\dot{x}))} \log D_\phi(\dot{x}) + E_{z\sim N(0,I)} \log(1 - D_\phi(G_\theta(z))]$$

Intuitively, $D_\theta$ tries to tell real samples apart from artificial samples, and $G_\theta$ tries to create artificial samples that are hard to distinguish from real samples, hence the name generative adversarial network.

In GAN, $p(x)$ and $p_\theta$ are never explicitly modeled. $G_\theta$ can produce good samples following $p_\theta(x)$, but it does not know the form of $p_\theta(x)$. In other words, $G_\theta$ defines a sampling process without knowing its distribution.


Figure 1: Generator $G_\theta$

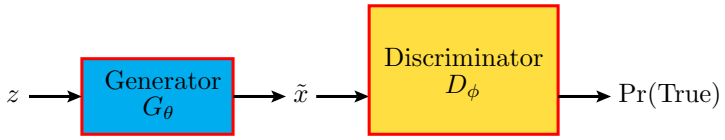


Figure 2: Generator $G_\theta$ and Discriminator $D_\phi$

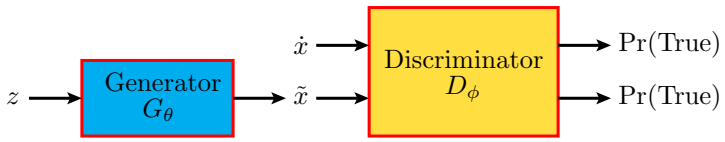


Figure 3: The Complete GAN Architecture

## 1.1 Jenson-Shannon Divergence

For a given $G_\theta$, $D_\phi$'s objective is to maximize

$$\begin{aligned} L_{\theta,\phi} &= E_{x\sim p(x)} \log D_\phi(x) + E_{x\sim p_\theta(x)}(1 - \log D_\phi(x)) \\ &= \int [p(x) \log D_\phi(x) + p_\theta(x) \log(1 - D_\phi(x))] dx \end{aligned} \tag{2}$$

Let $l_{\theta,\phi}(x)$ be the function under the integral in (2). Assuming $D_\phi$ is flexible and powerful enough, when $L_{\theta,\phi}$ is maximized with parameter $\phi^*$, $l_{\theta,\phi^*}(x)$ is also maximized everywhere. Take $l_{\theta,\phi}(x)$'s derivative against $D_\phi(x)$, we have

$$\frac{dl_{\theta,\phi}(x)}{dD_\phi(x)} = \frac{p(x)}{D_\phi(x)} - \frac{p_\theta(x)}{1 - D_\phi(x)}$$

Set the expression to 0, you derive $D_{\phi^*}(x)$:

$$D_{\phi^*}(x) = \frac{p(x)}{p(x) + p_\theta(x)} \tag{3}$$

Substitute $D_\phi(x)$ with (3) in (2), we have:

$$\begin{aligned} L_\theta &= L_{\theta,\phi^*} \\ &= \int [p(x) \log D_{\phi^*}(x) + p_\theta(x) \log(1 - D_{\phi^*}(x))] dx \\ &= \int [p(x) \log \frac{p(x)}{p(x) + p_\theta(x)} + p_\theta(x) \log \frac{p_\theta(x)}{p(x) + p_\theta(x)}] dx \\ &= \int [p(x) \log \frac{\frac{1}{2}p(x)}{\frac{1}{2}(p(x) + p_\theta(x))} + p_\theta(x) \log \frac{\frac{1}{2}p_\theta(x)}{\frac{1}{2}(p(x) + p_\theta(x))}] dx \\ &= \int p(x) \log \frac{p(x)}{\frac{1}{2}(p(x) + p_\theta(x))} dx + \int p_\theta(x) \log \frac{p_\theta(x)}{\frac{1}{2}(p(x) + p_\theta(x))} dx - \log 2 \int p(x) dx - \log 2 \int p_\theta(x) dx \\ &= DL[p(x), \frac{p(x) + p_\theta(x)}{2}] + DL[p_\theta(x), \frac{p(x) + p_\theta(x)}{2}] - \log 4 \end{aligned}$$

The first two terms is actually twice the Jenson-Shannon Divergence (JSD) defined as:

$$JS(p_1, p_2) = \frac{1}{2}[DL(p_1, \frac{p_1 + p_2}{2}) + DL(p_2, \frac{p_1 + p_2}{2})]$$

Using JSD, $L_\theta$ is further simplified as:

$$L_\theta = 2JS(p(x), p_\theta(x)) - \log 4$$

Now it becomes clear that the generator $G_\theta$ is really optimizing the J-S divergence between $p_\theta(x)$ and $p(x)$ given the optimal $D_{\phi*}$.

The Jenson-Shannon Divergence has a few properties as well.

- $JSD(p, q) \geq 0$
- $JSD(p, q) \geq 0$ iff $p = q$
- Unlike the K-L divergence, $JSD(p, q)$ is symmetric. That is $JSD(p, q) = JSD(q, p)$.

### 1.2 Contrast Maximization

When $D_\phi$ is trained reasonably well, which means $D_\phi(x)$ is close to 1 for real samples, and is close to 0 for artifical samples, $L_{\theta,\phi}$can be shown to maximize the contrast of $D_\phi(x)$ between real and artifical samples, as shown below.

$$\begin{aligned} L_{\theta,\phi} &= E_{x\sim p(x)} \log D_\phi(x) + E_{x\sim p_\theta(x)}(1 - \log D_\phi(x)) \\ &\simeq E_{x\sim p(x)}(D_\phi(x) - 1) + E_{x\sim p_\theta(x)} - D_\phi(x) \\ &= E_{x\sim p(x)} D_\phi(x) - E_{x\sim p_\theta(x)} D_\phi(x) - 1 \end{aligned} \tag{4}$$

(4) uses the first derivative of $\log(\cdot)$ at $\log(1)$ for approximation. Of course, when $D_\theta$ is not a very good discriminator yet, the above does not hold.

## 2 f-GAN

too much math and too little practical impact to write about... will pick up later

## 3 WGAN

### 3.1 Problems with GAN

GAN is known to generate very impressive photorealistic images, but it also has a few well documented drawbacks.

The first is a problem known as mode collapse. The discriminator $D_\phi$ only cares about distinguishing real samples from artificial samples. It does not care about whether those artificial samples have broad coverage or not. For example, suppose the real samples include images of different animals such as cats, dogs, and horses. $G_\theta$ is happy to generate pictures of only dogs as long as these pictures become more real each time $\theta$ is updated. $D_\phi$ is also perfectly happy with $G_\theta$'s behavior as long as each time $\phi$ is updated, $D_\phi$ can tell real animal pictures from these artifical dog picutres a little better.

The second notorious problem with GAN is its difficulty to train. Unlike other deep learning problems which see their loss function decreases gradually until converging to 0 in training, GAN's minimax objective does not offer any such guarantee. In practice, GAN's objective keeps oscillating during training. Deciding when to stop training is often a manual process. Another reason that prevents GAN's effective training has to do with $D_\phi$'s final activation function, typically sigmoid. When real samples and artificial samples are far apart, it is easy for $D_\phi$ to distinguish them. In this case, sigmoid's outputs are very close to 1 for real samples, and 0 for artificial samples. Its derivative is very close to 0, a problem known as vanishing gradient. These 0 gradients cannot provide effective back-propagation for $G_\theta$ to improve its sample generating process.

### 3.2 Intuition of WGAN

The inspiration of Wasserstein GAN (WGAN) comes from contrast maximization described in Section 1.2 and sigmoid's vanishing gradient problem described in Section 3.1. WGAN's objective is

$$\min_\theta \max_\phi L_{\theta,\phi} = \min_\theta \max_\phi [E_{x\sim p(x)} D_\phi(x) - E_{x\sim p_\theta(x)} D_\phi(x)]$$

Basically, $D_\phi$ tries to maximize its output for real samples and minimize its output for artificial samples. $G_\theta$ on the other hand tries to generate artifical samples that also get large outputs.

This objective looks just like the contrast maximization loss in Section (1.2), but there are two main differences: it is no longer an approximation in a narrow range and sigmoid is not used to compress $D_\phi$'s output to between 0 and 1. Completely removing $D_\phi$'s output value constraint may, however, pose another problem; $L_{\theta,\phi}$ may grow very rapidly out of bound and may still take the form of a sigmoid function and stifle back-propagation. Ideally, we'd like $D_\phi$ to behave roughly like a linear function of $x$. Given deep neural networks are differentiable for almost all input, we could force the norm of $D_\phi$'s gradient to be close to 1 everywhere:

$$\|\nabla_x D_\phi(x)\| \simeq 1$$

This constraint can be added to $L_{\theta,\phi}$ as a penalty term:

$$(\|\nabla_x D_\phi(x)\| - 1)^2$$

This penalty term needs to be numerically computable. Averaging over all possible value of $x$ is out of the question, but one possbility is to average it over both the real samples and the artificial samples, as shown below:

$$L_{\theta,\phi} = E_{x\sim p(x)} D_\phi(x) - E_{x\sim p_\theta(x)} D_\phi(x) - \lambda E_{x\sim p(x)}(\|\nabla_x D_\phi(x)\| - 1)^2 - \lambda E_{x\sim p_\theta(x)}(\|\nabla_x D_\phi(x)\| - 1)^2$$

where $\lambda$ is the knob adjusting the relative importance between maximizing contrast and making $D_\phi$ roughly a linear function.

A variant of the above formulation creates samples by randomly and linearly interpolating between real and artificial samples, and averages the penalty term over these samples instead. The final objective function, expressed in numerical computation form, is the following:

$$L_{\theta,\phi} = \frac{1}{N}\sum_i D_\phi(\dot{x}_i) - \frac{1}{N}\sum_i D_\phi(\tilde{x}_i) - \frac{\lambda}{N}\sum_i (\|\nabla_x D_\phi[\varepsilon_i \dot{x}_i + (1 - \varepsilon_i \tilde{x}_i)]\| - 1)^2$$

where $\varepsilon_i \sim U(0,1)$. GAN with this gradient norm penalty is called Wasserstein Generative Adversarial Network - Gradient Penalty (WGAN-GP).

Both $D_\phi$ and $G_\theta$ affects the penalty term. Though it is not mentioned in the original WGAN-GP paper, I don't think it is desirable to add the penalty term to $G_\theta$'s loss function. WGAN-GP's objective should instead be

$$\max_\phi [\frac{1}{N}\sum_i D_\phi(\dot{x}_i) - \frac{1}{N}\sum_i D_\phi(\tilde{x}_i) - \frac{\lambda}{N}\sum_i (\|\nabla_x D_\phi[\varepsilon_i \dot{x}_i + (1-\varepsilon_i)\tilde{x}_i]\| - 1)^2]$$
$$\min_\theta [\frac{1}{N}\sum_i D_\phi(\dot{x}_i) - \frac{1}{N}\sum_i D_\phi(\tilde{x}_i)]$$

There are a couple more things worth noting.

- $\nabla_x D_\phi(x)$ is the derivative against $x$, not against $\phi$. What is used in back-propagation to update $\phi$ includes $(\|\nabla_x D_\phi(x)\| - 1)^2$'s derivative against $\phi$.
- Obviously, with WGAN, $G_\theta$ no longer minimizes the Jenson-Shannon divergence even given an optimal $D_{\phi^*}$.

## 3.3 Math of WGAN

### 3.3.1 Infimum and supremum

Most people are familiar with the concept of maximum and minimum. Explicitly, if $X$ is a (partially) ordered set and $S$ a subset, then $\bar{s}$ is the maximum of $S$ iff $\bar{s} \in S$ and $s \leq \bar{s}$ for all $s \in S$. Similarly, $\underline{s}$ is the minimum of $S$ iff $\underline{s} \in S$ and $s \geq \underline{s}$ for all $s \in S$.

The supremum (sup) of $S$ can be defined like this. Let $T = \{t \in X | s \leq t \, \forall s \in S\}$, which defines the set of elements greater than all members of $S$. If $T$ is empty, $S$'s supremum does not exist, otherwise it is the mininum of $T$. If S has a maximum, it must be the same as $S$'s supremum. Even if $S$ does not have a maximum, it may still have a supremum. Below are three examples comparing maximum and supremum.

1. $S = \{x | x \leq 2\}$: $S$'s maximum is 2, and its supremum is 2 as well.
2. $S = \{x | x < 2\}$: $S$ does not have a maximum, but its supremum is 2.
3. $S = \{x | x > 2\}$: $S$ has neither a maximum nor a supremum.

Similar comparisons can be made between minimum and infimum. Informally, if one uses supremum and maximum interexchangeably, little is lost. The same goes for infimum and minimum.

### 3.3.2 Wasserstein Distance

K-L divergence and J-S divergence are often used to measure the closeness between two distributions. In fact, VAE (`amaires.github.io/VAE`) uses K-L divergence for optimization and GAN's $G_\theta$ minimizes the J-S divergence given an optimal $D_{\phi^*}$. Unfortunately, these two measures have discontinuity when two distributions have disjoint supports (the support of a function is the subset of the function domain not mapped to 0). For example, given two distributions defined below:

$$p(x) = \begin{cases} 1 & x = 0 \\ 0 & x \neq 0 \end{cases} \text{ and } p_\theta(x) = \begin{cases} 1 & x = \theta \\ 0 & x \neq \theta \end{cases}$$

It is not hard to figure out their K-L and J-S divergence:

$$KL(p, p_\theta) = \begin{cases} 0 & \theta = 0 \\ \infty & \theta \neq 0 \end{cases} \text{ and } JS(p, p_\theta) = \begin{cases} 0 & \theta = 0 \\ \log 2 & \theta \neq 0 \end{cases}$$

Ideally, we'd like a measure that is smoother. Wasserstein distance is exactly such a function defined as:

$$WS(p_1, p_2) = \inf_{\gamma \in \prod(p_1, p_2)} E_{(x,y)\sim\gamma} \|x - y\|_1$$

where $\prod(p_1, p_2)$ contains all joint distribution of $(x, y)$ such that $p_1(x) = \int \gamma(x, y)dy$ and $p_2(y) = \int \gamma(x, y)dx$. Wasserstein distance is also called earth-mover distance. It informally captures the minimal amount of mass/dirt needs to be moved to turn the shape of $p_1$ into that of $p_2$. Using this definition, the Wasserstain distance between $p(x)$ and $p_\theta(x)$ above can be calculated to be $|\theta|$, which is a continuous function of $\theta$.

In general, however, Wasserstein distance is intractable to calculate. Fortunately, Wasserstein distance has another definition, based on the Kantorovich-Rubinstein duality, which is easier to handle:

$$WS(p_1, p_2) = \sup_{\|f\|_L \leq 1} E_{x\sim p_1} f(x) - E_{x\sim p_2} f(x) \tag{5}$$

where $f(\cdot)$ is any real valued function. $\|f\|_L \leq 1$ means $f$'s Lipschitz constant is 1. Technically, it means

$$|f(x) - f(y)| \leq \|x - y\|_1 \; \forall x, y$$

Intuitively, it is equivalent to saying $f(x)$'s value should not change too much as $x$ changes.

Wasserstein distance (5) really captures the contrast maximization idea in Section 1.2 well. The Lipschitz continuity constraint can be approximated by the gradient penalty term introduced in 3.2.

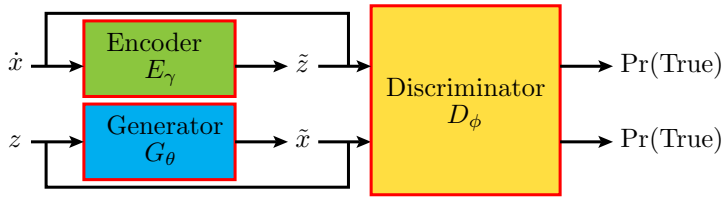


Figure 4: BiGAN architecture

### 3.3.3 GAN and WGAN

GAN's $D_\phi$ optimizes maximum likelihood of observing samples $\dot{x}$ and $\tilde{x}$. Given an optional $D_{\phi^*}$, $G_\theta$ minimizes the J-S divergence between $p_\theta(x)$ and $p(x)$.

In WGAN, $D_\phi$ maximizes the Wasserstein distance between $\dot{x}$ and $\tilde{x}$ while $G_\theta$ tries to reduce it. Wasserstein distance is a smoother and more effective measure of closeness between two distributions, resulting in more stable training and less mode collapse in WGAN than GAN.

## 4 Latent Representation and BiGAN

The training of Variational Autoencoders produces both a generator and a encoder. The latter is capable of extracting features or latent representations of data. GAN only has a generator $G_\theta$. It is conceivable that the pre-final layers of the discriminator $D_\phi$ may be used for feature representations. The intuition is that $D_\phi$ would have learned useful high-level representations of both real and artificial samples.

Bidirectional Generative Adversarial Networks (BiGAN), depicted in Figure 4, takes a much more direct approach to latent representation. It introduces an encoder $E_\gamma$ that maps real samples $\dot{x}$ to $\tilde{z}$ in the latent space. $D_\phi$ works on the joint distribution of $(x, z)$ and tries to maximize the contrast between real samples $(\dot{x}, \tilde{z})$ and artificial samples $(\tilde{x}, z)$. The loss function introduced in WGAN-GP can be used here. After training is done, new samples can be generated by $G_\theta$, and latent representations can be inferred via $E_\gamma$.

## 5 Image-to-image translation and CycleGAN

A GAN's generator maps samples drawn from $N(0, I)$ to meaningful images. Can GAN map images in one domain to ones in another domain, for example from summer pictures to winter pictures of the same place, from horse pictures to zebra pictures, and from photos to Van Gogh's paintings?

It is not hard to design a GAN for that purpose. For example, suppose our goal is to add black/white stripes to a horse and make it look like a zebra, we could replace $z \sim N(0, I)$ with a bunch of horse pictures, and the real samples will be drawn from zebra pictures. Unfortunately, this architecture does not ensure that the generated zebra picture will be much like the input horse picture. CycleGAN introduced two innovations to address this problem:

1. Create two GANs. The first GAN translates horse pictures to zebra pictures, and the second GAN translates zebra pictures to horse pictures.

2. Each generated zebra picture is then fed to the second GAN to translate to a horse picture. This generated horse picture should look similar to the original horse picture. A similar process happens with the reverse translation direction.

For simplicity, we'll use the following notation.

| Notation | Meaning |
|---|---|
| $x$ | real samples from domain $X$ |
| $y$ | real samples from domain $Y$ |
| $D_x$ | descriminator for samples in $X$ |
| $D_y$ | descriminator for samples in $Y$ |
| $G_{xy}$ | generator that maps samples in $X$ to samples in $Y$ |
| $G_{yx}$ | generator that maps samples in $Y$ to samples in $X$ |
| $L_{GAN}(X, Y, G_{xy}, D_y)$ | GAN's loss function that involves $G_{xy}$ and $D_y$ |
| $L_{GAN}(Y, X, G_{yx}, D_x)$ | GAN's loss function that involves $G_{yx}$ and $D_x$ |

CycleGAN's optimization objective is

$$L_{GAN}(X, Y, G_{xy}, D_y) + L_{GAN}(Y, X, G_{yx}, D_x) + \lambda E_x \left\| G_{yx}(G_{xy}(x)) - x \right\|_1 + \lambda E_y \left\| G_{xy}(G_{yx}(y)) - y \right\|_1$$