Generative Adversarial Networks

Muchang Bahng

Spring 2023

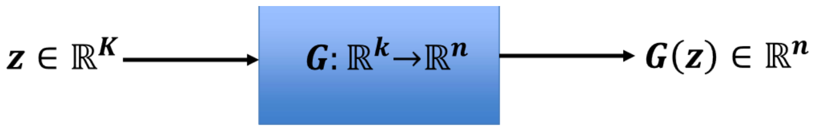
Contents

1 Generative Adversarial Networks

The year after Kigma published his paper on VAEs, Ian Goodfellow in 2014 published [GPAM+14] on *generative adversarial networks*. Unlike the current generative models, which relied on "fancy" methods like variational Bayes or contrastive divergence, Goodfellow claimed that we can just do this with vanilla MLPs. The general idea is to approximate the true density $p^*(x)$, you are trying to build a function (a neural network) G_α that is a transformation of a simple latent variable, e.g. $Z \sim \mathcal{N}(0, 1)$, such that the pdf of $G_\alpha(Z)$ is similar to $p^*(x)$.

Remember that for the generative models that we have so far, we were able to *explicitly model* the true pdf.

1. *Explicit and Analytical.* The simplest types of densities can be analytically written and can be sampled from directly (e.g. Gaussian with inverse CDF, Box-Muller transform, and even GMMs).
2. *Explicit and Approximate.* More complicated models such as the RBM and VAE cannot be analytically written, but can still be directly sampled from with probabilities. Boltzmann machines approximate the explicit density as $p_\theta(x) = \frac{1}{Z} e^{-E(x,z)}$ for some quadratic term $-E(x, z)$. VAEs construct another family of posterior distributions $\{q_\lambda\}$ where each λ is the output of an encoder neural network E_α. Later on, we will see that normalizing flows and autoregressive models also fall in this category.
3. *Implicit.* GANs approximate the true pdf through a transformation of random variables and are implicit in the way that you can't estimate the probabilities but you can still sample from them.[1]



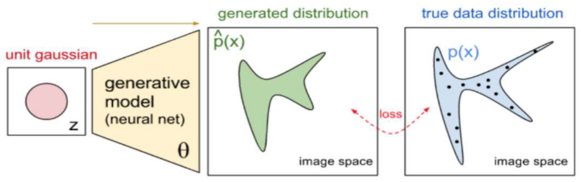


Figure 1: You essentially want to take some simple latent distribution and construct a differentiable a **generator model** that maps it to a more complicated distribution in the observable space.

The problem with VAEs is that they tend to generate blurry images, which is the result of optimizing a variational lower bound rather than the true objective. This makes it easy to identify whether a given image is from the true dataset or has been artificially generated. In contrast, GANs generate high-resolution images because we are directly optimizing the true objective.

Definition 1.1 (Generative Adversarial Networks)

A **generative adversarial network (GAN)** is a nonlinear latent-variable model consisting of by optimizing a pair of generator and disciminator neural networks, which play a game where one tries to bea the other.

1. A latent random variable Z with pdf $p(z)$.
2. The **generator** tries to generate fake samples to fool the discriminator. We sample from a latent space $\mathbf{z}$ and run that through the neural network to get $\mathbf{x} = \mathcal{G}_{\theta_g}(\mathbf{z})$.i It should be differentiable, but does not have to be invertible.
3. The **discriminator** tries to distinguish between real and fake samples, like a critic which can tell

[1]Given $Z \sim \mathcal{N}(0, 1)$ and the sine function $f(z) = \sin(z)$, it's hard to analytically compute the pdf of $X = f(Z)$. However, to sample from this, we can just sample z from Z and then transform it through f.

from real from fake. It should also be differentiable, and its output is essentially $0 \leq \mathcal{D}_{\theta_d}(\mathbf{x}) \leq 1$, with a value of 1 if real, 0 if fake.

We want to train these two models against each other, and in the end, we throw $\mathcal{D}$ away, since it's only role is to force $\mathcal{G}_{\theta_g}$ to work harder, which leaves us with a really good generative $\mathcal{D}_{\theta_g}$.

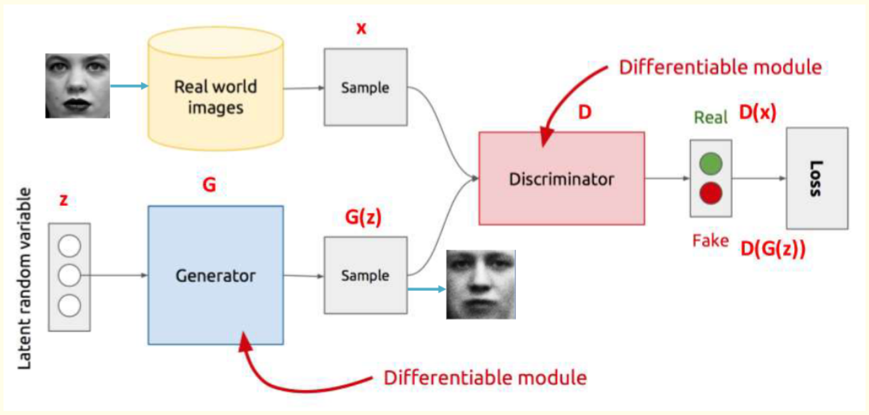


Figure 2

Definition 1.2 (GAN Loss)

Now each of these networks will have its own set of parameters which we have to optimize. We want to optimize them by maximizing the likelihood such that the model says “real" to the samples from the world and “fake" to the generated samples. This leads to

$$\mathcal{L}(\theta_d, \theta_g) = V(\mathcal{D}, \mathcal{G}) = \underbrace{\mathbb{E}_{x\sim real}\big[\log \mathcal{D}_{\theta_d}(\mathbf{x})\big]}_{\substack{\text{log-prob that } \mathcal{D} \text{ correctly} \\ \text{predicts real data as real}}} + \underbrace{\mathbb{E}_{\mathbf{z}}\big[\log\big(1 - \mathcal{D}_{\theta_d}(\mathcal{G}_{\theta_g}(\mathbf{z}))\big)\big]}_{\substack{log-probthat\mathcal{D} \text{ correctly} \\ \text{predicts generated data as fake}}} \tag{1}$$

Therefore, the discrimiator is trying to maximize its reward (to get max value of 0), and the generator is trying to minimize the discriminator's reward (pulling this log probability down to $-\infty$). This is known as a **minimax** optimization problem, and we have to find

$$\min_{\theta_g} \max_{\theta_d} V(\mathcal{D}_{\theta_d}, \mathcal{G}_{\theta_g}) \tag{2}$$

which is some saddle point.

Before we even talk about evaluating the gradients for this, let's prove a result on the properties of such an optimum.

Theorem 1.1 (Global Optimality)

Given the minimax loss above, for a fixed $\mathcal{G}$, the optimal discriminator $\mathcal{D}^*_G$ is given by

$$\mathcal{D}^*_G(\mathbf{x}) = \frac{p(\mathbf{x} \mid \text{real})}{p(\mathbf{x} \mid \text{real}) + p(\mathbf{x} \mid \text{synthesized})} \tag{3}$$

Therefore, the global minimum of the training criterion, $\max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G})$ is achieved if and only if

$$p(\mathbf{x} \mid \text{real}) = p(\mathbf{x} \mid \text{synthesized}) \tag{4}$$

Proof.

We first have

$$V(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{\mathbf{x}\sim real} \log \mathcal{D}_{\theta_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z}} \log\left(1 - \mathcal{D}_{\theta_d}(\mathcal{G}_{\theta_g}(\ddagger))\right) \tag{5}$$

Since $\mathcal{G}$ is fixed (i.e. θ_g is fixed) and acting, we can write the second expectation with respect to the probability measure induced by $\mathcal{G}$.

$$\begin{aligned} V(\mathcal{D}, \mathcal{G}) &= \mathbb{E}_{\mathbf{x}\sim real} \log \mathcal{D}_{\theta_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{w}\sim \text{fake}} \log\left(1 - \mathcal{D}_{\theta_d}(\mathbf{w})\right) \\ &= \int p(\mathbf{x} \mid \text{real}) \log \mathcal{D}_{\theta_d}(\mathbf{x})\, d\mathbf{x} + \int p(\mathbf{x} \mid \text{fake}) \log\left(1 - \mathcal{D}_{\theta_d}(\mathbf{x})\right) d\mathbf{x} \end{aligned}$$

where the $\mathbf{x}$ in the second integral is a dummy variable. Taking the derivative w.r.t. $\mathbf{x}$ and setting equal to 0 gives

$$p(\mathbf{x} \mid \text{real}) \frac{1}{\mathcal{D}_{\theta_d}(\mathbf{x})} + p(\mathbf{x} \mid \text{fake}) \cdot \frac{-1}{1 - \mathcal{D}_{\theta_d}(\mathbf{x})} = 0 \tag{6}$$

implies that

$$\mathcal{D}_{\theta_d}(\mathbf{x}) = \frac{p(\mathbf{x} \mid \text{real})}{p(\mathbf{x} \mid \text{real}) + p(\mathbf{x} \mid \text{fake})} \tag{7}$$

If the discriminator $\mathcal{D}$ is optimal, then the generator is minimizing the **Jensen-Shannon divergence** between the real and generated (model distributions). However, $\mathcal{D}$ is not optimal in practice since we have limited computational resources, the loss is non-convex, etc.

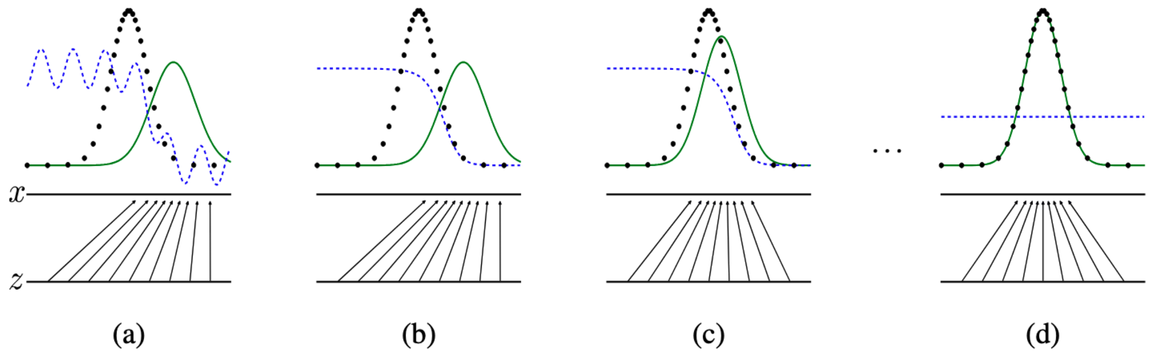


Figure 3

Now we can talk about evaluating the gradients and optimizing it. What is the gradient of such a loss function? This is quite easy since the expectations are over the true data distribution and the known prior z, and so we can swap gradients and expectations accordingly.

Lemma 1.1 (Gradients of GAN Loss)

Proof.

Note that the generator has no effect on the probability of $\mathcal{D}$ correctly identifying real images, so it only focuses on the latter term.

Algorithm 1.1 (Alternative Gradient Descent)

The idea is to train both models simultaneously via SGD using mini-batches consisting of some generated samples and some real-world samples, which is called the **alternative gradient descent algorithm**. It is usually the case that the generator does better than the disciminator, so we sometimes make $k > 1$ to allow more steps for training.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
1: for number of training iterations do
2:     for $k$ steps do
3:         Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
4:         Sample minibatch of $m$ examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
5:         Update the discriminator by ascending its stochastic gradient:
               $$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$$
6:     end for
7:     Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
8:     Update the generator by descending its stochastic gradient:
           $$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$$
9: end for
```

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

There is a vanishing gradient problem in GANs. For instance, assume that the model $\mathcal{G}_{\theta_g}$ has very bad parameters, and it generates very bad samples that the disciminator can detect very well. Then, $\mathbb{E}_{\mathbf{z}}\big[\log\big(1 - \mathcal{D}_{\theta_d}(\mathcal{G}_{\theta_g}))\big)\big]$ will be very close to 0, and the generator's cost will be very flat. Therefore, the gradient would die out and the generator can't improve! For example, if $\mathcal{D}$ was just a sigmoid function, then we can approximate the gradient of the expectation with a sample of the gradient, which would die out.

$$\begin{aligned}
\nabla_{\theta_g} V(\mathcal{D}, \mathcal{G}) &= \nabla_{\theta_g} \mathbb{E}_{z \sim q((\mathbf{z})}\big[\log\big(1 - \mathcal{D}(\mathcal{G}(\mathbf{z}))\big)\big] \\
&\approx \nabla_a \log\big(1 - \sigma(a)\big) \\
&= -\frac{\sigma(a)\big(1 - \sigma(a)\big)}{1 - \sigma(a)} = -\sigma(a) = -\mathcal{D}(\mathcal{G}(\mathbf{z}))
\end{aligned}$$

and so the gradient goes to 0 if $\mathcal{D}$ is confident, i.e. $\mathcal{D}(\mathcal{G}(\mathbf{z})) \to 0$. Therefore, we can modify the cost for the generator term by changing the cost to

$$\mathbb{E}_{\mathbf{z}} \log\big(1 - \mathcal{D}_{\theta_d}(\mathcal{G}_{\theta_g}(\mathbf{z}))\big) \tag{8}$$

and trying to minimize it.

Theorem 1.2 (Convergence)

Algorithm 1.2 ()

An implementation of a GAN in PyTorch is here.

1.1 Conditional GAN

1.2 CycleGAN

1.3 Deep Convolution GAN (DCGAN)

References

[GPAM+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.