Introduction to GAN

DATE: 2020-07-18

Scribe: Yuhao Wu

GAN(Generative Adversarial Network) is first raised by Ian Goodfellow in his paper "Generative Adversarial Network" [2]. Of course, from its name, we know that GAN is to generate something. One of its greatest application is in "Face Generation". In the end of 2019, NVIDIA published their "StyleGAN2" [3](*code available*), which is to produce fake face picture with resolution 1024×1024. GAN is also highly recognized by Yann LeCun. He said "Adversarial training is the coolest thing since sliced bread".

1 Basic Structure

For GAN, it should have one "Generator" and one "Discriminator", the basic structure is shown in figure 1 below.

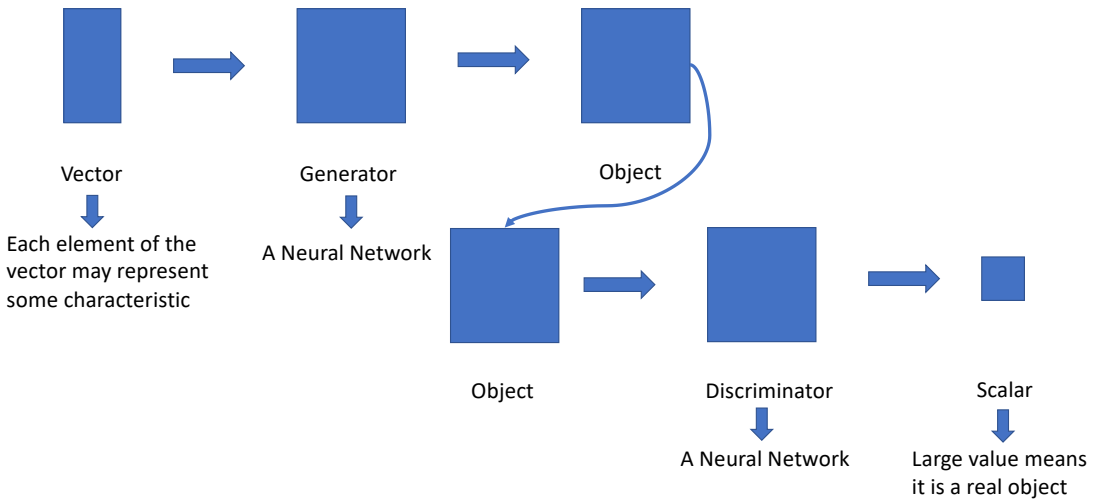

Figure 1: Basic structure of GAN

For the object, it can be one picture, or a 3D mesh, or maybe a sentence. According to what you want, you may need to do some change on the output layer of "Generator". If you want a sentence, you may need a RNN; if you want a picture, you may add one deconvolution.

For the vector, we say each element of the vector may represent some characteristic. We assume that our output is a manifold in a higher dimension. So, the vector is the best for data with the same dimension. *But how do we know how many dimension we need. We have no idea. Looks like it is a hyper-parameter.*

2 Theoretical Result of GAN

2.1 Value Function

- $G(z;\theta_g)$ is a differentiable function represented by a multilayer perceptron with parameters θ_g. Its input is one latent space variable z.
- $D(x;\theta_d)$ is a differentiable function represented by a multilayer perceptron with parameters θ_d.

What we want to optimize is the following value function:

$$\min_G \max_D V(D,G) = \min_G \max_D \ \mathbf{E}_{x\sim p_{data}(x)}[\log(D(x))] + \mathbf{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]. \tag{1}$$

Since it is binary classification under this "Discriminator", we have the following cross entropy:

$$H((x,y),D) = -\Big[y\log(D(x)) + (1-y)\log(1-D(x_1))\Big] \tag{2}$$

For infinite data point we have:

$$H((x,y)_{i=0}^{\infty},D) = -\sum_{i=0}^{\infty}\Big[y\log(D(x)) + (1-y)\log(1-D(x_1))\Big]. \tag{3}$$

For each x_i, it is either from the true data, which should satisfy $x \sim P_{data}(x)$; or it is from the "Generator", which should satisfy $x = G(z), z \sim P_Z(z)$. We separate the datas apart. If it is from the real data, it would zero out the latter part of Equation(3); if it is from the "Generator", it would zero out the first part of Equation(3). When we are doing training, we would provide half of the data from "Generator", half from real data. So, the $P(y=1) = P(y=0) = \frac{1}{2}$. Thus, we have:

$$H((x,y)_{i=0}^{\infty},D) = -\frac{1}{2}\Big[\mathbf{E}_{x\sim p_{data}(x)}[\log(D(x))] + \mathbf{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]\Big]. \tag{4}$$

The above statement comes from Ian's 2016 tutorial [1] section 3.2 "Cost Function". If you have time, could you please check if I understand them right.

It needs to minimize the maximum value of $V(D,G)$. Intuitively speaking, it is really easy to understand: we would train the "Discriminator" first, which is trying to make the "Discriminator" give real images scores as high as possible. Then, we need to train the "Generator", which is trying to make the "Generator" produce fake images to confuse the "Discriminator" so that "Discriminator" would give real image low scores.

2.2 Convergence of the Algorithm

The "Generator" generates data $G(z)$ when $z \sim P_z$. We can say that the generator G implicitly defines a probability distribution p_g. According to [2], Section 4, Ian has proved that Algorithm1 would converge p_g to a good estimator of p_{data}, if given enough capacity and training time.

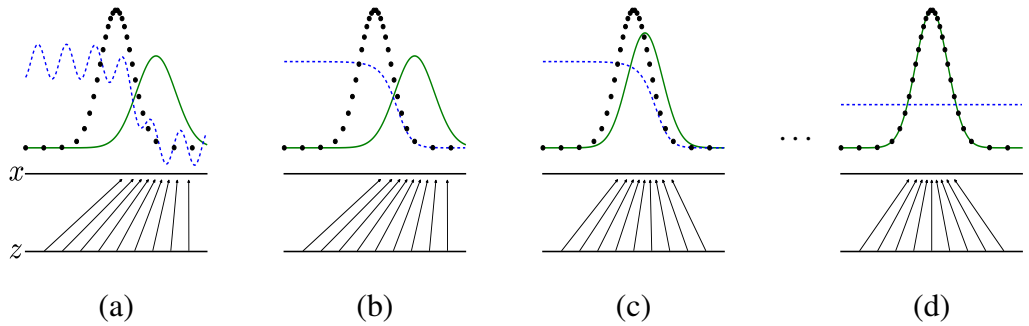

Figure 2: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

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.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

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

References

[1] I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[3] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.