University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 21: November 6

Lecturer: Alexander Schwing Scribe: Kuocheng Wang

1 Review

Recall the maximum likelyhood function we used for discriminative model :

$$p(y|x) = \frac{\frac{e^{F(y,x,w)}}{\epsilon}}{\frac{\sum_{\hat{y}} e^{F(\hat{y}\ ,x,w)}}{\epsilon}} \tag{1}$$

y is the output space, x is the input data. Descrimative model is trying to predict a label based on the input data. The label can be a class (classification), or can be a number (regression). Equation 1 is for classification. However, an actual artificial intelligent system should be able to do more than just predicting labels. Another interesting aspect we have recently learned is the generative model. In generative model, there is no label. Instead, we are interested in modeling p(x). Basically, instead of using model to learn the labels, the formulation allows us to generate data based on the data the system have seen. It is obvious that classification is in general an easier task.
There are several applications of this. For example, we can do synthetic of object (images, text), environment simulator (reinforcement learning, planning), and help leverage unlabeled data. But how do we model p(x)? From what we have learned before, maximum likelihood is generally used for generative model

$$\theta^\star = \sum_i \log p(x^i; \theta) \tag{2}$$

Here θ is introduced as parameters to fit the mean and variance of the distribution, such as Gaussian distribution or mixture of Gaussian. We mentioned this idea in Variation auto-encoder. As a recap, the auto-encoder structure is shown in Figure 1. z is introduced as a latent variable, sometimes is referred to as noise. Given z, we can generate the distribution of x with $P(Z|\theta)$. In order to map z to x, audo-encoder uses original sample x to generate Z. This part is called encoder, and then we can generate x based on Z, which is decoder.

The loss functions is

$$L(p_\theta, q_\phi) \approx -D_{KL}(q_\phi, p) + 1/N \sum_{i=1}^{N} \log p_\theta(x|z^i) \tag{3}$$

One of the issues what had to address is how to compute the the normalized constant, which is solved by using an encoder. For more information, please review Lecture 20.

2 Generative adversarial network

This scribe is based on [5]. [4], [1] are two reading materials that are helpful to understand GAN better. The main idea behind Generative adversarial network (GAN) is instead of writing a formula for p(x), we will learn to sample directly. The advatage is it avoid the summation over large probability spaces $p(z|x)$.

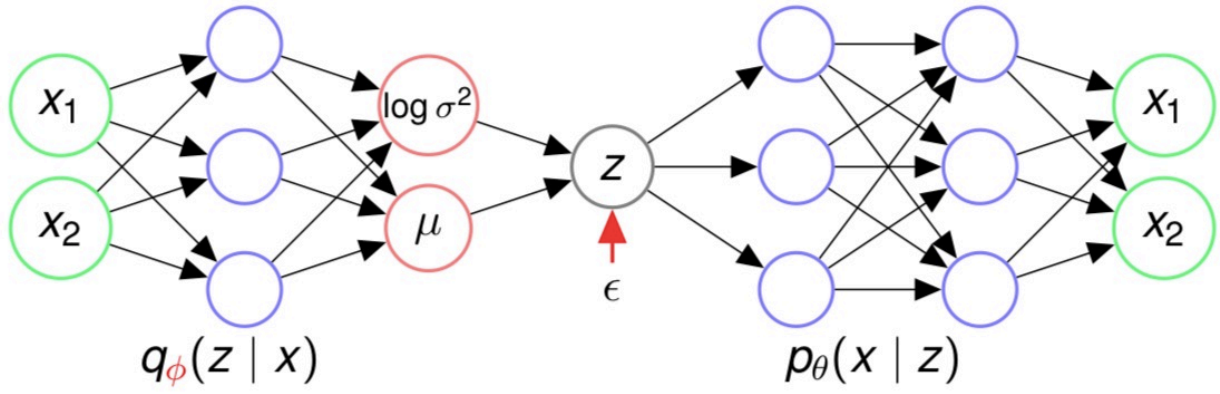

Figure 1: Auto-encoder.

2.1 Formulation

GAN formulates the problem as a game between two players, Generator G and Discriminator D. The task of the players are G generates examples and D predicts whether the example is artificial or real. G tries to "trick" D by generating samples that are hard to distinguish. Figure 2 shows the process in more detail. The left part shows the discriminator with input data x that samples from real data. D is a neural network, which can be as simple as a CNN that tries to predict x as probability of 1 (real image). The right hand side takes z as the input sample, which many times come from noise, after going through a generator network, and then feed into the discriminator. The discriminator is trying to predict whatever genrator feeds in as probability of 0 (fake image).

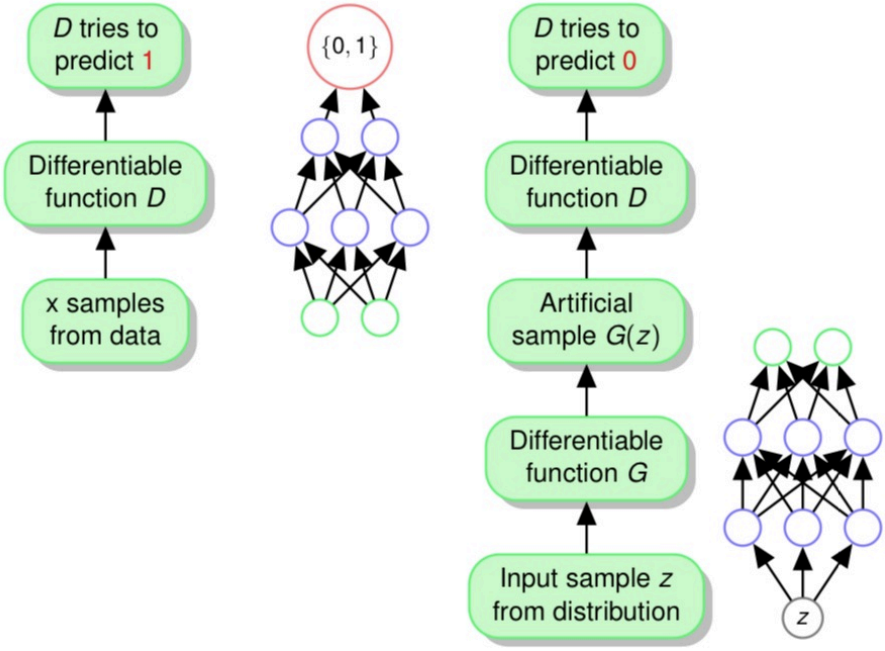

Figure 2: Auto-encoder.

Define generator as $G_\theta(z)$ and descriminator as $D_w(x) = p(y = 1|x)$. Equation (4) is how we choose w and (5) is how we choose θ.

$$\min_{w} -\sum_{x} \log D_w(x) - \sum_{z} \log(1 - D_w(G_\theta(z))) \tag{4}$$

$$\max_{\theta} \min_{w} -\sum_{x} \log D_w(x) - \sum_{z} \log(1 - D_w(G_\theta(z))) \tag{5}$$

For discriminator, since $G_\theta(z)$ generates fake images, $D_w(G_\theta(z))$ will label it as 0, and for x which is real image, $D_w(x)$ will output 1. For generator, it tries to trick discriminator such that $D_w(G_\theta(z))$ becomes as close to 1 as possible. Since equation 5 contains equation 4, we use equation 5 as the objective function for GAN. A general algorithm to solve this problem using gradient descent is shown in Figure 3.

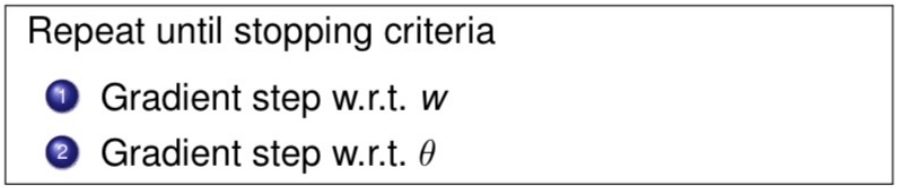

Figure 3: Auto-encoder.

In practice, there are heuristics that can make this algorithm more stable.

2.2 GAN's formulation with Euler-Lagrangian

Assume there is arbitrary capacity for the data, the otpitimal desciminator is

$$\min_{D} -\int_{x} p_{data}(x) \log D(x) dx - \int_{z} p_z(z) \log(1 - D(G_\theta(z))) dz \tag{6}$$

Define $p_G(x)$ as the probability distribution induced by the generator on x space (which is generated by z), the above equation becomes

$$\min_{D} -\int_{x} p_{data}(x) \log D(x) + P_G(x) \log(1 - D(x)) dx \tag{7}$$

Euler-Lagrange is an energy based method which tries to solve the stationary point of a dynamic system. Equation (6) can be treated as a dynamic system equation, Hence Euler-Lagrange formulation is used to solve for the stationary point of the equation. Equation 8 is the Euler-Lagrange formulation,

$$S(D) = -\int_{x} L(x, D, \dot{D}) dx \tag{8}$$

The stationary point D is the solution to the following equation

$$\frac{\partial L(x, D, \dot{D})}{\partial D} - \frac{d}{dx} \frac{\partial L(x, D, \dot{D})}{\partial \dot{D}} = 0 \tag{9}$$

We don't have $\dot{D}$ in this case, equantion 9 becomes

$$\frac{\partial L(x, D, \dot{D})}{\partial D} = \frac{-p_{data}}{D} + \frac{p_G}{1 - D} = 0 \tag{10}$$

Hence,

$$D^{\star}(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \tag{11}$$

Plug $D^{\star}(x)$ back into equation 7

$$\begin{aligned}
&- \int_x [p_{data}(x) \log D^{\star}(x) + P_G(x) \log(1 - D^{\star}(x))] dx \\
&= - \int_x [p_{data}(x) \log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} + P_G(x) \log(1 - \frac{p_{data}(x)}{p_{data}(x) + p_G(x)})] dx \\
&= - \int_x [p_{data}(x) \log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} + P_G(x) \log(\frac{p_G(x)}{p_{data}(x) + p_G(x)})] dx
\end{aligned} \tag{12}$$

Recall that KL divergence

$$KL(p, q) = p(x) \int log(p(x)/q(x)) dx \tag{13}$$

Let's define

$$M = \frac{1}{2}(p_{data} + p_G) \tag{14}$$

Define

$$\begin{aligned}
JSD(p_{data}, p_G) &= \frac{1}{2} KL(p_{data}, M) + \frac{1}{2} KL(p_G, M) \\
&= \frac{1}{2} \int_x p_{data}(x) \log \frac{p_{data}(x)}{M} dx + 1/2 \int_x p_G(x) \log \frac{p_G(x)}{M} dx \\
&= \frac{1}{2} \int_x [p_{data}(x) \log \frac{p_{data}(x)}{\frac{1}{2}(p_{data} + p_G)} + p_G(x) \log \frac{p_G(x)}{\frac{1}{2}(p_{data} + p_G)}] dx \\
&= \frac{1}{2} \int_x [p_{data}(x) \log \frac{2p_{data}}{(p_{data} + p_G)} + p_G(x) \log \frac{2p_G(x)}{(p_{data} + p_G)}] dx \\
&= \frac{1}{2} \int_x p_{data}(x) \log 2 + p_{data}(x) \log \frac{p_{data}}{(p_{data} + p_G)} + p_G(x) \log 2 + p_G(x) \log \frac{p_G}{(p_{data} + p_G)} dx \\
&= \frac{1}{2} \log 2 \int_x (p_{data}(x) + p_G(x)) dx + \int_x p_{data}(x) \log \frac{p_{data}}{(p_{data} + p_G)} + p_G(x) \log \frac{p_G}{(p_{data} + p_G)} dx \\
&= \log 2 + \frac{1}{2} \int_x p_{data}(x) \log \frac{p_{data}}{(p_{data} + p_G)} + p_G(x) \log \frac{p_G}{(p_{data} + p_G)} dx
\end{aligned} \tag{15}$$

Notice, in the above equation,

$$\int_x (p_{data}(x)) dx = 1 \tag{16}$$

The same is true for $p_G(x)$ because both of these are probability distributions.
Equation 12 can be rewritten as

$$-2JSD(p_{data}, p_G) + \log 4 \tag{17}$$

When $p_{data}(x) = p_G(x)$, $JSD(p_{data}, p_G) = 0$, we will achieve the optimal discriminator.

3 Heuristics to improve GAN

3.1 Wasserstein distance

Recall the objective function of GAN in equation 5. If G is very bad and D is very good, the maximization with respect to θ is very small, which makes it hard to train. The loss function is hence changed to only train generator, which is shown in Equation 18. Figure 4 shows the loss function with respect to $G_\theta(z)$. The modified loss funtion has a much larger gradient than the original loss function. One of the issues of this formulation is it has no joint cost function for D and G.

$$\min_{\theta} -\sum_{z} \log(D_w(G_\theta(z))) \tag{18}$$

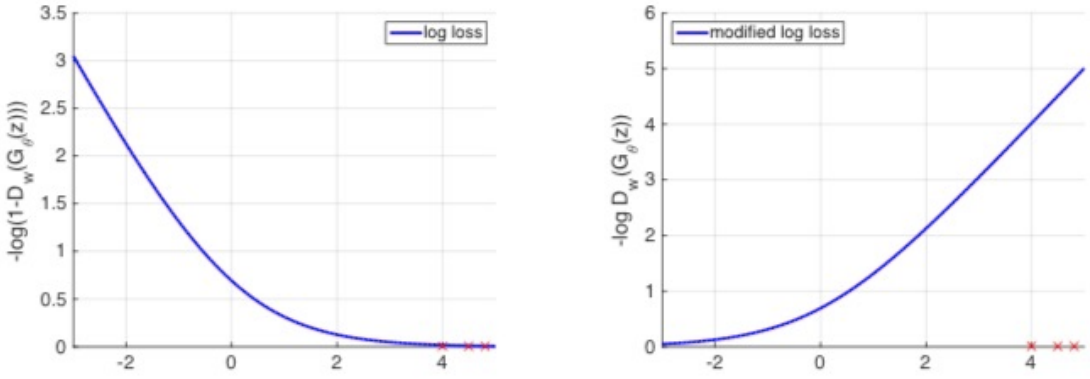

Figure 4: loss.

The orginal GANs were very unstable to train and frequently experienced mode collapse. There is pretty much no successful training example of large models and it is very sensitive to hyperparameters and architecture. There are also plenty of tricks to try to overcome these issues. Wasserstein GAN is one of them. Equation 19 is the Wasserstein distance. It is used to compute the distance between 2 distributions. Figure 5 shows a pictorial representation of Wasserstein distance.

$$W(p_{data}, p_G) = \min_{p_J(x,x') \epsilon \pi(p_{data}, p_G)} E_{p_J}[||x - x'||] \tag{19}$$

Equation 20 is the formulation of WasserStein GAN

$$\min_{p_G} W(p_{data}, p_G) = \min_{p_G} \min_{p_J(x,x') \epsilon \pi(p_{data}, p_G)} E_{p_J}[||x - x'||] \tag{20}$$

What are the issues with this formulation? First, p_{data} is not available; second, how can we computer the joint distribution p_j? The proposed solution is Kantorovich-Rubinsteinduality.

$$W(p_{data}, p_G) = \max_{||f||_L \leq 1} E_{p_{data}}[f(x)] - E_{p_G}[f(x')] = \min_{p_G} \max_{w} E_{p_{data}}[f_w(x)] - E_{p_G}[f_w(x')] \tag{21}$$

Figure 6 is the Wasserstein GAN from [2]. The paper is published in 2017, which is very recent. Anyone who is interested in the formulation is encouraged to read more about it.

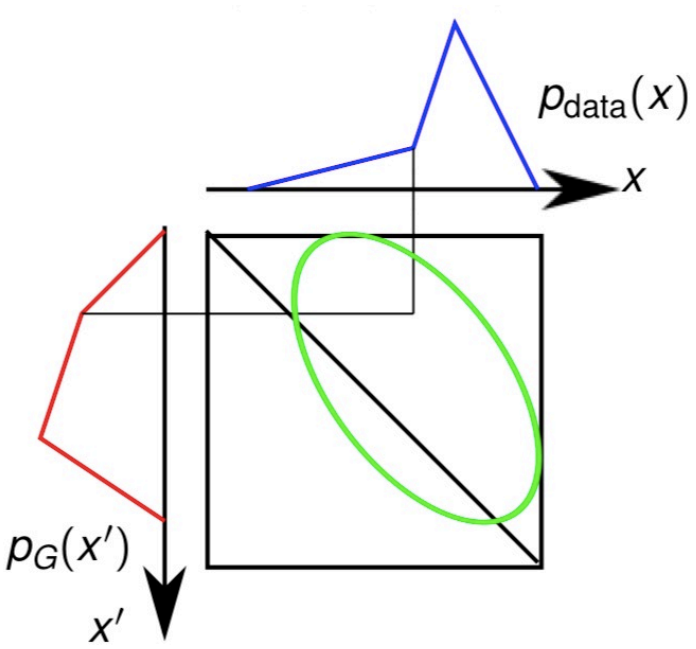

Figure 5: Wasserstein distance.

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 priors.
    $g_w \leftarrow \nabla_w [\frac{1}{m} \sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  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

Figure 6: Wasserstein GAN's algorithm.

3.2 Layer normalization

Layer normalization attempted to address certain issues related to applying batch normalization to recurrent neural networks and to remove batch dependencies between the input and the output. This batch dependency makes the gradient penalty invalid [3] . In batch normalization, the features of a single sample are normalized based on all of the other features in that sample without any information about other samples in the batch. The output from layer normalization is the same during training and testing.

4 Result

Figure 7 shows how we can get multiple mode data generator with Wasserstein GAN.

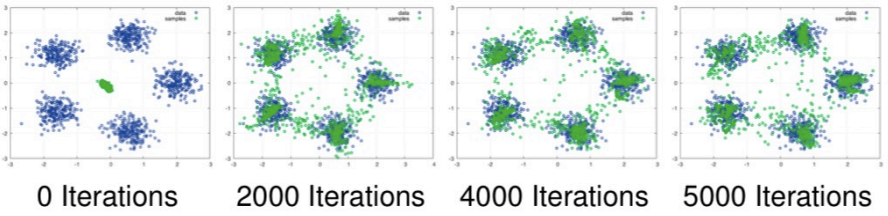

Figure 7: mode.

Figure 8 shows the fake bed, Figure 9 shows the generated flowers and their captions. One other problem with GAN is the generated images are blurry. This can be seen in the generated flowers. It is one of the research topics researchers are interested in.

Figure 8: bed.

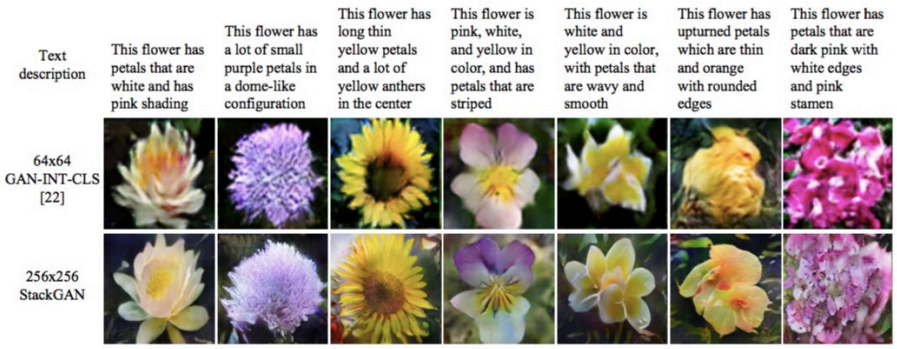

Figure 4. Example results by our proposed StackGAN and GAN-INT-CLS [22] conditioned on text descriptions from Oxford-102 test set.

Figure 9: flower.

5 Quiz

1. What generative modeling techniques do you know about?
   The generative modeling techniques we have learned so far is k-Means, Gaussian Mixture Model, Hidden Markov model, Variational Auto-Encoders.

2. What are the short-comings of those techniques?
   k-Means need to choose K. It sensitive to outliers and it can get stuck in local minima.
   Gaussian Mixture Model need to choose how many Gaussians in the beginning. The EM algorithm used to solve the optimization problem is only guaranteed to converge to a point with zero gradient with respect to the parameters. It sometimes get stuck in saddle points.
   Hidden Markov model, most of time, refers to Viterbi algorithm, is both computationally and memory expensive [6].

3. What differentiates GANs from other generative models?
   GAN does not write a formula for p(x), it just learn to sample directly.

4. What are the short-comings of GANs
   GANs have problems in mode collapse. It takes a long time to train and the result may be almost the same as the original image. The generated images are blurry, which is hard to see sometimes. Many techniques such as Wasserstein distance or layer batch normalization are applied to make it work better.

References

[1] Arjovsky. *Wasserstein gan.* arXiv preprint, 2017.

[2] M. Arjovsky, S. Chintala, and L. Bottou. *Wasserstein generative adversarial networks.* 2017.

[3] L. Courtney. *Deep Learning.* 2018.

[4] I. Goodfellow. *Generative Adversarial Networks.* Advances in neural information processing systems (pp. 2672-2680), 2014.

[5] A. G. Schwing. *Pattern Recognition.* 2018.

[6] H. Tanaka. *Hidden Markov Models and Iterative Aligners: Study of their Equivalence and Possibilities).* Proceedings. International Conference on Intelligent Systems for Molecular Biology, 1993.