University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 21: November 6

Lecturer: Alexander Schwing Scribe: Zhichun Wan

1 Outline

Generative Adversarial Network is a generative model that simultaneously train a generator and discriminator by putting these two models in a game setting [1]. This lecture includes a recap of previous lectures and focuses on the topic of Generative Adversarial Network(GAN). We'll draw relations between GAN and generative models we talked about in the past. The goals for this lecture includes:

- Learning Generative Adversarial Network
- Understanding generative model
- Differentiating between generative and differentiative models

2 Recap

2.1 Maximum Likelihood(ML)

For most of the lectures we talked about Maximum Likelihood model.

- In discriminative case, we model $p(y|x)$
- In generative case, we model $p(x)$

In all cases, we model some parameters, w or θ

To find the parameters, we set up an optimization problem and try to optimize the parameters. The optimization problem tries to maximize the log likelihood or minimize the negative log likelihood.

Typical parameters:

- Discriminative case:
  1. Linear regression – slope
  2. Logistic regression or SVM – separating hyperplanes
  3. Deep nets – convolution, fully connected layers etc.
- Generative case:
  1. K-means – just means
  2. Gaussian – means and standard deviations
  3. Gaussion Mixture Models – means, and standard deviations and mixture components π.

The general form of maximum likelihood model for discriminative case:

$$p(\mathbf{y}|x) = \frac{expF(y, x, w)/\epsilon}{\sum_{\hat{y}} expF(\hat{y}, x, w)/\epsilon} \tag{1}$$

For generative case, we model $p(x)$. Why model distribution?

- To synthesize the objects (images, texts)
- Environment simulators
- Leveraging unlabeled data

Given data points x, how can we model $p(x)$? Maximum likelihood.

$$\theta^* = \max_{\theta} \sum_i \log p(x^{(i)}, \theta) \tag{2}$$

- In first case, try a simplistic model, p being a Gaussian. θ is then mean and variance. We just need to fit the mean and variance.
- Second case, Gaussian Mixture Model, θ being parameters for mixture distribution. We started to see the concept of alternating optimization (Expectation-maximization)
- Third case, Variational Auto-encoder. Key difference is that we cannot explicitly compute posterior anymore – approximate with deep nets.

2.2 Variational Auto-Encoder

Auto-encoders takes data x and codes it into latent space z and then reconstruct the data. Below is a picture for the variational auto-encoder.

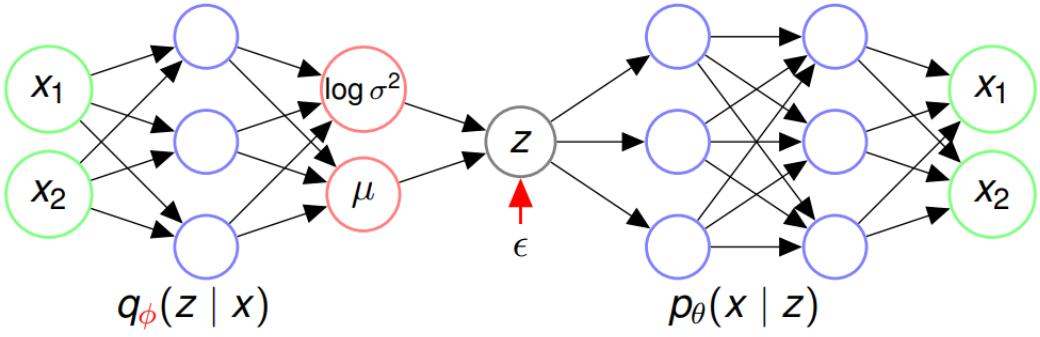


Figure 1: Variational Auto Encoder

Decoder distribution $p_\theta(x|z)$
Loss function:

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

Issue: intractable to compute because of the normalization constant – approximate it with $q_\phi(z|x)$ which is now parametric.

- Why do we use $\log \sigma^2$? It requires no constraint, while ensures sigma is positive.
- Re-parametrization trick: instead of drawing a sample from distribution with μ and σ^2 , we sample from standard normal $\mathcal{N}(0, 1)$. Then $z = \mu + \sigma\epsilon$. In this way we don't need to back propagate through the sampling process.

Benefits of the VAE: we can draw sample from very simple distribution and plug into decoder to get sample for x.

3 Lecture Topic: Generative Adversarial Network

3.1 Intro

Generative Adversarial Network came from about the same time as VAE and has the same idea – we want to take a sample from simple distribution and transform it to more complex distribution. The main idea with GAN is that we don't write a formula for $p(x)$, but learn to sample directly. The advantage with this approach is that we don't need summation over large probability spaces any more.

3.2 Problem formulation

In GAN, we formulate the problem as a game between two players – a generator G and a discriminator D.

The task of the players

- G generates examples
- D predicts whether example is real or artificial

The process

1. Take a sample from simple distribution (e.g. Gaussian).
2. Feed the sample to deep net which performs transformation like up-convolution to generate fake data
3. Take both real and fake data and feed to a differentiable function which is the discriminator.
4. Discriminator tries to predict whether the sample is real or fake

The above process is shown in figure 2 below. The top part is essentially training a binary classifier – maximize log likelihood

Mathematical formulation

- Generator $G_\theta(z)$
- Discriminator $D_w(x) = p(y = 1|x)$

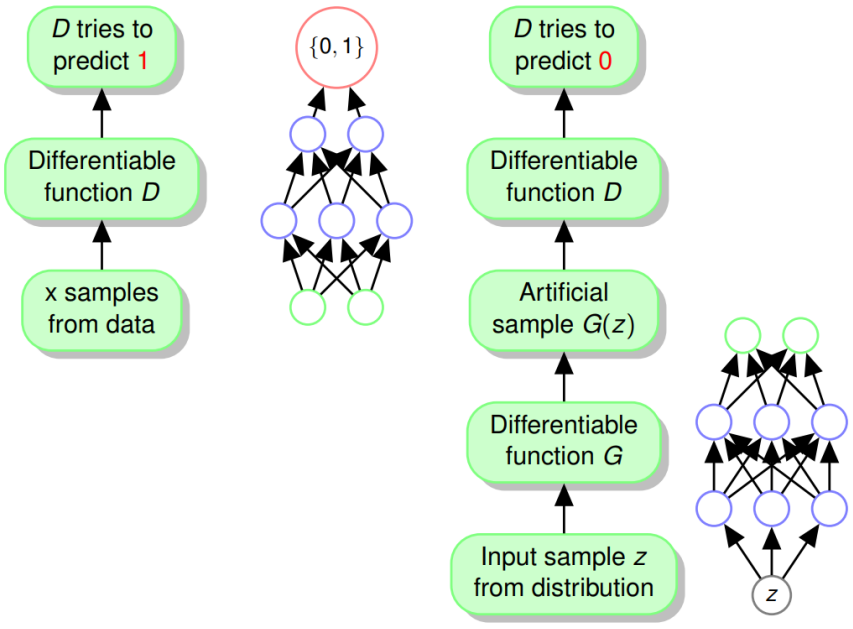


Figure 2: Pictorial explanation of GAN

How to choose w? We want to minimize the negative log likelihood for discriminator:

$$\min_w -\sum_x \log D_w(x) - \sum_z \log(1 - D_w(G_\theta(z))) \tag{4}$$

Note that the objective functions comprise of two parts, the first part corresponds to the real data and second part corresponds to the artificial data from generator.

How to choose theta? we want the generator to make the task of discriminator as hard as possible, so:

$$\max_\theta \min_w -\sum_x \log D_w(x) - \sum_z \log(1 - D_w(G_\theta(z))) \tag{5}$$

The above function is the joint objective for GAN. The discriminator tries to separate real data from fake data. The generator is trying to do the opposite to "trick" the discriminator.

Optimization

- Theoretically, repeat until stopping criteria
- Gradient Descent w.r.t w and w.r.t θ
- Problem: scaling to high-dimensional data is hard as we are trying to find a saddle point in the objective (the max and min in the objective function).
- In practice, heuristics make this optimization more stable

3.3 Analysis

Why are we guaranteed to learn the correct distribution? We'll make some simplifying assumptions.

1st assumption: Discriminator has arbitrary capacity, meaning it can model anything.
2nd assumption: We are in continuous space.

Proof:
Firstly compute the expectation for optimal discriminator:

$$\min_D : -\int_x p_{data}(x)\log D(x)dx - \int_z p_z(z)\log(1-D(G_\theta(z)))dz$$

$$= -\int_x p_{data}(x)\log D(x) + p_G(x)log(1-D(x))dx \tag{6}$$

$p_G(x)$ is distribution induced by Generator in x space. We replaced the generator by this $p_G(x)$. Take Euler-Lagrange formalism:

$$S(D) = \int_x L(x, D, \dot{D})dx \tag{7}$$

Functions with this form have stationary points at:

$$\frac{\partial L(x,D,\dot{D})}{\partial D} - \frac{d}{dx}\frac{\partial L(x,D,\dot{D})}{\partial \dot{D}} = 0 \tag{8}$$

There is no derivative in function(5), so stationary D becomes:

$$\frac{\partial L(x,D,\dot{D})}{\partial D} = -\frac{p_{data}}{D} + \frac{p_G}{1-D} = 0 \tag{9}$$

Solving for above equation, we have optimal discriminator:

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \tag{10}$$

Now with the optimal discriminator, we want to get the optimal generator. Plug D* into the original objective function:

$$-\int_x p_{data}(x)\log D^*(x) + p_G(x)log(1-D^*(x))dx$$

$$= -\int_x p_{data}(x)\log\frac{p_{data}(x)}{p_{data}(x)+p_G(x)} + p_G(x)log(\frac{p_G(x)}{p_{data}(x)+p_G(x)})dx$$

$$= -2JSD(p_{data}, p_G) + log(4) \tag{11}$$

JSD is the Jensen-Shannon divergence:

$$JSD(p_{data}, p_G) = \frac{1}{2}KL(p_{data}, M) + \frac{1}{2}KL(p_G, M), M = \frac{1}{2}(p_{data} + p_G) \tag{12}$$

KL is minimum when both distribution agree. And so is JSD minimum when the distributions agree. Consequently, the optimal generator is at:

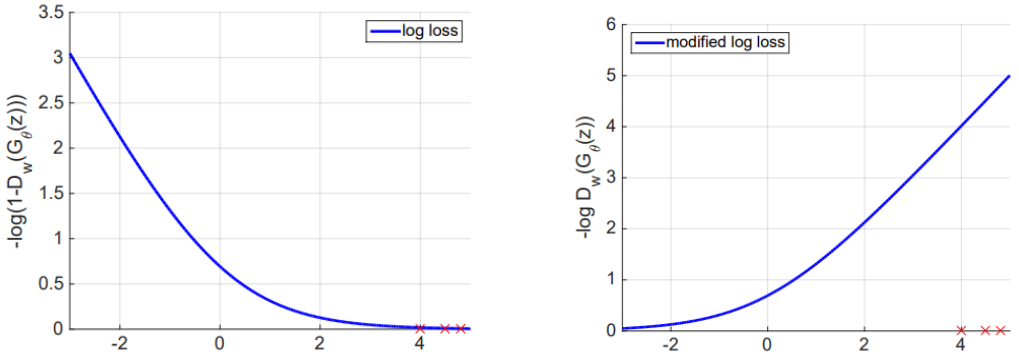
$$p_G(x) = p_{data}(x) \tag{13}$$

This means if we have an optimal discriminator and optimal generator given an arbitrary capacity, then the sample from generator would have same distribution as of the real data.

3.4 GAN in practice

Solving GAN optimization problem is non-trivial because of the saddle point.

Issue 1: If G is very bad and D is very good, then there is almost no gradient – hard to find a good initialization for the generator. This is a very likely situation in real training.



(a) Original loss curve

(b) Modified loss curve

Figure 3: Loss curves for GAN. Note the x axis is the logits - yF

As shown in the Figure 3(a), if D is really good, the loss would be very small, which means all the points will be at the right side of the curve, which are shown as the red dots.
Maximization of the loss function with respect to θ is same as asking the program to push the points leftwards along the curve. However the gradients at these points are really small and may not move at all. To think it in another way, every time the generator makes some progress, it will be rejected by the discriminator.

The workaround is to solve another loss function for the generator:

$$\min_{\theta} -\sum_{z} logD_w(G_\theta(z)) \tag{14}$$

This flips the loss curve around, as shown in the curve in Figure 3(b). Now the gradients are much stronger and we will be able to push the points left.
This workaround does have its drawback. By minimizing a separate loss function for the generator, we lost the joint objective for D and G. And for theoretic analysis, it becomes pointless to analyze the two separate loss functions.
However in practice, this works very well.

Issue 2: Mode collapse: if we are not careful in training, the generator may not capture the whole distribution and only outputs a certain mode. This issue could be improved by using more careful training.

Issue 3: Evaluation: the criteria for judging performance becomes very subjective.

Additional tricks:

- Use gradient penalties instead of JSD
- Don't put batch-norm in to discriminator as that may convoluted the real and fake data

3.5 Examples and Applications

3.5.1 Example 1 – training GAN visualization

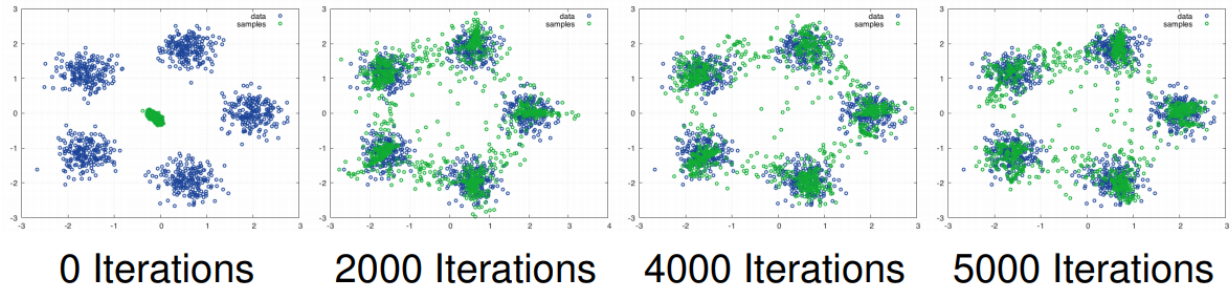


Figure 4: Example of GAN training

Above is a toy example. the generator starts from simple 2D Gaussians, as shown by the green dots. The blue does represent the distribution of real data. As we see, as training iterations go up, the green dots gradually move to the locations of the blue dots. As iterations go on, the green dots will eventually move to the same distribution as the blue dots.

3.5.2 Example 2 – Application: modeling images

Figure 5: Modeling images using GAN

3.5.3 Example 3 – Application: modeling images

Figure 6 is a set of bedroom images generated using Deep Convolutional Generative Adversarial Networks(DCGAN) [2]

3.5.4 Example 4 – Generate images from text

Figure 7 is a set images generated from text descriptions using stack-GAN. [3]

Figure 6: Bedroom images generated using DCGAN

Figure 7: Generate images from text

3.6 Wrap-up

Generative models so far:

- K-means
- Gaussian
- Gaussian Mixture Model
- Variational Auto Encoders
- Hidden Markov Models
- GAN

Differentiating factor between GAN and others: no empirical lower bounds
Shortcomings of GAN: mode collapse, hard to train, compute resources, etc

3.7 Extention: Wasserstein GAN

The idea is to not use the JSD, instead Wasserstein distance:

$$W(p_{data}, p_G) = \min_{p_J(x,x') \in \prod(p_{data}, p_G)} \mathbb{E}_{p_J}[||x - x'||] \tag{15}$$

How to formulate GANs using Wasserstein distance?

$$\min_{p_G} W(p_{data}, p_G) = \min_{p_G} \min_{p_J(x,x') \in \prod(p_{data}, p_G)} \mathbb{E}_{p_J}[||x - x'||] \tag{16}$$

Now we have two mins in the formulation, so we are not looking for saddle points any more. However, second min is hard to compute. The proposal is to use dual space instead of the primal space. Moreover, p_{data} is not available, only samples.
Proposed solution: Kantorovich - Rubinstein duality

$$W(p_{data}, p_G) = \max_{||f||_{L \leq 1}} \mathbb{E}_{p_{data}}[f(x)] - \mathbb{E}_{p_G}[f(x')] \tag{17}$$

$$\min_{p_G} \max_{w} \mathbb{E}_{p_{data}}[f_w(x)] - \mathbb{E}_{p_G}[f_w(x')] \tag{18}$$

Now we have a min max again. Below is a result picture showing the Wasserstein GAN.

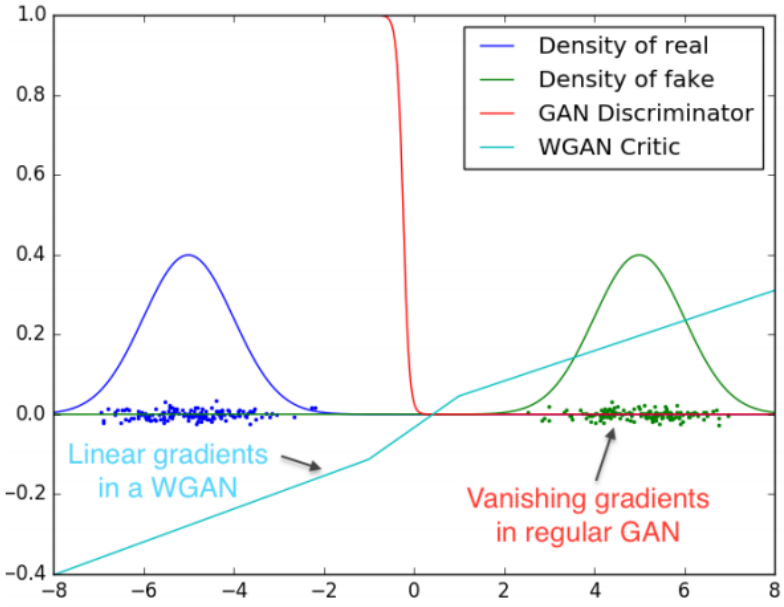


Figure 8: Results of Wasserstein GAN

4 Conclusion

In this lecture, we reviewed maximum likelihood and generative models. We learned a new generative model GAN about its problem formulation, proof and issues in practice. We further studied an extension of GAN: Wasserstein GAN and its formulation.

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[2] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[3] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *arXiv preprint*, 2017.