University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 21: Generative Adversarial Nets

Lecturer: Alexander Schwing Scribe: Aiyu Cui

1 Overview

In this scribe, we summarize lecture 21 of ECE 544 offered by Dr. Schwing in Fall 2018. We will first briefly go through a review of previous related topics, and then talk aboue Generative Adversarial Nets (GANs). Note that all the images are from lecture slides [7], except for those who are cited specifically. You may want to check our course at https://courses.engr.illinois.edu/ece544na/fa2018.

2 Brief Review

Before we start introducing Generative Adversarial Nets (GANs) [2], let's briefly review some concepts here.

2.1 Discriminative vs. Generative

In previous lectures, we have learned both discriminative models and generative models to to model the distribution of data. In general, given a data sample $(\vec{x}, y)$ such that $\vec{x} \in \mathbb{R}^{D_X}$ and $y \in \mathbb{R}$, in which $\vec{x}$ is input data and y is in a discrete output space, a discriminative model will model distribution $p(y|\vec{x})$ and a generative model will model $p(\vec{x}, y)$, from which we can get $p(\vec{x}|y)$ by Bayesian rule. Thus, for generative model, labelled data is not required.

Applications of discriminative models include classifications, detections, segmentation and etc.

Applications of generate models include synthesis of objects (images, text),environment simulator (reinforcement learning, planning), item leveraging unlabeled data and etc.

2.2 Minimize loss functions

Loss function is a protocol to evaluate the difference between model output and ground truth for discriminative models, or how much the clustering criteria is satisfied by the model output for generative models. The goal of minimizing loss functions is to obtain an optimal set of parameters of a model, which allows the model best reconstruct the distributions of sample data. For example, parameters in a fully connected layer include W and b. Common ways to achieve it includes maximize likelihood, minimize squared error (MSE) and so on.

2.3 Generative Models: How to model $p(x)$?

Intuitively, we may assume $p(\vec{x})$ can be described well by a simple distribution, e.g.,Guassian distribution, so then we can rewrite our model as $p(\vec{x}; \mu, \Sigma)$ where μ and Σ are the mean and covariance matrix respectively. Then we can apply maximum likelihood approach to find optimal parameters θ (μ and Σ) as

$$\theta^* = \max_\theta \sum_i \log p(x^i; \theta). \quad (1)$$

Apart from Gaussian distribution, we can also use other sample models like k-means, Gaussian mixture models (GMM) to reconstruct $p(\vec{x})$ by optimizing different parameters. Besides, Variational auto-coder [3] is a slight more complex and powerful way to model $p(\vec{x})$, which will be reviewed briefly in next subsection.

2.4 Variational Auto-Coder

The goal of variational auto-coder [3] is to model $p_\theta(\vec{x}|\vec{z})$, i.e., given $\vec{z}$, we want to generate, compute or sample data $\vec{x}$. The idea of this method is to sample z from a simple Guassian distribution $(\vec{z})$ and then transform it through a deep net to a more complex distribution $p_\theta(\vec{x}|\vec{z})$. To ensure that we know which $\vec{z}$ maps to which $\vec{x}$, we train $p(\vec{z})$ as an auto-encoder (encode $\vec{x}$ to $\vec{z}$) by a deep net $q_\phi(\vec{z}|\vec{x})$, which approximates the mean and variances of $p_\phi(\vec{z}|\vec{x})$ (a Gaussian distribution), as shown in figure 1. In this way, once we have done the training, we could "generate" new data sample by

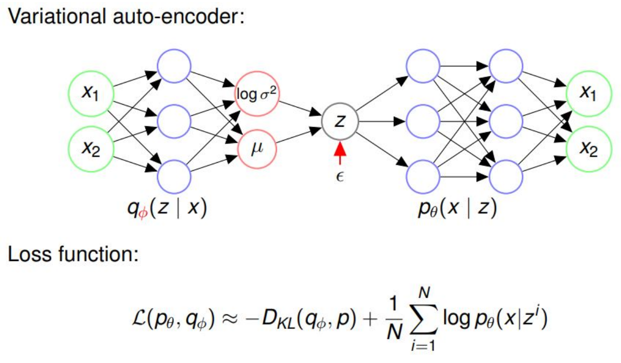

Variational auto-encoder:
x_1
x_2
$\log\sigma^2$
μ
z
ϵ
x_1
x_2
$q_\phi(z \mid x)$
$p_\theta(x \mid z)$
Loss function:
$\mathcal{L}(p_\theta, q_\phi) \approx -D_{KL}(q_\phi, p) + \frac{1}{N}\sum_{i=1}^{N} \log p_\theta(x|z^i)$

Figure 1: Architecture of Variational Auto-encoder: note that we try to estimate $log\sigma$ $(log\Sigma)$ rather than σ because $log\sigma$ $(log\Sigma)$ could automatically ensure the range constraints that are hard to encode directly. Moreover, to sample z, we could first sample a $\vec{\epsilon}$ from standard normal $N(\vec{0}, diag(1))$ and calculate $\vec{z} = \Sigma\vec{\epsilon} + \vec{\mu}$.

drawing sample $\vec{z}$ and throwing it to the decoder $p(\vec{x}|\vec{z})$, which is modeled by another deep neural net. This way is innovative in a sense that previously, people usually try to generate $\vec{x}$ by finding a complex distribution $p(\vec{x})$ and sampling data directly from it.

So far, the variational auto-encoder models the Gaussian distribution well, but what if we want to use other distributions than Gaussian? How can we backpropagate more complex distributions? Generative Adversarial Nets (GANs) are another solution which could address these questions and will be discussed in the coming section.

3 Generative Adversarial Nets (GANs)

Similarly, GAN [2] is another approach to generate data samples. The main idea of GAN is not to write a formula for $p(\vec{x})$ but to learn to sample directly, so that we do not have to deal with the summation over large probability spaces. Then this problem is formed as a game between two players:

- Generator G - generate examples
- Discriminator D - predict whether the example is artificial or real

In this game, G tries to "trick" D by generating samples $G_\theta(\vec{z})$ that are hard to distinguish from the real data, where $\vec{z}$ is a randomly sampled noise vector. G and D could be deep neural nets. A visualization is shown in figure 2.

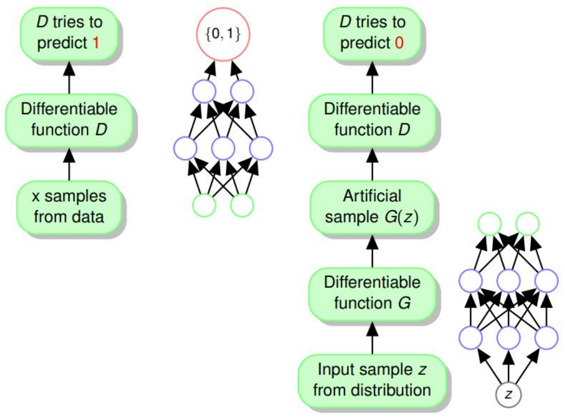

D tries to predict 1
Differentiable function D
x samples from data
{0, 1}
D tries to predict 0
Differentiable function D
Artificial sample G(z)
Differentiable function G
Input sample z from distribution
z

Figure 2: Architecture of GAN

3.1 Loss function

As the game formulated, D is trying to make $D_w(\vec{x}) = p(y=1|\vec{x})$ and $D_w(G_\theta(\vec{z})) = p(y=0|G_\theta(\vec{z}))$. G is trying to make $D_w(G_\theta(\vec{z})) = p(y=1|G_\theta(\vec{z}))$, where y is a binary indicator, if $y=1$, the data is real and if $y=0$, the data is fake. To achieve the goal of D, we choose w by:

$$\min_w -\sum_{\vec{x}} \log D_w(\vec{x}) - \sum_{\vec{z}} \log(1 - D_w(G_\theta(\vec{z}))) \tag{2}$$

and we choose θ by:

$$\max_\theta \min_w -\sum_{\vec{x}} \log D_w(\vec{x}) - \sum_{\vec{z}} \log(1 - D_w(G_\theta(\vec{z}))) \tag{3}$$

3.1.1 Why this loss works

Next, we are going to show that by optimizing this loss function, we could obtain a global optimal $p(G(Z)) = p(X)$, under assumptions that we can find a optimal discriminator with arbitrary capability (ability to model any distribution) and continuity.

First, to train criterion for the discriminator D, given any generator G, we should maximize the quantity $V(G,D)$. (minimizing discrete version of $-V(G,D)$ is equivalent to expression used in (3).)

$$\begin{aligned} V(G,D) &= \int_{\vec{x}} p_{data}(\vec{x}) \log D_w(\vec{x}) dx + \int_{\vec{z}} p_z(\vec{z}) \log(1 - D_w(G_\theta(\vec{z}))) dz \\ &= \int_{\vec{x}} p_{data}(\vec{x}) \log D_w(\vec{x}) dx + \int_{\vec{x}} p_G(\vec{x}) \log(1 - D_w(\vec{x})) dx \\ &= \int_{\vec{x}} p_{data}(\vec{x}) \log D_w(\vec{x}) + p_G(\vec{x}) \log(1 - D_w(\vec{x})) dx \end{aligned} \tag{4}$$

where we can do the integral because of continuity and $p_z(Z)$ is mapped to $p_G(X)$.

By Euler-Lagrange formalism, quantity (4) satisfy the form $S(D) = \int_x L(x, D, D')dx$, so that we have the equality with stationary D:

$$\frac{\partial L(\vec{x}, D, D')}{\partial D} - \frac{d}{dx}\frac{\partial L(\vec{x}, D, D')}{\partial D'} = 0 \tag{5}$$

since in our case, $L(\vec{x}, D, D')$ does not depend on D', the second part in the above equation can be ignored. We then can find the **optimal discriminator** D^* as:

$$\begin{aligned}\frac{\partial L(\vec{x}, D, D')}{\partial D} &= -\frac{p_{data}}{D} + \frac{p_G}{1-D} = 0 \\ \therefore D^*(\vec{x}) &= \frac{p_{data}}{p_{data}(\vec{x}) + p_G(\vec{x})}\end{aligned} \tag{6}$$

Second, With the optimal discriminator and the consequent criterion for generator $C(G) = C(G|D^*) = \max_\theta V(G, D)$, we can the optimize the generator G by minimizing $C(G)$ as:

$$\begin{aligned}
C(G) &= \min_\theta \int_{\vec{x}} p_{data}(\vec{x}) \log D_w(\vec{x}) + p_G(\vec{x}) \log(1 - D_w(\vec{x}))dx \\
&= \min_\theta \int_{\vec{x}} p_{data}(\vec{x}) \log \frac{p_{data}}{p_{data}(\vec{x}) + p_G(\vec{x})} + p_G(\vec{x}) \log \frac{p_G}{p_{data}(\vec{x}) + p_G(\vec{x})} dx \\
&= \min_\theta \int_{\vec{x}} p_{data}(\vec{x})(\log \frac{p_{data}}{\frac{p_{data}(\vec{x})+p_G(\vec{x}))}{2}} - \log 2) + p_G(\vec{x})(\log \frac{p_G}{\frac{p_{data}(\vec{x})+p_G(\vec{x}))}{2}} - \log 2)dx \\
&= \min_\theta KL(p_{data}||\frac{p_G + p_{data}}{2}) + KL(p_G||\frac{p_G + p_{data}}{2}) - \log 2 \int_{\vec{x}} p_{data}(\vec{x})dx - \log 2 \int_{\vec{x}} p_G(\vec{x})dx \\
&= \min_\theta 2\Big(\frac{1}{2}KL(p_{data}||\frac{p_G + p_{data}}{2}) + \frac{1}{2}KL(p_G||\frac{p_G + p_{data}}{2}\Big) - 2\log(2) \\
&= \min_\theta 2JSD(p_{data}, p_G) - \log(4)
\end{aligned} \tag{7}$$

where KD is Kullback-Leibler divergence and JSD is Jensen-Shannon divergence:

$$KL(P||Q) = \int_{-\infty}^{\infty} p(x) \log\Big(\frac{p(x)}{q(x)}\Big)dx \tag{8}$$

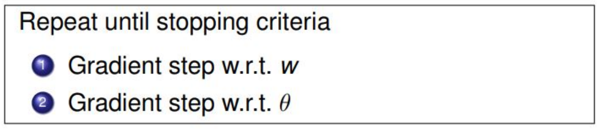
$$JSD(P||Q) = \frac{1}{2}KL(P||M) + \frac{1}{2}KL(Q||M), \text{ where } M = \frac{1}{2}(P + Q) \tag{9}$$

Since the Jensen-Shannon divergence between two distribution is always non-negative and zero only when the two distributions are equal, we prove that $C^* = -\log 4$ when and only when $p_G = pdata$, which is to say, the generator perfectly models the real data's distribution. Hence, we show that the loss function mathematically works to achieve the goal of GANs.

3.2 Optimize loss function

The next question will be how we can optimize the loss function (3). A intuitive way to do it would be stochastic gradient descent as shown in figure 3.

However, in practice, there are some problems with using SGD for GAN directly and heuristics would work better. For example, if we get discriminator D trained pretty well but generator G is


Repeat until stopping criteria

1. Gradient step w.r.t. *w*
2. Gradient step w.r.t. θ

Figure 3: SGD of GAN

very bad, we will nearly get no gradient, if we try to improve optimization by:

$$\begin{aligned} C(G) &= \max_{\theta} V(D,G) \\ &= \boxed{\max_{\theta}} \min_{w} -\sum_{\vec{x}} \log D_w(\vec{x}) - \boxed{\sum_{\vec{z}} \log(1 - D_w(G_\theta(\vec{z})))} \end{aligned} \tag{10}$$

because the value of $V(D,G)$ is already small enough and thus we could not get large change in y even with a large x on a negative log curve as shown in figure 4. To avoid it, we can play a little

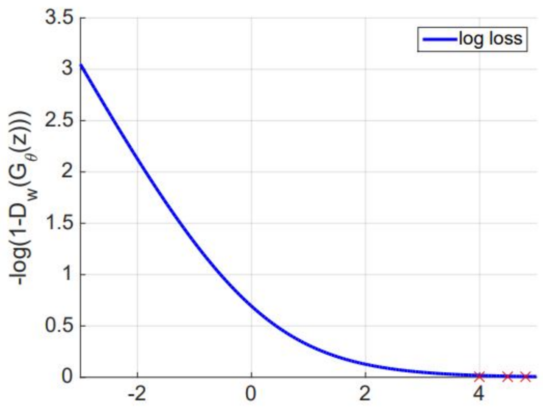


Figure 4: Example when GAN not working well

trick here. Instead of trying to optimize $V(D,G)$ as (10) jointly, to optimize G, we do:

$$\min_{\theta} -\log D_w(G_\theta(\vec{z})) \tag{11}$$

In this way, we flip the log curve, so that we could get enough gradient change every time to approach the optimal as shown in figure 5. Though in this way, the proof we just showed in the previous section no longer holds, it works well in practice.

4 Evaluation of GAN

Here we are going to show some example generated images out of GAN model and its variance models. GAN is currently a very hot topic in Computer Vision field and many interesting research are going on every day. There is seemingly not a general way to quantitatively evaluate the generation solutions, so only the quality results are showing here in figures 6. 7, 8 and 9.

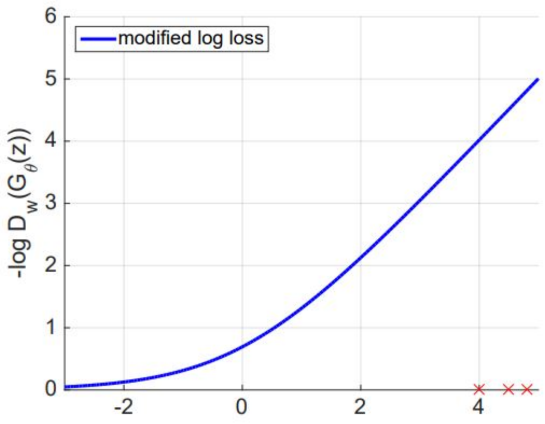


Figure 5: The modified loss to address the above problem

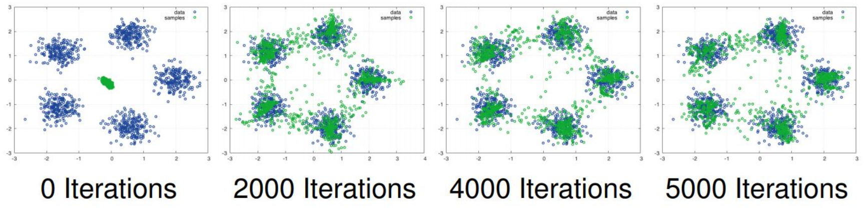


Figure 6: This is a simulation of modeling distribution. The blue dots are from real set and green dots are generated. As shown, the more iterations were trained, the learned distribution is closer to the real distribution.

5 Wasserstein GAN

a variation of GAN called Wasserstein GAN [1] was introduced in 2017.

5.1 Wasserstein distance

Wasserstein distance between two distributions μ and υ is measured as:

$$W(\mu,\upsilon) = \inf_{\gamma \in \prod(\mu,\upsilon)} \mathbb{E}_{X,G}\Big[||X-G||_1\Big]. \tag{12}$$

Visualization is available in figure 10. Wasserstein distance has the following properties:

- Given $G \sim \upsilon$ and $X \sim \mu$, if $W(\upsilon,\mu) = 0$, then $\upsilon = \mu$.
- Let $G^N \sim \upsilon^N$ and $X \sim \mu$. If $W(\mu,\upsilon^N) \to 0$ as $N \to \infty$, then $G^N \xrightarrow{d} X$, where $G^N \xrightarrow{d} X$ denotes convergence in distribution, i.e. $\mathbb{E}\Big[f(G^N)\Big] \to \mathbb{E}\Big[f(X)\Big]$ for any continuous, bounded function f.

By the Kantorovich-Rubinstein duality,

$$W(\mu,\upsilon) = \sup_{||f||_L \leq 1} \Big[\mathbb{E}_{X\sim\mu}[f(X)] - \mathbb{E}_{G\sim\upsilon}[f(G)]\Big], \tag{13}$$

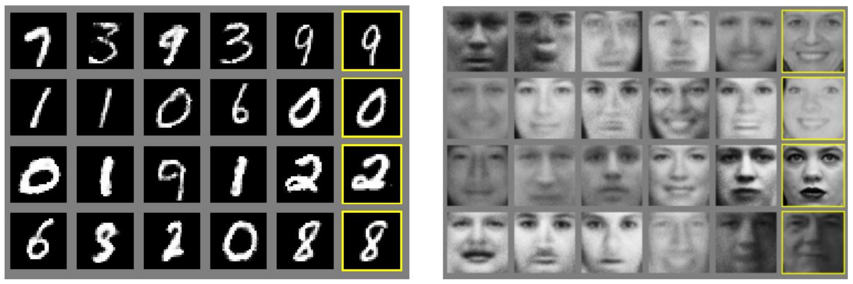


Figure 7: Outputs from GAN [2]. The rightmost column shows the nearest training sample, which can demonstrate the model is not memorizing the training set.

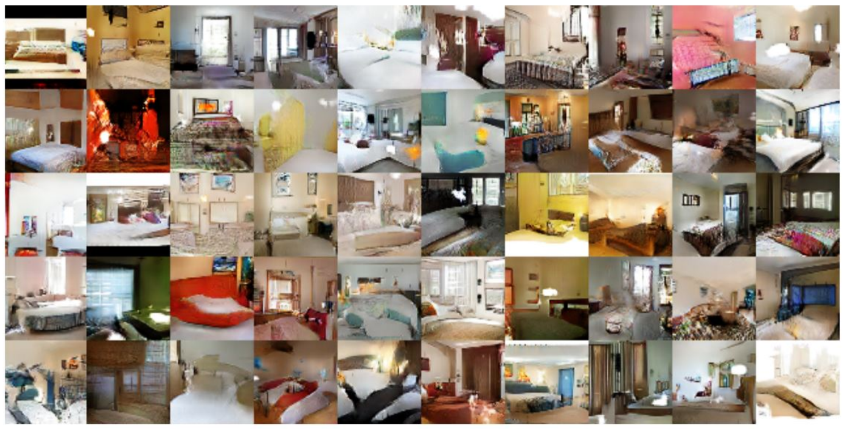


Figure 8: Outputs of a variation of GAN, DCGAN [5].

where $\{f : ||f||_L \leq 1\}$ is the set of functions f such that

$$||f(x) - f(x')||_1 \leq ||x - x'||_1 \tag{14}$$

That is, function f which are Lipschitz continuous with Lipschitz constant $K = 1$.

5.2 Wasserstein GAN

Wasserstein GAN uses a different metrics to measure distance between the two distribution p_{data} and p_G:

$$W(p_{data}, p_G) = \min_{p_J(x,x') \in \prod(p_{data}, p_G)} \mathbb{E}_{p_G}\Big[||x - x'||_1\Big] \tag{15}$$

where $\prod(p_{data}, p_G)$ is any probability measure with marginal p_{data} and p_G. Thus, we can formulate GAN using Wasserstein distance as:

$$\min_{P_G} W(p_{data}, p_G) = \min_{P_G} \min_{p_J(x,x') \in \prod(p_{data}, p_G)} \mathbb{E}_{p_G}\Big[||X - G||_1\Big] \tag{16}$$

However, we cannot directly calculate it since p_{data} is not available. Instead, by Kantorovich-Rubinstein duality, we know:

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

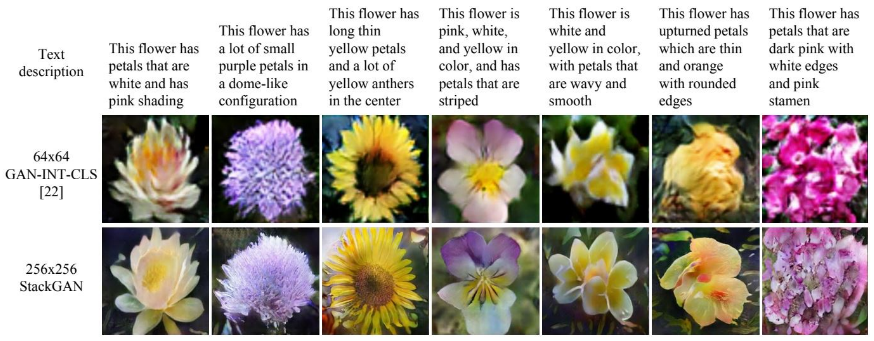

Text description
This flower has petals that are white and has pink shading
This flower has a lot of small purple petals in a dome-like configuration
This flower has long thin yellow petals and a lot of yellow anthers in the center
This flower is pink, white, and yellow in color, and has petals that are striped
This flower is white and yellow in color, with petals that are wavy and smooth
This flower has upturned petals which are thin and orange with rounded edges
This flower has petals that are dark pink with white edges and pink stamen
64x64 GAN-INT-CLS [22]
256x256 StackGAN

Figure 9: Example results StackGAN [9] and GAN-INT-CLS [6], which is conditioned on text description from Oxford-102 [4] test set.

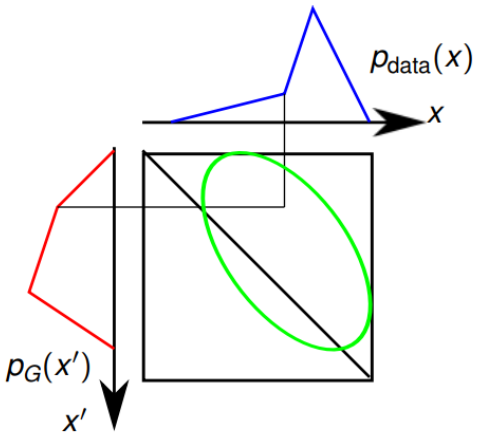


Figure 10: Wasserstein distance: the more similar the two distributions are, the closer the joint distribution is to the diagonal so that the smaller distance is.

Then, the loss becomes:

$$\min_{P_G} W(p_{data}, p_G) = \min_{P_G} \max_{||f||_L \leq 1} \mathbb{E}_{p_{data}}[f(x)] - \mathbb{E}_{p_G}[f(x')]. \tag{18}$$

Here is an example result comparing with GAN shown in figure 11.

6 Quiz

- What generative modeling techniques do you know about?
  - Variational auto-encoder, GAN and RNN and so on.
- What are the short-comings of those techniques?
  - Variational Auto-encoder (VAEs): the generate images is slightly blurry. [8]
  - RNN: inefficient sampling and no low-dimensional code. [8]
- What differentiates GANs from other generative models?

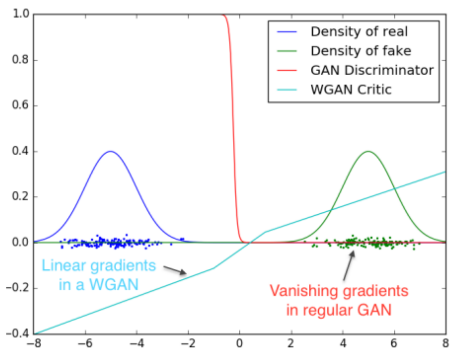


Figure 11: As shown, N from gradient vanishing problem, but not for WGAN.

    - rather than learn a given distribution, GAN tries to model the data directly.

- What are the short-comings of GANs?
    - difficult to optimize (unstable) [8]

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[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] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[4] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conference on*, pages 722–729. IEEE, 2008.

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

[6] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

[7] A. G. Schwing. L21: Generative adversarial nets. *ECE 544, University of Illinois at Urbana-Champaign*, 2018.

[8] A. G. Schwing. L22: Autoregressive methods (rnns/lstms/grus). *ECE 544, University of Illinois at Urbana-Champaign*, 2018.

[9] 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.