CS 182/282A Designing, Visualizing and Understanding Deep Neural Networks
Spring 2021 Sergey Levine **Discussion 11**

This discussion focuses on generative adversarial networks and adversarial attacks.

1 Generative Adversarial Networks

In the previous discussion, we focused on training generative models by directly maximizing the likelihood of the inputs we observe in our training data. This week, we'll consider an alternative method for training generative models for high dimensional inputs that does not explicitly compute likelihoods, but will train the generative model to fool a learned a learned *discriminator* network, which is being trained to distinguish between real and generated inputs.

1.1 The GAN Game

For GANs, we will typically generate samples by first sampling vectors from some fixed noise distribution $Z \sim p(Z)$, and then passing them through a (deterministic) generator function G_θ (parameterized by θ) to obtain samples $\tilde{X} = G_\theta(Z) \sim p_G(\tilde{X})$.

Our discriminator will be a binary classifier D_ϕ (parameterized by ϕ), which will be trained to discriminate between the generated samples $\tilde{X}$ and the true data distribution $X \sim p(X)$.

We can write view the training of GANs as solving a two-player game given by

$$\begin{aligned}\min_{G_\theta} \max_{D_\phi} &= E_{X\sim p(X)}[\log D_\phi(X)] + E_{\tilde{X}\sim p_G(\tilde{X})}[\log\Big(1 - D_\phi(\tilde{X})\Big)] \\ &= E_{X\sim p(X)}[\log D_\phi(X)] + E_{Z\sim p(Z)}[\log\big(1 - D_\phi(G_\theta(Z))\big)].\end{aligned}$$

If we hold the generator G_θ fixed, then training the discriminator D_ϕ would be exactly the same as training a normal binary classifer. If we hold the discriminator D_ϕ fixed, then training the generator is simply optimizing the generator to generate samples that the discriminator thinks are valid inputs.

Note that this objective does not require us to compute the likelihoods $p_\theta(\tilde{x})$ for any generated image (or any other image) under the generator's distribution, which gives us more flexibility unlike our previously covered latent variable models which required tractably computable log-likelihoods to train.

In practice, we optimize GANs by alternating taking gradient steps on the discriminator and generator, rather than fully optimizing the discriminator before updating the generator as this minimax game suggests.

1.2 GANS with the "perfect" discriminator

To gain intuition for what training a GAN *should* do, we consider an idealized setting where our discriminator is infinitely expressive and is fully optimized to convergence for every generator update.

In this case, for any input x and fixed generator G, the optimal discriminator D^* assigns probability

$$D^*(x) = \frac{p(x)}{p(x) + p_G(x)}.$$

Problem 1: Optimal Discriminator

Show that the optimal discriminator probability $D^*(x)$ is given by the expression above.

Solution 1: Optimal Discriminator

Since our discriminator is perfectly expressive, we only need to consider what happens at each input x pointwise.
The relevant part of the objective for the discriminator value at a particular x is simply

$$\max_{D(x)} p(x)\log D(x) + p_G(x)\log\big(1 - D(x)\big).$$

To optimize this, we can then take derivatives with respect to the discriminator probability $D(x)$ and solve for a stationary point. Taking derivatives, we have

$$\begin{aligned} 0 &= p(x)\cdot\frac{1}{D(x)} - p_G(x)\cdot\frac{1}{1-D(x)} \\ p_G(x)D(x) &= p(x)(1-D(x)) \\ D(x) &= \frac{p(x)}{p(x)+p_G(x)}. \end{aligned}$$

We can substitute this optimal discriminator into our two player game and reduce to a single optimization over the generator G_θ as

$$\min_{G_\theta} E_{X\sim p(X)}\left[\log\left(\frac{p(X)}{p(X)+p_G(X)}\right)\right] + E_{\tilde{X}\sim p_G(\tilde{X})}\left[\log\left(\frac{p_G(\tilde{X})}{p(\tilde{X})+p_G(\tilde{X})}\right)\right].$$

Defining $q(x) = \frac{p(x)+p_G(x)}{2}$, then we can rewrite the objective as

$$\min_{G_\theta} \underbrace{E_{X\sim p(X)}\left[\log p(X) - \log q(X)\right]}_{D_{KL}(p(x)\|q(x))} + \underbrace{E_{\tilde{X}\sim p_G(\tilde{X})}\left[\log p_G(\tilde{X}) - \log q(\tilde{X})\right]}_{D_{KL}(p_G(x)\|q(x))} + \text{constant}.$$

We recognize this objective as being (up to the additive constant that doens't matter for optimization) to precisely be the Jensen-Shannon divergence between the true data distribution $p(X)$ and the generator distribution $p_G(\tilde{X})$. This shows that in the ideal setting with a perfect discriminator, training a GAN does in fact optimize the generator distribution to be close to the data distribution (as measured by the Jensen-Shannon divergence).

1.3 Training GANs in Practice

Of course, we will generally not find the optimal discriminator (due to computational limitations and representational limitations on the discriminator architecture), so we will generally not precisely be minimizing the Jensen-Shannon divergence when training GANs. It also turns out that having a perfect discriminator and directly trying minimize the Jensen-Shannon can be very undesirable for training GANs. In Figure 1, we see that the ideal discriminator values (red) are essentially constant on all the generated data, so the gradient for the generator would be extremely small, which can make optimizing the generator extremely slow.

The key idea to improve GAN training is that we don't actually care how well the discriminator does by itself, but rather we only care that it provides a useful signal to improve the generator. A common way we can accomplish this is to restrict the expressivity of discriminator, often by enforcing some additional smoothness condition. As we see in Figure 2, the smoother discriminator values in green can provide a more useful learning signal for the generator.

Wasserstein GANs (which motivate imposing a Lipschitz constraint on the discriminator as minimizing the Wasserstein distance between the generator and real data distributions), gradient penalty GANs, and

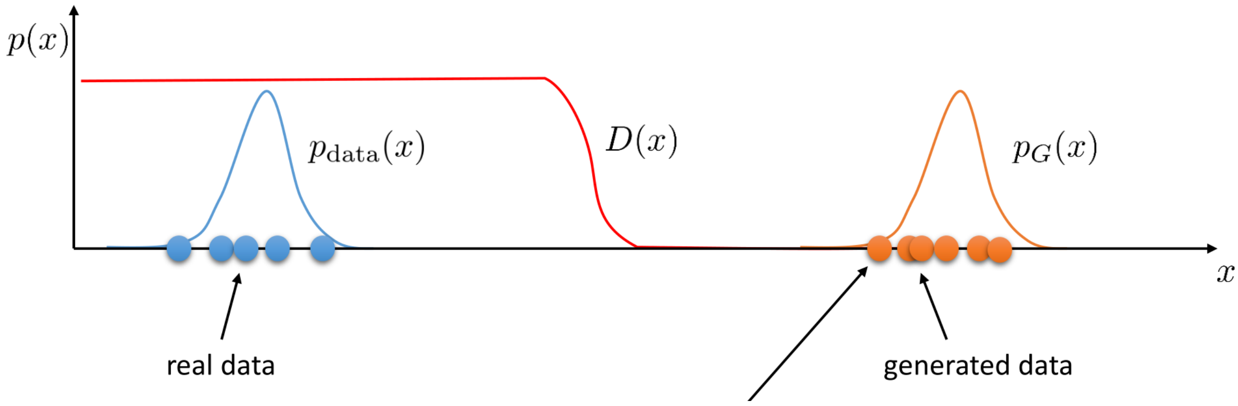

Figure 1: Perfect discriminator values (red line) with real (blue) and generated (orange) inputs in a 1D example. The generator will have very little gradient signal on how to improve, making this discriminator undesirable.

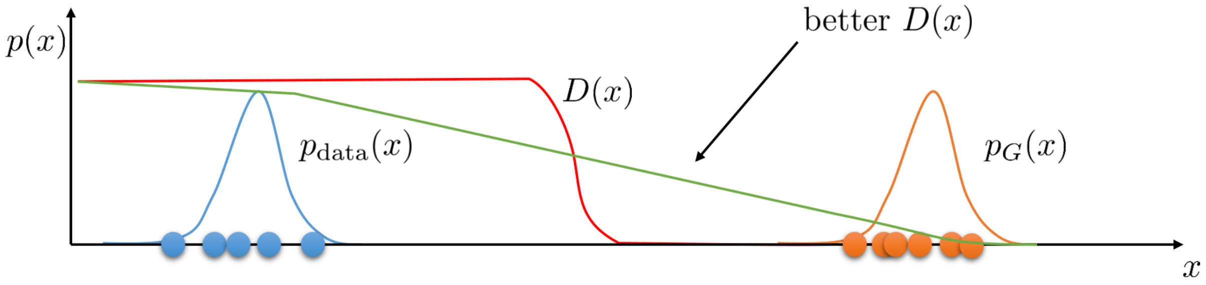

Figure 2: A discriminator constrained to be Lipschitz continuous (green line) with real (blue) and generated (orange) inputs in a 1D example. The generator can now follow the slope of the green line to move generated samples towards the real data.

spectral normalized GANs all enforce different notions of smoothness on the discriminator in order to make learning the generator easier.

Other tricks to avoid vanishing gradients include using real-valued discriminators like in Least-squares GAN (which turns out to be equivalent to minimizing the Pearson χ^2 divergence under ideal settings), and instance noise, which adds noise to the inputs to smooth out the densities of both the real and generated data distributions, improving the learning signal.

1.3.1 Techniques for Lipschitz Continuity

Recall from lecture that we can the Wasserstein distance $W(p, p_G)$ as

$$W(p, p_G) = \sup_{\|f\|_L \leq 1} E_{X \sim p(X)}[f(x)] - E_{\tilde{X} \sim p_G(\tilde{X})}[f(x)],$$

where f is restricted to be a 1-Lipschitz function. The original WGAN paper suggested approximating f (analagous to the discriminator) with a neural network parameterized by ϕ and enforcing a Lipschitz constraint by clipping the each entry of the weights ϕ to have magnitude less than ϵ. While this **weight clipping** will ensure that the discriminator f_ϕ is K-Lipschitz for some K, but the exact constant K would depend on the architecture and can be a bit complicated to compute.

Another more elegant way to enforce Lipschitz continuity via **gradient penalties**. Here, we add an additional term $\lambda(\left\|\nabla_x f(x)\right\|_2 - 1)^2$ to our loss for the discriminator, directly encouraging the norm of the gradients to have norm close to 1. While this doesn't strictly enforce 1-Lipschitzness due to it only being a soft penalty on the gradient norm, it is effective and commonly used.

Finally, **spectral normalization** can be used to strictly enforce a 1-Lipschitz constraint on the discriminator f_ϕ. We first note that if two functions f, g are L_1 and L_2 Lipschitz respectively, then their composition $f \circ g$ is $L_1 L_2$-Lipschitz, and we can extend this to any finite composition of Lipschitz functions.

Problem 2: Composition of Lipschitz Functions

If f, g are L_1, L_2 Lipschitz functions, then prove their composition $f \circ g$ is $L_1 L_2$-Lipschitz.

Solution 2: Composition of Lipschitz Functions

If $y_1 = g(x_1)$ and $y_2 = g(x_2)$, then we know by definition that $\|y_1 - y_2\| \leq L_2 \|x_1 - x_2\|$. Similarly, if $z_1 = f(y_1)$ and $z_2 = f(y_2)$, then $\|z_1 - z_2\| \leq L_1 \|y_1 - y_2\|$. Subsituting in our bound on $\|y_1 - y_2\|$, we have $\|z_1 - z_2\| \leq L_1 L_2 \|x_1 - x_2\|$, which means the mapping $f \circ g$ is $L_1 L_2$-Lipschitz.

We then note typical neural nets can be written as compositions of functions $f_n \circ \sigma \circ \ldots, \sigma, f_1$, where σ represents the nonlinear activation functions and f_i are some affine layer parameterized by (W_i, b_i). Therefore, one way to ensure f_ϕ is 1-Lipschitz is by ensuring each f_i and σ are all 1-Lipschitz.

We can easily verify that the ReLU activation is 1-Lipschitz (as it is either constant with slope zero or linear with slope 1), so all that remains is to enforce that each linear layer f_i is 1-Lipschitz. Clearly, the Lipschitzness of f_i does not depend on the bias parameter b_i, so we only need to consider the Lipschitzness of the function $g(x) = W_i x$. The Lipschitz constant K of the linear function g can be written as the supremum

$$\sup_{\|x\|_2 = 1} \|W_i x\|_2,$$

which we recognize to be the spectral norm $\sigma(W_i)$ (the largest singular value of W_i). Thus, a simple way to enforce 1-Lipschitzness for each linear layer (and thus the whole network) is to simply renormalize $W_i \leftarrow \frac{W_i}{\sigma(W_i)}$ after each gradient update, as the spectral norm $\sigma(W_i)$ is fairly straightforward and cheap to compute.

2 Adversarial Examples

Adversarial examples are inputs that are specially chosen or constructed to fool a model. One reason we study adversarial examples is that they directly offer ways to exploit our learned models. As a real world example, we can imagine someone modifying a stop sign in order to fool a self-driving car system into not stopping, which can potentially lead to crashes.

In this section, we will discuss some common formulations for adversarial attacks, strategies for constructing these adversarial examples, and techniques for mitigating adversarial attacks.

2.1 Formulation

One common way we formulate adversarial attacks is by allowing an additive perturbation δ to a real image x, while restricting the size of the perturbation by enforcing $\|\delta\| < \epsilon$ for some choice of size $\|\cdot\|$ and some budget ϵ. Intuitively, the idea here is that we would like the adversarial examples to be imperceptible to humans and to not change the "true" label, hence the restriction on how much we are allowed to perturb the real input.

To find an adversarial example for some particular loss function and model specified by θ, we want to fool the model (by forcing it to have high loss) by solving the constrained problem

$$\delta^* * = \arg\max_{\delta} L_\theta(\underbrace{x+\delta}_{\text{perturbed input}}, \underbrace{y}_{\text{original label}}) \\ \text{s.t} \|\delta\| \leq \epsilon,$$

and take $x' = x + \delta^*$ as our adversarial example. For images, common choices of the norm $\|\cdot\|$ include the ∞-norm $\|\delta\|_\infty = \max_i |\delta_i|$ or the usual 2-norm $\|\delta\|_2 = \sqrt{\sum_i \delta_i^2}$.

Finally, we note that this definition, while providing a formal definition of adversarial attacks that is reasonable to analyze, does not necessarily align well with adversarial examples in the real world. Real world adversaries are not necessarily limited in how much they perturb the input (at least, often not limited in precisely the way we assume in this formulation).

2.2 Adversarial Attack Strategies

We first consider **white-box attacks**, where we assume the adversary has full access to our model θ. In particular, we assume the adversary can compute gradients $\nabla_x L_\theta(x, y)$, The attacker can then solve the constrained problem with standard first-order constrained optimization techniques like dual gradient ascent to find the optimal perturbation δ^*.

We can also construct much simpler attacks with the **fast gradient sign method** (FGSM). Instead of solving for the optimal δ^* exactly, which can be somewhat computationally expensive, we instead make a first-order approximation to the loss as

$$L'_\theta(x + \delta, y) \approx L_\theta(x, y) + \nabla_x L_\theta(x, y)^T \delta.$$

Using our linearized loss L' instead of the true loss L, we can then easily construct the optimal δ^* using only one gradient evaluation and often a closed form solution for the optimal δ^*.

For example, the optimal solution for a 2-norm constraint would result in the perturbation

$$\delta^* = \epsilon \frac{\boldsymbol{\nabla}_x L_\theta(x, y)}{\left\|\boldsymbol{\nabla}_x L_\theta(x, y)\right\|_2}.$$

For an ∞-norm constraint of ϵ, the optimal perturbation is given by

$$\delta^* = \epsilon \, \mathrm{sign}(\boldsymbol{\nabla}_x L_\theta(x, y)),$$

hence why this simple attack is known as the fast gradient sign method.

Problem 3: FGSM optimal perturbation

Show that the optimal perturbation δ given a linear loss and ∞-norm constraint is given by the expression above.

Solution 3: FGSM optimal perturbation

Suppose we have a d-dimensional perturbation $\delta \in \mathbb{R}^d$. We can rewrite the ∞-norm constraint as d separate constraints of the form $|\delta_i| \leq \epsilon$. Our objective is to then solve

$$\arg\max_{\delta} \nabla_x L_\theta(x, y)^T \delta \qquad\qquad \text{s.t. for all } i \in \{1, \ldots, d\} : |\delta_i| \leq \epsilon.$$

Noting that $\nabla_x L_\theta(x, y)^T \delta = \sum_{i=1}^d g_i \delta_i$ (where g_i are the entries of the gradient), we see that this problem is *separable*, meaning that we can treat each coordinate δ_i completely independently.
Now, in each dimension i, the solution can be obviously found by plotting a linear function along the interval $[-\epsilon, \epsilon]$. If the linear function has positive slope (g_i is positive), then the optimum is at ϵ, and if it has negative slope (g_i is negative), then the optimum is at $-\epsilon$.
Hence, the optimal perturbation always has an entry of size ϵ at each coordinate, with the sign depending on the sign of the partial derivative g_i.

In practice, this simple method is very fast and convenient, works well against standard neural networks, but can often be defeated by simple defenses.

We can also consider **black-box attacks**, which do not assume access to the internal workings of the model θ, but instead only observes the model's predictions. Without access to gradients, we can no longer directly run first-order optimization algorithms for the FGSM, as those required taking the gradient of the loss with respect to the input. However, in black box settings, we can still *estimate* the gradient with a finite differences approach.

Recalling that the (single variable) derivative $f'(x)$ is defined as the limit $\lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$, a simple finite differences method estimates gradients by taking small perturbations in each dimension to the input and seeing how the loss changes to approximate that coordinate of the derivative. In practice, this allows us to approximate the gradient with only a moderate number of queries to the model, which we can then use to run our adversarial attacks.

Another way we can construct black box attacks is to learn our own model, perform a white box attack on our own model, and reuse the generated adversarial image to attack the target model. It turns out this strategy often works quite well, and relies on the two models having similar gradients with respect to the input.

2.3 Adversarial Defense

There are many techniques for detecting adversarial examples or making models robust to them. One common way to robustfy networks to adversarial attacks is to incorporate adversarial attacks into the training procedure via a robust loss function, which we refer to as **adversarial training**.

Instead of finding the optimal parameter $\theta^* = \arg\min_\theta \sum_{x,y \in D} L_\theta(x, y)$ we instead minimize the robust loss

$$\theta^* = \arg\min_{\theta} \sum_{x,y \in D} \max_{\|\delta\| \leq \epsilon} L_\theta(x + \delta, y).$$

Thus, we are explicitly training our network to be robust to the type of attack we expect at test time.

However, adversrial training does come with its drawbacks. Computing the adversarial attack for each input at every training step can slow down training, and often times decreases the accuracy on clean data.

Additionally, adversarial training with respect to a certain perturbation constraint does not necessarily generalize to other perturbations.

Problem 4: Adversarial Robustness to Other Pertubations

Suppose we are training a model to classify whether or not there is a stop sign in an image, and our training set consists only of images taken during the day. We use adversarial training to be robust to perturbations δ satisfying $\|\delta\|_\infty \leq 0.5$, and verify that our model robustly generalizes well with this perturbation. Which of the following different perturbation sets should we expect our model to also be robust to?

1. $\{\delta : \|\delta\|_\infty \leq 0.1\}$
2. $\{\delta : \|\delta\|_\infty \leq 1.0\}$
3. $\{\delta : \|\delta\|_2 \leq 0.5\}$
4. $\{\delta : \|\delta\|_2 \leq 1.0\}$
5. Images of stop signs at night, with much dimmer lighting.

Solution 4: Adversarial Robustness to Other Perturbations

The correct answers are choices 1 and 3.

Choice 1 ($\|\delta\|_\infty \leq 0.1$) is a perturbation set strictly contained within the one we trained for, so if we generalized well to the larger perturbation set, we will also generalize to the smaller one.

We will not necessarily generalize well to choice 2 ($\|\delta\|_\infty \leq 1.0$), as the new perturbation set is larger than the one we trained for. In this new set, the perturbations are allowed to be more severe than those we've verified our model is robust to, so these more severe attacks can still potentially fool our model.

We will generalize to choice 3 ($\|\delta\|_2 \leq 0.5$), since we know $\|\delta\|_2 \geq \|\delta\|_\infty$ for any δ, which implies this set is contained within our training perturbations.

We will not necessarily do well for choice 4 ($\|\delta\|_2 \leq 1.0$), since again, it contains perturbations that were not allowed during training.

We also should not expect to generalize well to choice 5, as nighttime can cause drastic changes to the image when measured by ∞-norm. In general, it is hard to translate notions of natural distribution shifts into precise mathematical formulations like a norm constraint.