CS 182/282A Designing, Visualizing and Understanding Deep Neural Networks
Spring 2022 Marvin Zhang

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 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.

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.

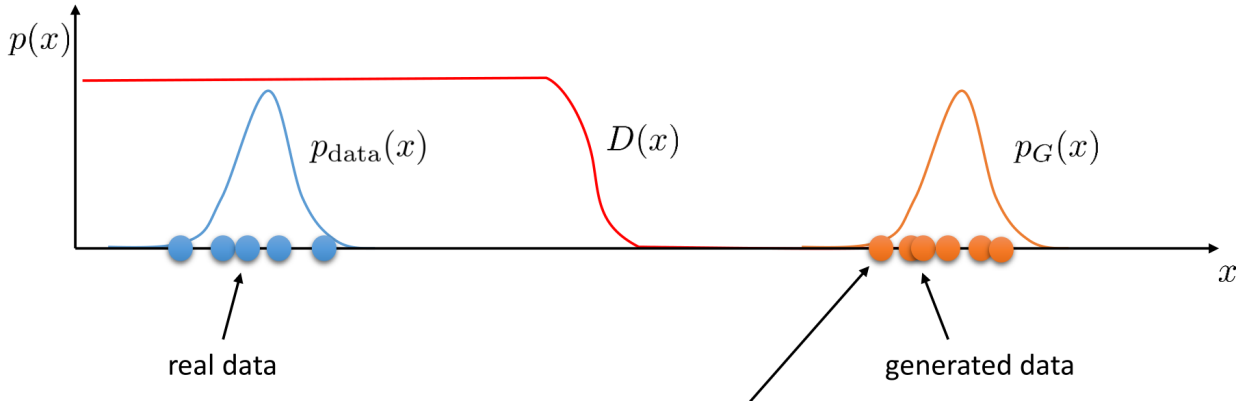

Figure 1: Perfect discriminator values (red line) with real (blue) and generated (orange) inputs in a 1D example. The generator will have very little gradient signal on how to improve, making this discriminator undesirable.

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

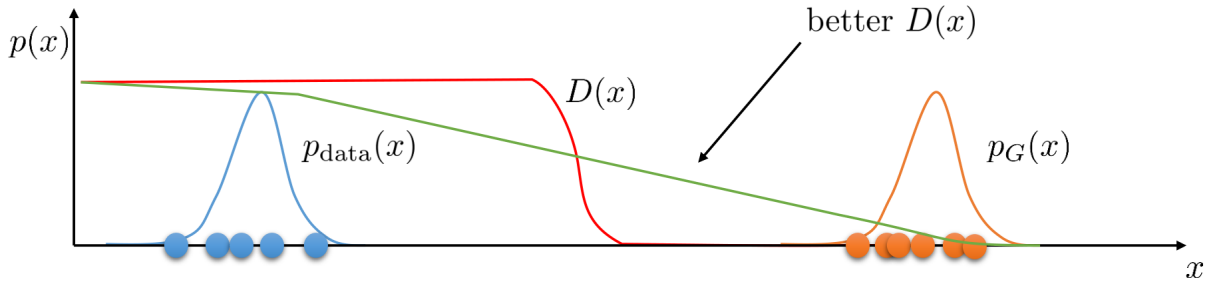

Figure 2: A discriminator constrained to be Lipschitz continuous (green line) with real (blue) and generated (orange) inputs in a 1D example. The generator can now follow the slope of the green line to move generated samples towards the real data.

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

We define 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.

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 Latent Variable Models

Formally, a latent variable model p is a probability distribution over observed variables x and latent variables z (variables that are not directly observed but inferred), $p_\theta(x, z)$. Because we know z is unobserved, using learning methods learned in class (like supervised learning methods) are unsuitable.

Indeed, our learning problem of maximizing the log-likelihood of the data turns from,

$$\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_{i=1}^{N} \log p_\theta(x_i)$$

to the following,

$$\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_{i=1}^{N} \log \int p_\theta(x_i|z)p(z)dz$$

where we recognize $p(x) = \int p(x|z)p(z)dz$. Unfortunately, the integral is intractable, but we will discuss ways to find a tractable lower bound.

2.1 Varational Autoencoders (VAE)

The VAE uses the autoencoder framework to generate new images. For the following description of encoder and decoder of the VAE, let us assume our input x is a 28×28 photo of a handwritten digit in black-and-white, and we wish to encode this information into a latent representation of space z.

Encoder Encoder maps a high-dimensional input x (like the pixels of an image) and then (most often) outputs the parameters of a Gaussian distribution that specify the hidden variable z. In other words, they output $\mu_{z|x}$ and $\Sigma_{z|x}$. We implement this as a deep neural network, parameterized by ϕ, which computes the probability $q_\phi(z|x)$. We can sample from this distribution to get noisy values of the representation z.

Decoder Decoder maps the latent representation back to a high dimensional reconstruction, denoted as $\hat{x}$, and outputs the parameters to the probability distribution of the data. We implement this as another neural network, parametrized by θ, which computes the probability $p_\theta(x|z)$. Following the digit example, if we represent each pixel as a 0 (black) or 1 (white), probability distribution of a single pixel can be then represented using a Bernoulli distribution. Indeed, the decoder gets as input the latent representation of a digit z and outputs 784 Bernoulli parameters, one for each of the 784 pixels in the image.

Training VAEs To train VAEs, we find parameters that maximize the likelihood of the data,

$$\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_{i=1}^{N} \log \int p_\theta(x_i|z)p(z)dz$$

This integral is intractable. However, we can show that it is possible to optimize a tractable lower bound on the data likelihood, called the Evidence Lower Bound (ELBO),

$$\mathcal{L}_i = \mathbb{E}_{z\sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] - D_{KL}(q_\phi(z|x_i)||p(z))$$

Problem 2: Blurry Images

Why do VAEs typically produce blurry images?

2.2 Variational Inference

In this subsection, we will derive the variational approximation in discrete form, and discuss the re-parametrization trick.

Problem 3: Latent Variable Model

Write out the log-likelihood objective of a discrete latent variable model.

Problem 4: Variational Approximation

Show that

$$\sum_{i=1}^{N} \log p_\theta(x_i) \geq \sum_{i=1}^{N} \mathbb{E}_{q(z|x_i)}[\log p_Z(z) - \log q(z|x_i) + \log p_\theta(x_i|z)]$$

Hint: Use Jensen's Inequality, which states, $\log \mathbb{E}[X] \geq \mathbb{E}[\log X]$

Problem 5: Variational Approximation Optimization

To optimize the Variational Lower Bound derived in the previous problem, which distribution do we sample z from?

Combining it with Entropy Recall the entropy function,

$$H(p) = -\mathbb{E}[\log p(x)] = -\int_X p(x) \log p(x) dx$$

and also recall KL-Divergence,

$$D_{KL}(q||p) = \mathbb{E}\left[\log \frac{q(x)}{p(x)}\right] = -E[\log p(x)] - H(q)$$

We can show that, our derived approximation can be reformulated as the Evidence Lower Bound (ELBO),

$$\mathcal{L}_i = \mathbb{E}_{z\sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] - D_{KL}(q_\phi(z|x_i)||p(z))$$

Re-Parametrization Trick The re-parametrization trick allows us to break $q_\phi(z|x)$ into a deterministic and stochastic portion, and re-parametrize from $q_\phi(z|x)$ to $g_\phi(x, \epsilon)$. In fact, we can let, $z = g_\phi(x, \epsilon) = g_0(x) + \epsilon \cdot g_1(x)$ where $\epsilon \sim p(\epsilon)$. This reparametrization trick is simple to implement, and has low variance [1]

Problem 6: Reparametrization Example

Let us get an intuition for how we might use re-parametrization in practice. Assume we have a normal distribution q, parametrized by θ, such that, $q_\theta(x) \sim \mathcal{N}(\theta, 1)$, and we would like to solve,

$$\min_\theta \mathbb{E}_q[x^2]$$

Use the re-parametrization trick on x to derive the gradient.

[1] See Appendix of this paper for more information