CS 236 Homework 3 Solutions

Instructors: Stefano Ermon and Aditya Grover

{ermon,adityag}@cs.stanford.edu

Available: 11/12/2018; Due: 23:59 PST, 12/3/2018

Problem 1: Generative adversarial networks (15 points)

In this problem, we will implement a generative adversarial network (GAN) that models a high-dimensional data distribution $p_{\text{data}}(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathbb{R}^n$. To do so, we will define a generator $G_\theta : \mathbb{R}^k \to \mathbb{R}^n$; we obtain samples from our model by first sampling a k-dimensional random vector $\boldsymbol{z} \sim \mathcal{N}(0, I)$ and then returning $G_\theta(\boldsymbol{z})$.

We will also define a discriminator $D_\phi : \mathbb{R}^n \to (0, 1)$ that judges how realistic the generated images $G_\theta(\boldsymbol{z})$ are, compared to samples from the data distribution $x \sim p_{\text{data}}(\boldsymbol{x})$. Because its output is intended to be interpreted as a probability, the last layer of the discriminator is frequently the **sigmoid** function,

$$\sigma(x) = \frac{1}{1 + e^{-x}},$$

which constrains its output to fall between 0 and 1. For convenience, let $h_\phi(\boldsymbol{x})$ denote the activation of the discriminator right before the sigmoid layer, i.e. let $D_\phi(\boldsymbol{x}) = \sigma(h_\phi(\boldsymbol{x}))$. The values $h_\phi(\boldsymbol{x})$ are also called the discriminator's **logits**.

There are several common variants of the loss functions used to train GANs. They can all be described as a procedure where we alternately perform a gradient descent step on $L_D(\phi; \theta)$ with respect to ϕ to train the discriminator D_ϕ, and a gradient descent step on $L_G(\theta; \phi)$ with respect to θ to train the generator G_θ:

$$\min_\phi L_D(\phi; \theta), \qquad \min_\theta L_G(\theta; \phi).$$

In lecture, we talked about the following losses, where the discriminator's loss is given by

$$L_D(\phi; \theta) = -\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z} \sim \mathcal{N}(0,I)}[\log(1 - D_\phi(G_\theta(\boldsymbol{z})))],$$

and the generator's loss is given by the **minimax loss**

$$L_G^{\text{minimax}}(\theta; \phi) = \mathbb{E}_{\boldsymbol{z} \sim \mathcal{N}(0,I)}[\log(1 - D_\phi(G_\theta(\boldsymbol{z})))].$$

1. [**5 points**] Unfortunately, this form of loss for L_G suffers from a *vanishing gradient* problem. In terms of the discriminator's logits, the minimax loss is

   $$L_G^{\text{minimax}}(\theta; \phi) = \mathbb{E}_{\boldsymbol{z} \sim \mathcal{N}(0,I)}[\log(1 - \sigma(h_\phi(G_\theta(\boldsymbol{z}))))].$$

   Show that the derivative of L_G^{minimax} with respect to θ is approximately 0 if $D(G_\theta(\boldsymbol{z})) \approx 0$, or equivalently, if $h_\phi(G_\theta(\boldsymbol{z})) \ll 0$. You may use the fact that $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. Why is this problematic for the training of the generator when the discriminator successfully identifies a fake sample $G_\theta(\boldsymbol{z})$?

Solution: Using the chain rule and the fact that $\sigma'(x) = \sigma(x)(1 - \sigma(x))$,

$$\begin{aligned}
\frac{\partial L_G^{\text{minimax}}}{\partial \theta} &= \mathbb{E}_{\boldsymbol{z}\sim\mathcal{N}(0,I)}\Big[\frac{\sigma'(h_\phi(G_\theta(\boldsymbol{z})))}{1-\sigma(h_\phi(G_\theta(\boldsymbol{z})))}\frac{\partial}{\partial\theta}h_\phi(G_\theta(\boldsymbol{z}))\Big] \\
&= \mathbb{E}_{\boldsymbol{z}\sim\mathcal{N}(0,I)}\Big[\frac{\sigma(h_\phi(G_\theta(\boldsymbol{z})))(1-\sigma(h_\phi(G_\theta(\boldsymbol{z}))))}{1-\sigma(h_\phi(G_\theta(\boldsymbol{z})))}\frac{\partial}{\partial\theta}h_\phi(G_\theta(\boldsymbol{z}))\Big] \\
&= \mathbb{E}_{\boldsymbol{z}\sim\mathcal{N}(0,I)}\Big[\sigma(h_\phi(G_\theta(\boldsymbol{z})))\,\frac{\partial}{\partial\theta}h_\phi(G_\theta(\boldsymbol{z}))\Big] \\
&= \mathbb{E}_{\boldsymbol{z}\sim\mathcal{N}(0,I)}\Big[D_\phi(G_\theta(\boldsymbol{z}))\,\frac{\partial}{\partial\theta}h_\phi(G_\theta(\boldsymbol{z}))\Big].
\end{aligned}$$

When the discriminator successfully identifies a fake sample $G_\theta(\boldsymbol{z})$, it outputs $D_\phi(G_\theta(\boldsymbol{z})) \approx 0$, causing $\frac{\partial L_G^{\text{minimax}}}{\partial \theta} \approx 0$. Because the update to the generator's parameters is proportional to this gradient, the training of the generator slows or stops.

2. [**10 points**] Because of this vanishing gradient problem, in practice, L_G^{minimax} is typically replaced with the **non-saturating loss**

$$L_G^{\text{non-saturating}}(\theta;\phi) = -\mathbb{E}_{z\sim\mathcal{N}(0,I)}[\log D_\phi(G_\theta(\boldsymbol{z}))].$$

To turn the non-saturating loss into a concrete algorithm, we will take alternating gradient steps on Monte Carlo estimates of L_D and $L_G^{\text{non-saturating}}$:

$$\begin{aligned}
L_D(\phi;\theta) &\approx -\frac{1}{m}\sum_{i=1}^{m}\log D_\phi(\boldsymbol{x}^{(i)}) - \frac{1}{m}\sum_{i=1}^{m}\log(1 - D_\phi(G_\theta(\boldsymbol{z}^{(i)})), \\
L_G^{\text{non-saturating}}(\theta;\phi) &\approx -\frac{1}{m}\sum_{i=1}^{m}\log D_\phi(G_\theta(\boldsymbol{z}^{(i)})),
\end{aligned}$$

where m is the batch size, and for $i = 1, \ldots, m$, we sample $\boldsymbol{x}^{(i)} \sim p_{\text{data}}(\boldsymbol{x})$ and $\boldsymbol{z}^{(i)} \sim \mathcal{N}(0, I)$.

Implement and train a non-saturating GAN on Fashion MNIST for one epoch. Read through `run_gan.py`, and in `codebase/gan.py`, implement the `loss_nonsaturating` function. To train the model, execute `python run_gan.py`. You may monitor the GAN's output in the `out_nonsaturating` directory. Note that because the GAN is only trained for one epoch, we cannot expect the model's output to produce very realistic samples, but they should be roughly recognizable as clothing items.

Please package all of the programming parts (1.2, 3.2, and 4.5) together using `make_submission.sh` and submit the resulting ZIP file on GradeScope.

Solution:

```
def loss_nonsaturating(g, d, x_real, *, device):
    batch_size = x_real.shape[0]
    z = torch.randn(batch_size, g.dim_z, device=device)
    x_fake = g(z)
    d_real = d(x_real)
    d_fake = d(x_fake)

    # Either of the below are acceptable
    d_loss = -F.logsigmoid(d_real).mean() - F.logsigmoid(-d_fake).mean()
    d_loss = F.binary_cross_entropy_with_logits(d_real,
                 torch.ones(batch_size, device=device)) + \
             F.binary_cross_entropy_with_logits(d_fake,
                 torch.zeros(batch_size, device=device))
    g_loss = -F.logsigmoid(d_fake).mean()
    return d_loss, g_loss
```

(Note that $1 - \sigma(x) = \sigma(-x)$.)

Problem 2: Divergence minimization (25 points)

Now, let us analyze some theoretical properties of GANs. For convenience, we will denote $p_\theta(\boldsymbol{x})$ to be the distribution whose samples are generated by first sampling $\boldsymbol{z} \sim \mathcal{N}(0, I)$ and then returning the sample $G_\theta(\boldsymbol{z})$. With this notation, we may compactly express the discriminator's loss as

$$L_D(\phi; \theta) = -\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x} \sim p_\theta(\boldsymbol{x})}[\log(1 - D_\phi(\boldsymbol{x}))].$$

1. **[10 points]** Show that L_D is minimized when $D_\phi = D^*$, where

$$D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}.$$

(Hint: for a fixed $\boldsymbol{x}$, what t minimizes $f(t) = -p_{\text{data}}(\boldsymbol{x}) \log t - p_\theta(\boldsymbol{x}) \log(1 - t)$?)

Solution: Following the hint, we set $f'(t) = 0$ to obtain that the optimal t satisfies

$$0 = -\frac{p_{\text{data}}(\boldsymbol{x})}{t} + \frac{p_\theta(\boldsymbol{x})}{1 - t},$$

or, solving for t,

$$t = \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}.$$

This is the unique minimizer of f, since f is the sum of strictly convex functions.

Then, we can write

$$\begin{aligned} L_D(\phi; \theta) &= -\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x} \sim p_\theta(\boldsymbol{x})}[\log(1 - D_\phi(\boldsymbol{x}))] \\ &= -\int p_{\text{data}}(\boldsymbol{x}) \log D_\phi(\boldsymbol{x})\, d\boldsymbol{x} - \int p_\theta(\boldsymbol{x}) \log(1 - D_\phi(\boldsymbol{x}))\, d\boldsymbol{x} \\ &= \int f(D_\phi(\boldsymbol{x}))\, d\boldsymbol{x}. \end{aligned}$$

To minimize L_D, it suffices to minimize $f(D_\phi(\boldsymbol{x}))$ for every $\boldsymbol{x}$. We can do this by setting $D_\phi(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}$ for every $\boldsymbol{x}$.

2. **[5 points]** Recall that $D_\phi(\boldsymbol{x}) = \sigma(h_\phi(\boldsymbol{x}))$. Show that the logits $h_\phi(\boldsymbol{x})$ of the discriminator estimate the log of the likelihood ratio of $\boldsymbol{x}$ under the true distribution compared to the model's distribution; that is, show that if $D_\phi = D^*$, then

$$h_\phi(\boldsymbol{x}) = \log \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}.$$

Solution: Note that

$$D_\phi(\boldsymbol{x}) = \sigma(h_\phi(\boldsymbol{x})) = \frac{1}{1 + e^{-h_\phi(\boldsymbol{x})}}.$$

Setting this to the expression for $D^*(\boldsymbol{x})$ in part 1, we find that

$$1 + e^{-h_\phi(\boldsymbol{x})} = \frac{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})},$$

or

$$e^{-h_\phi(\boldsymbol{x})} = \frac{p_\theta(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x}) - p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})}.$$

Invert and take the logarithm of both sides to arrive at the result.

3. **[5 points]** Consider a generator loss defined by the sum of the minimax loss and the non-saturating loss,

$$L_G(\theta;\phi) = \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[\log(1 - D_\phi(\boldsymbol{x}))] - \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})].$$

Show that if $D_\phi = D^*$, then

$$L_G(\theta;\phi) = \mathrm{KL}(p_\theta(\boldsymbol{x})||p_{\mathrm{data}}(\boldsymbol{x})).$$

Solution:

$$\begin{aligned} L_G(\theta;\phi) &= \mathbb{E}_{p_\theta(\boldsymbol{x})}[\log(1 - D_\phi(\boldsymbol{x}))] - \mathbb{E}_{p_\theta(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})] \\ &= \mathbb{E}_{p_\theta(\boldsymbol{x})}\Big[\log\frac{1 - D_\phi(\boldsymbol{x}))}{D_\phi(\boldsymbol{x}))}\Big] \\ &= \mathbb{E}_{p_\theta(\boldsymbol{x})}\Big[\log\frac{p_\theta(\boldsymbol{x}))}{p_{\mathrm{data}}(\boldsymbol{x}))}\Big] \\ &= \mathrm{KL}(p_\theta(\boldsymbol{x})||p_{\mathrm{data}}(\boldsymbol{x})). \end{aligned}$$

4. **[5 points]** Recall that when training VAEs, we minimize the negative ELBO, an upper bound to the negative log likelihood. Show that the negative log likelihood, $-\mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{data}}(\boldsymbol{x})}[\log p_\theta(\boldsymbol{x})]$, can be written as a KL divergence plus an additional term that is constant with respect to θ. Does this mean that a VAE decoder trained with ELBO and a GAN generator trained with the L_G defined in the previous part are implicitly learning the same objective? Explain.

Solution: The VAE can be seen as minimizing an upper bound to

$$-\mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{data}}(\boldsymbol{x})}[\log p_\theta(\boldsymbol{x})] + \mathbb{E}_{x\sim p_{\mathrm{data}}(\boldsymbol{x})}[\log p_{\mathrm{data}}(\boldsymbol{x})] = \mathbb{E}_{p_\theta(\boldsymbol{x})}\Big[\log\frac{p_{\mathrm{data}}(\boldsymbol{x}))}{p_\theta(\boldsymbol{x}))}\Big] = \mathrm{KL}(p_{\mathrm{data}}(\boldsymbol{x})||p_\theta(\boldsymbol{x})).$$

However, in general, $\mathrm{KL}(p_{\mathrm{data}}(\boldsymbol{x})||p_\theta(\boldsymbol{x})) \neq \mathrm{KL}(p_\theta(\boldsymbol{x})||p_{\mathrm{data}}(\boldsymbol{x}))$, so the two objectives are not the same.

Problem 3: Conditional GAN with projection discriminator (20 points)

So far, we have trained GANs that sample from a given dataset of images. However, many datasets come with not only images, but also labels that specify the class of that particular image. In the MNIST dataset, we have both the digit's image as well as its numerical identity. It is natural to want to generate images that correspond to a particular class.

Formally, an *unconditional* GAN is trained to produce samples $\boldsymbol{x} \sim p_\theta(\boldsymbol{x})$ that mimic samples $\boldsymbol{x} \sim p_{\mathrm{data}}(\boldsymbol{x})$ from a data distribution. In the class-conditional setting, we instead have have labeled data $(\boldsymbol{x}, y) \sim p_{\mathrm{data}}(\boldsymbol{x}, y)$ and seek to train a model $p_\theta(\boldsymbol{x}, y)$. Since it is the class conditional generator $p_\theta(\boldsymbol{x}|y)$ that we are interested in, we will express $p_\theta(\boldsymbol{x}, y) = p_\theta(\boldsymbol{x}|y)p_\theta(y)$. We will set $p_\theta(\boldsymbol{x}|y)$ to be the distribution given by $G_\theta(\boldsymbol{z}, y)$, where $\boldsymbol{z} \sim \mathcal{N}(0, I)$ as usual. For simplicity, we will assume $p_{\mathrm{data}}(y) = \frac{1}{m}$ and set $p_\theta(y) = \frac{1}{m}$, where m is the number of classes. In this case, the discriminator's loss becomes

$$\begin{aligned} L_D(\phi;\theta) &= -\mathbb{E}_{(\boldsymbol{x},y)\sim p_{\mathrm{data}}(\boldsymbol{x},y)}[\log D_\phi(\boldsymbol{x}, y)] - \mathbb{E}_{(\boldsymbol{x},y)\sim p_\theta(\boldsymbol{x},y)}[\log(1 - D_\phi(\boldsymbol{x}, y))] \\ &= -\mathbb{E}_{(\boldsymbol{x},y)\sim p_{\mathrm{data}}(\boldsymbol{x},y)}[\log D_\phi(\boldsymbol{x}, y)] - \mathbb{E}_{y\sim p_\theta(y)}[\mathbb{E}_{\boldsymbol{z}\sim\mathcal{N}(0,I)}[\log(1 - D_\phi(G_\theta(\boldsymbol{z}, y), y))]]. \end{aligned}$$

Therefore, the main difference for the conditional GAN is that we must structure our generator $G_\theta(\boldsymbol{z}, y)$ and discriminator $D_\phi(\boldsymbol{x}, y)$ to accept the class label y as well. For the generator, one simple way to do so is to encode y as a one-hot vector $\boldsymbol{y}$ and concatenate it to $\boldsymbol{z}$, and then apply neural network layers normally. (A one-hot representation of a class label y is an m-dimensional vector $\boldsymbol{y}$ that is 1 in the yth entry and 0 everywhere else.)

In practice, the effectiveness of the model is strongly dependent on the way the discriminator depends on y. One heuristic with which to design the discriminator is to mimic the form of the theoretically optimal discriminator. That is, we can structure the neural network used to model D_ϕ based on the form of D^*, where D^* minimizes L_D. To calculate the theoretically optimal discriminator, though, it is necessary to make some assumptions.

1. **[10 points]** Suppose that when $(\boldsymbol{x}, y) \sim p_{\mathrm{data}}(\boldsymbol{x}, y)$, there exists a feature mapping φ under which $\varphi(\boldsymbol{x})$ becomes a mixture of m unit Gaussians, with one Gaussian per class label y. Assume that when $(\boldsymbol{x}, y) \sim$

$p_\theta(\boldsymbol{x}, y)$, $\varphi(\boldsymbol{x})$ also becomes a mixture of m unit Gaussians, again with one Gaussian per class label y. Concretely, we assume that the ratio of the conditional probabilities can be written as

$$\frac{p_{\text{data}}(\boldsymbol{x}|y)}{p_\theta(\boldsymbol{x}|y)} = \frac{\mathcal{N}(\varphi(\boldsymbol{x})|\boldsymbol{\mu}_y, I)}{\mathcal{N}(\varphi(\boldsymbol{x})|\hat{\boldsymbol{\mu}}_y, I)},$$

where $\boldsymbol{\mu}_y$ and $\hat{\boldsymbol{\mu}}_y$ are the means of the Gaussians for p_{data} and p_θ respectively.

Show that under this simplifying assumption, the optimal discriminator's logits $h^*(\boldsymbol{x}, y)$ can be written in the form

$$h^*(\boldsymbol{x}, y) = \boldsymbol{y}^T(A\varphi(\boldsymbol{x}) + \boldsymbol{b})$$

for some matrix A and vector $\boldsymbol{b}$, where $\boldsymbol{y}$ is a one-hot vector denoting the class y. In this problem, the discriminator's output and logits are related by $D_\phi(\boldsymbol{x}, y) = \sigma(h_\phi(\boldsymbol{x}, y))$. (Hint: use the result from problem 2.2.)

Solution:

$$\begin{aligned}
h_\phi(x, y) &= \log \frac{p_{\text{data}}(\boldsymbol{x}, y)}{p_\theta(\boldsymbol{x}, y)} \\
&= \log \frac{p_{\text{data}}(\boldsymbol{x}|y)}{p_\theta(\boldsymbol{x}|y)} + \log \frac{p_{\text{data}}(y)}{p_\theta(y)} \\
&= \log \frac{p_{\text{data}}(\boldsymbol{x}|y)}{p_\theta(\boldsymbol{x}|y)} \\
&= \log \frac{\exp(-\frac{1}{2}||\varphi(\boldsymbol{x}) - \boldsymbol{\mu}_y||^2)}{\exp(-\frac{1}{2}||\varphi(\boldsymbol{x}) - \hat{\boldsymbol{\mu}}_y||^2)} \\
&= -\frac{1}{2}||\varphi(\boldsymbol{x}) - \boldsymbol{\mu}_y||^2 + \frac{1}{2}||\varphi(\boldsymbol{x}) - \hat{\boldsymbol{\mu}}_y||^2 \\
&= -\frac{1}{2}||\varphi(\boldsymbol{x})||^2 - \boldsymbol{\mu}_y^T\varphi(\boldsymbol{x}) - \frac{1}{2}||\boldsymbol{\mu}_y||^2 + \frac{1}{2}||\varphi(\boldsymbol{x})||^2 + \hat{\boldsymbol{\mu}}_y^T\varphi(\boldsymbol{x}) + \frac{1}{2}||\hat{\boldsymbol{\mu}}_y||^2 \\
&= (\hat{\boldsymbol{\mu}}_y - \boldsymbol{\mu}_y)^T\varphi(\boldsymbol{x}) + \frac{1}{2}(||\hat{\boldsymbol{\mu}}_y||^2 - ||\boldsymbol{\mu}_y||^2) \\
&= \boldsymbol{y}^T\left(\begin{bmatrix} (\hat{\boldsymbol{\mu}}_1 - \boldsymbol{\mu}_1)^T \\ \vdots \\ (\hat{\boldsymbol{\mu}}_m - \boldsymbol{\mu}_m)^T \end{bmatrix}\varphi(\boldsymbol{x}) + \begin{bmatrix} \frac{1}{2}(||\hat{\boldsymbol{\mu}}_1||^2 - ||\boldsymbol{\mu}_1||^2) \\ \vdots \\ \frac{1}{2}(||\hat{\boldsymbol{\mu}}_m||^2 - ||\boldsymbol{\mu}_m||^2) \end{bmatrix}\right)
\end{aligned}$$

2. [**10 points**] Implement and train a conditional GAN on Fashion MNIST for one epoch. The discriminator has the structure described in part 1, with φ, A and b parameterized by a neural network with a final linear layer, and the generator accepts a one-hot encoding of the class. In `codebase/gan.py`, implement the `conditional_loss_nonsaturating` function. To train the model, execute `python run_conditional_gan.py`. You may monitor the GAN's output in the `out_nonsaturating_conditional` directory. You should be able to roughly recognize the categories that correspond to each column.

   Solution:

```
def conditional_loss_nonsaturating(g, d, x_real, *, device):
    batch_size = x_real.shape[0]
    z = torch.randn(batch_size, g.dim_z, device=device)
    x_fake = g(z, y_real)
    d_real = d(x_real, y_real)
    d_fake = d(x_fake, y_real)

    d_loss = -F.logsigmoid(d_real).mean() - F.logsigmoid(-d_fake).mean()
    g_loss = -F.logsigmoid(d_fake).mean()
    return d_loss, g_loss
```

Problem 4: Wasserstein GAN (35 points)

In many cases, the GAN algorithm can be thought of as minimizing a divergence between a data distribution $p_{\text{data}}(\boldsymbol{x})$ and the model distribution $p_\theta(\boldsymbol{x})$. For example, the minimax GAN discussed in the lectures minimizes the Jensen-Shannon divergence, and the loss in problem 2.3 minimizes the KL divergence. In this problem, we will explore an issue with these divergences and one potential way to fix it.

1. **[5 points]** Let $p_\theta(x) = \mathcal{N}(x|\theta, \epsilon^2)$ and $p_{\text{data}}(x) = \mathcal{N}(x|\theta_0, \epsilon^2)$ be normal distributions with standard deviation ϵ centered at $\theta \in \mathbb{R}$ and $\theta_0 \in \mathbb{R}$ respectively. Show that

$$\text{KL}(p_\theta(x)||p_{\text{data}}(x)) = \frac{(\theta - \theta_0)^2}{2\epsilon^2}.$$

Solution:

$$\begin{aligned}
\text{KL}(p_\theta(x)||p_{\text{data}}(x)) &= \mathbb{E}_{x\sim\mathcal{N}(\theta,\epsilon^2)}\Big[\log \frac{\exp(-\frac{1}{2\epsilon^2}(x-\theta)^2)}{\exp(-\frac{1}{2\epsilon^2}(x-\theta_0)^2)}\Big] \\
&= \mathbb{E}_{x\sim\mathcal{N}(\theta,\epsilon^2)}\Big[\frac{1}{2\epsilon^2}\big(-(x-\theta)^2 + (x-\theta_0)^2\big)\Big] \\
&= \mathbb{E}_{x\sim\mathcal{N}(\theta,\epsilon^2)}\Big[\frac{1}{2\epsilon^2}(2x\theta - 2x\theta_0 - \theta^2 + \theta_0^2)\Big] \\
&= \frac{1}{2\epsilon^2}(2\theta^2 - 2\theta\theta_0 - \theta^2 + \theta_0^2) \\
&= \frac{(\theta-\theta_0)^2}{2\epsilon^2}.
\end{aligned}$$

2. **[5 points]** Suppose $p_\theta(x)$ and $p_{\text{data}}(x)$ both place probability mass in only a very small part of the domain; that is, consider the limit $\epsilon \to 0$. What happens to $\text{KL}(p_\theta(x)||p_{\text{data}}(x))$ and its derivative with respect to θ, assuming that $\theta \neq \theta_0$? Why is this problematic for a GAN trained with the loss function L_G defined in problem 2.3?

Solution: Unless $\theta = \theta_0$, both $\text{KL}(p_\theta(x)||p_{\text{data}}(x))$ and its derivative go to infinity. If the discriminator is trained to optimality, the generator will receive extremely large gradients and hence training will be unstable.

3. **[5 points]** To avoid this problem, we'll propose an alternative objective for the discriminator and generator. Consider the following alternative objectives:

$$\begin{aligned}
L_D(\phi;\theta) &= \mathbb{E}_{x\sim p_\theta(x)}[D_\phi(x)] - \mathbb{E}_{x\sim p_{\text{data}}(x)}[D_\phi(x)] \\
L_G(\theta;\phi) &= -\mathbb{E}_{x\sim p_\theta(x)}[D_\phi(x)],
\end{aligned}$$

where D_ϕ is no longer constrained to functions that output a probability; instead D_ϕ can be a function that outputs any real number. As defined, however, these losses are still problematic. Again consider the limit $\epsilon \to 0$; that is, let $p_\theta(x)$ be the distribution that outputs $\theta \in \mathbb{R}$ with probability 1, and let $p_{\text{data}}(x)$ be the distribution that outputs $\theta_0 \in \mathbb{R}$ with probability 1. Why is there no discriminator D_ϕ that minimizes this new objective L_D?

Solution: Unless $\theta = \theta_0$, L_D can approach $-\infty$ by setting $D_\phi(\theta) \to -\infty$ or $D_\phi(\theta_0) \to \infty$. Therefore there is no discriminator that minimizes L_D.

4. **[5 points]** Let's tweak the alternate objective so that an optimal discriminator exists. Consider the same objective L_D and the same limit $\epsilon \to 0$. Now, suppose that D_ϕ is restricted to differentiable functions whose derivative is always between -1 and 1. It can still output any real number. Is there now a discriminator D_ϕ out of this class of functions that minimizes L_D? Briefly describe what the optimal D_ϕ looks like as a function of x.

Solution: Let $c = D_\phi(\theta_0)$. Because D_ϕ has slope between -1 and 1, the smallest $D_\phi(\theta)$ can be is $c - |\theta - \theta_0|$, which is achieved by setting D_ϕ to be a straight line of slope ± 1 connecting the points (θ_0, c) and $(\theta, c - |\theta - \theta_0|)$.

5. **[15 points]** The Wasserstein GAN with gradient penalty (WGAN-GP) enables stable training by penalizing functions whose derivatives are too large. It achieves this by adding a penalty on the 2-norm of the gradient of the discriminator at various points in the domain. It is defined by

$$\begin{aligned} L_D(\phi;\theta) &= \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[D_\phi(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[D_\phi(\boldsymbol{x})] + \lambda \mathbb{E}_{\boldsymbol{x}\sim r_\theta(\boldsymbol{x})}[(\|\nabla D_\phi(\boldsymbol{x})\|_2 - 1)^2] \\ L_G(\theta;\phi) &= -\mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[D_\phi(\boldsymbol{x})], \end{aligned}$$

where $r_\theta(\boldsymbol{x})$ is defined by sampling $\alpha \sim \text{Uniform}([0,1])$, $\boldsymbol{x}_1 \sim p_\theta(\boldsymbol{x})$, and $\boldsymbol{x}_2 \sim p_{\text{data}}(\boldsymbol{x})$, and returning $\alpha\boldsymbol{x}_1 + (1-\alpha)\boldsymbol{x}_2$. The hyperparameter λ controls the strength of the penalty; a setting that usually works is $\lambda = 10$.

Implement and train WGAN-GP for one epoch on Fashion MNIST. In `codebase/gan.py`, implement the `loss_wasserstein_gp` function. To train the model, execute `python run_gan.py --loss_type wasserstein_gp`. You may monitor the GAN's output in the `out_wasserstein_gp` directory.

Solution:

```
def loss_wasserstein_gp(g, d, x_real, *, device):
    batch_size = x_real.shape[0]
    z = torch.randn(batch_size, g.dim_z, device=device)
    x_fake = g(z)
    d_real = d(x_real)
    d_fake = d(x_fake)

    alpha = torch.rand(x_real.shape[0], 1, 1, 1, device=device)
    x_r = alpha * x_fake + (1 - alpha) * x_real
    d_r = d(x_r)

    grad = torch.autograd.grad(d_r.sum(), x_r, create_graph=True)
    grad_norm = grad[0].reshape(batch_size, -1).norm(dim=1)
    d_loss = (d_fake - d_real).mean() + 10 * ((grad_norm - 1)**2).mean()
    g_loss = -d_fake.mean()
    return d_loss, g_loss
```

Note the use of `d_r.sum()`. We are trying to obtain the following matrix of derivatives (we will then take the norm of each row, for use in the gradient penalty):

$$\texttt{grad} = \begin{bmatrix} \frac{\partial D(\boldsymbol{x}^{(1)})}{\partial \boldsymbol{x}^{(1)}} \\ \vdots \\ \frac{\partial D(\boldsymbol{x}^{(m)})}{\partial \boldsymbol{x}^{(m)}} \end{bmatrix}.$$

In principle, we could compute $\frac{\partial D(\boldsymbol{x}^{(i)})}{\partial \boldsymbol{x}^{(i)}}$ using a `for` loop over each element in the batch and then stack the resulting derivatives. However, this is inefficient. Instead, notice that

$$\frac{\partial D(\boldsymbol{x}^{(i)})}{\partial \boldsymbol{x}^{(i)}} = \frac{\partial}{\partial \boldsymbol{x}^{(i)}} \sum_{j=1}^{m} D(\boldsymbol{x}^{(j)}),$$

because each $D(\boldsymbol{x}^{(j)})$ is constant w.r.t. $\boldsymbol{x}^{(i)}$ for $i \neq j$. Therefore, if we let

$$X = \begin{bmatrix} \boldsymbol{x}^{(1)} \\ \vdots \\ \boldsymbol{x}^{(m)} \end{bmatrix},$$

we find that

$$\texttt{grad} = \frac{\partial}{\partial X} \sum_{j=1}^{m} D(\boldsymbol{x}^{(j)}).$$

Problem 5: Noise contrastive estimation (5 points)

Noise contrastive estimation is a technique to learn an energy-based model $p_\theta(\boldsymbol{x}) = \frac{1}{Z}\exp(-E_\theta(\boldsymbol{x}))$ to model a data distribution $p_{\text{data}}(\boldsymbol{x})$. (Recall that p_θ is only a proper probability distribution if $Z = \int \exp(-E_\theta(\boldsymbol{x}))\, d\boldsymbol{x}$.) Noise contrastive estimation (NCE) requires access to a second, tractable probabilistic model $p_{\text{noise}}(\boldsymbol{x})$ and optimizes the following objective:

$$\max_{\theta, Z} \mathbb{E}_{x \sim p_{\text{data}}(\boldsymbol{x})}[\log \sigma(-E_\theta(\boldsymbol{x}) - \log Z - \log p_{\text{noise}}(\boldsymbol{x}))] + \mathbb{E}_{x \sim p_{\text{noise}}(\boldsymbol{x})}[\log(1 - \sigma(-E_\theta(\boldsymbol{x}) - \log Z - \log p_{\text{noise}}(\boldsymbol{x})))].$$

1. **[5 points]** Let (θ^*, Z^*) be the parameters that maximize this objective. Show that $\frac{1}{Z^*}\exp(-E_{\theta^*}(\boldsymbol{x})) = p_{\text{data}}(\boldsymbol{x})$. (Hint: use the result from problem 2.2.)

   Solution: Using problem 2.2,

   $$-E_{\theta^*}(\boldsymbol{x}) - \log Z^* - \log p_{\text{noise}}(\boldsymbol{x}) = \log \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{noise}}(\boldsymbol{x})}.$$

   The results follows from exponentiating both sides.