CS 274E Homework 3

Deep Generative Models, CS 274E, Fall 2024

Due Date: 11:59pm **Thursday Nov 14**, submit via Gradescope

Instructions and Guidelines for Homeworks

- Please answer all of the questions and submit a scanned copy of your written solutions to Gradescope (either hand-written or typed are fine as long as the writing is legible).
- All homeworks will get equal weight in computation of the final grade for the class.
- The homeworks are intended to help you work through the concepts we discuss in class in more detail. It is important that you try to solve the problems yourself. The homework problems are important to help you better learn and reinforce the material from class.
- If you can't solve a problem, you can discuss it *verbally* with another student. However, please note that before you submit your homework solutions you are not allowed to view (or show to any other student) any *written material* directly related to the homeworks.
- You are allowed to use reference materials in your solutions, such as class notes, textbooks, other reference material (e.g., from the Web), or solutions to other problems in the homework. It is strongly recommended that you first try to solve the problem yourself, without resorting to looking up solutions elsewhere.
- Please submit your written solution to Gradescope, and all related code to Canvas. We will grade mostly based on the written solution, but will look into the code if necessary.

Problem 1: Implementing the Variational Autoencoder (VAE) (20 points)

For this problem we will be using PyTorch to implement the variational autoencoder (VAE) and learn a probabilistic model of the MNIST dataset of handwritten digits. Formally, we observe a vector of binary pixels $\mathbf{x} \in \{0,1\}^d$, and let $\mathbf{z} \in \mathbb{R}^k$ denote a set of latent variables. Our goal is to learn a latent variable model $p_\theta(\mathbf{x})$ of the high-dimensional data distribution $p_{\text{data}}(\mathbf{x})$.

A VAE is a latent variable model that fits a distribution $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int p(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z}$ to $p_{\text{data}}(\mathbf{x})$. Specifically, the VAE is defined by the following generative process:

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|0, \sigma_0^2 I); \tag{1}$$

$$p_\theta(\mathbf{x}|\mathbf{z}) = \text{Bern}(\mathbf{x}|f_\theta(\mathbf{z})) \tag{2}$$

where, without loss of generality, one usually sets $\sigma_0 = 1$. In other words, we assume that the latent variables $\mathbf{z}$ are sampled from a unit Gaussian distribution $\mathcal{N}(\mathbf{z}|0, I)$. The latent $\mathbf{z}$ are then passed through a neural network decoder $f_\theta(\cdot)$ to obtain the parameters of the d Bernoulli random variables which model the pixels in each image.

Although we would like to maximize the marginal likelihood $p_\theta(\mathbf{x})$ of the training data, computation of $p_\theta(\mathbf{x}) = \int p(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z}$ is generally intractable as it involves integration over all possible values of $\mathbf{z}$. Therefore, we posit a variational approximation q_ϕ to the true posterior and perform amortized inference as we have seen in class:

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mu_\phi(\mathbf{x}), \text{diag}(\sigma_\phi^2(\mathbf{x}))) \tag{3}$$

Specifically, we pass each image $\mathbf{x}$ through a neural network which outputs the mean μ_ϕ and diagonal covariance $\text{diag}(\sigma_\phi^2(\mathbf{x}))$ of the multivariate Gaussian distribution that approximates the distribution over latent variables $\mathbf{z}$ given $\mathbf{x}$. We then maximize the lower bound to the marginal log-likelihood to obtain an expression known as the **evidence lower bound** (ELBO):

$$\log p_\theta(\mathbf{x}) \geq \text{ELBO}(\mathbf{x}; \theta, \phi) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p(\mathbf{z}))) \tag{4}$$

Notice that the ELBO as shown on the right hand side of Eq. 4 decomposes into two terms: (1) **the reconstruction loss**: $-\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]$, and (2) **the Kullback-Leibler (KL) term**: $D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})))$.

Your objective is to implement the variational autoencoder by modifying `utils.py` and `vae.py`.

1. **[5 points]** Implement the reparameterization trick in the function `sample_gaussian` of `utils.py`. Specifically, your answer will take in the mean `m` and variance `v` of the Gaussian distribution $q_\phi(\mathbf{z}|\mathbf{x})$ and return a sample $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$.

2. **[5 points]** Next, implement `negative_elbo_bound` in the file `vae.py`. Several of the functions in `utils.py` will be helpful, so please check what is provided. Note that we ask for the *negative* ELBO, as PyTorch optimizers *minimize* the loss function. Additionally, since we are computing the negative ELBO over a mini-batch of data $\{\mathbf{x}^{(i)}\}_{i=1}^n$, make sure to compute the *average*

$-\frac{1}{n}\sum_{i=1}^{n}\text{ELBO}(\mathbf{x}^{(i)};\theta,\phi)$ over the mini-batch. Finally, note that the ELBO itself cannot be computed exactly since exact computation of the reconstruction term is intractable. Instead we ask that you estimate the reconstruction term via Monte Carlo sampling

$$-\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] \approx -\log p_\theta(\mathbf{x}|\mathbf{z}^{(1)}),$$

where $\mathbf{z}^{(1)} \sim q_\phi(\mathbf{z}|\mathbf{x})$ denotes a single sample. The function `kl_normal` in `utils.py` will be helpful. Note: `negative_elbo_bound` also expects you to return the *average* reconstruction loss and KL divergence.

3. **[5 points]** To test your implementation, run `python run_vae.py` to train the VAE. Once the run is complete (20000 iterations), it will output (assuming your implementation is correct): the average (1) negative ELBO, (2) KL term, and (3) reconstruction loss as evaluated on a test subset that we have selected. Report the three numbers you obtain as part of the write-up. Since we're using stochastic optimization, you may wish to run the model multiple times and report each metric's mean and corresponding standard error. (Hint: the negative ELBO on the test subset should be somewhere around 100.)

4. **[5 points]** Visualize 200 digits (generate a single image tiled in a grid of 10×20 digits) sampled from the fitted distribution $p_\theta(\mathbf{x})$.

Problem 2: Generative adversarial networks (10 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 $\boldsymbol{x} \in \mathbb{R}^n$. In contrast to a VAE (see Problem 1), a GAN learns an implicit distribution, i.e., while we can use a trained GAN to generate samples from the fitted model, we cannot evaluate (or maximize) the probability that the model assigns to an arbitrary test (or training) point. This rules out training by maximizing the marginal likelihood. Instead, we will implement 'adversarial training'.

We 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 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 $\mathbf{x} \sim p_{\text{data}}(\boldsymbol{x})$. A discriminator value close to 1 means that an image is very realistic whereas a value close to 0 means that it is obviously fake. Because its output is intended to be interpreted as a probability, the last layer of the discriminator is frequently the **sigmoid** function,

$$\sigma(\xi) = \frac{1}{1 + e^{-\xi}} \tag{5}$$

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

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

$$\min_\phi L_D(\phi; \theta), \qquad \min_\theta L_G(\theta; \phi). \tag{6}$$

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

$$L_D(\phi; \theta) = -\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D_\phi(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z} \sim \mathcal{N}(0,I)}[\log(1 - D_\phi(G_\theta(\boldsymbol{z})))], \tag{7}$$

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

$$L_G^{\text{minimax}}(\theta; \phi) = \mathbb{E}_{\boldsymbol{z} \sim \mathcal{N}(0,I)}[\log(1 - D_\phi(G_\theta(\boldsymbol{z})))]. \tag{8}$$

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

$$L_G^{\text{minimax}}(\theta; \phi) = \mathbb{E}_{\boldsymbol{z} \sim \mathcal{N}(0,I)}[\log(1 - \sigma(h_\phi(G_\theta(\boldsymbol{z}))))]. \tag{9}$$

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 $h_\phi(G_\theta(\boldsymbol{z})) \ll 0$. Why is this problematic for the training of the generator when the discriminator successfully identifies a sample $G_\theta(\boldsymbol{z})$ as fake?

Hint: You may want to start by showing that

$$1 - \sigma(\xi) = \sigma(-\xi) \qquad \text{and} \qquad \nabla_\xi \log(\sigma(\xi)) = \sigma(-\xi). \tag{10}$$

2. **[5 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 $L_G^{\text{non-saturating}}$:

$$L_D(\phi;\theta) \approx -\frac{1}{m}\sum_{i=1}^{m}\log D_\phi(\boldsymbol{x}^{(i)}) - \frac{1}{m}\sum_{i=1}^{m}\log(1-D_\phi(G_\theta(\boldsymbol{z}^{(i)}))), \tag{11}$$

$$L_G^{\text{non-saturating}}(\theta;\phi) \approx -\frac{1}{m}\sum_{i=1}^{m}\log D_\phi(G_\theta(\boldsymbol{z}^{(i)})), \tag{12}$$

where m is the minibatch 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 the code in `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 include the last generated images for this question (`out_nonsaturating\fake_0900.png`).