CS 446 / ECE 449 — Homework 5

your NetID here

Version 1.0

Instructions.

- Homework is due **Wednesday, November 16, at noon CST**; you have **3** late days in total for **all Homeworks**.
- The template for Q4 Diffusion is available at this link.
- The template for Q2 VAE Coding is available at this link.
- Everyone must submit individually at gradescope under `Homework 5`, `Homework 5 Diffusion Code`, and `Homework 5 VAE Code`.
- The "written" submission at `Homework 5` **must be typed**, and submitted in any format gradescope accepts (to be safe, submit a PDF). You may use LaTeX, markdown, google docs, MS word, whatever you like; but it must be typed!
- When submitting at `Homework 5`, gradescope will ask you to **mark out boxes around each of your answers**; please do this precisely!
- Please make sure your NetID is clear and large on the first page of the homework.
- Your solution **must** be written in your own words. Please see the course webpage for full **academic integrity** information. You should cite any external reference you use.
- We reserve the right to reduce the auto-graded score for `Homework 5 Code` if we detect funny business (e.g., your solution lacks any algorithm and hard-codes answers you obtained from someone else, or simply via trial-and-error with the autograder).
- When submitting to `Homework 5 VAE Code`, only upload `hw5_vae.py`. Additional files will be ignored.
- When submitting to `Homework 5 Diffusion Code`, only upload `hw5_diffusion.py`. Additional files will be ignored.

1. Variational Auto-Encoders [Written]

We use VAEs to learn the distribution of the data x. Let z denote the unobserved latent variable. We refer to the approximated posterior $q_\phi(z|x)$ as the encoder and to the conditional distribution $p_\theta(x|z)$ as the decoder. Use these names to answer the following questions.

(a) We are interested in modeling data $x \in \{0,1\}^G$. Hence, we choose $p_\theta(x|z)$ to follow G independent Bernoulli distributions. Recall, a Bernoulli distribution has a probability density function of

$$P(k) = \begin{cases} 1-p & \text{if } k=0 \\ p & \text{if } k=1 \end{cases}.$$

Write down the explicit form for $p_\theta(x|z)$. Use $\hat{y}_j$ to denote the $j^{\text{th}} \in [1, G]$ dimension of the decoder's output.

(b) We further assume that $z \in \mathbb{R}^2$ and that $q_\phi(z|x)$ follows a multi-variate Gaussian distribution with an identity covariance matrix. What is the output dimension of the encoder and why?

(c) We want to maximize the log-likelihood $\log p_\theta(x)$. To this end we introduce a joint distribution $p_\theta(x,z)$ and reformulate the log-likelihood via

$$\log p_\theta(x) = \log \sum_z q_\phi(z|x) \frac{p_\theta(x,z)}{q_\phi(z|x)}.$$

Use Jensen's inequality to obtain a bound on the log-likelihood and divide the bound into two parts, one of which is the Kullback-Leibler divergence $\mathrm{KL}(q_\phi(z|x), p(z))$.

(d) State at least two properties of the KL-divergence.

(e) Recall, the evidence lower bound (ELBO) of the log likelihood, $\log p_\theta(x)$, is

$$\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\phi(z|x)||p(z)). \tag{1}$$

We can also write the ELBO as

$$\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z) + \log p(z) - \log(q_\phi(z|x)]. \tag{2}$$

Practically, will training a VAE using the formulation in Eq. 1 be the same as the one in Eq. 2? If not, why use one formulation over another?

(f) Observe that the ELBO in Eq. 1 works for any q_ϕ distribution. Is it a good idea to choose $q_\phi(z|x) := \mathcal{N}(0, I)$? In other words, why is an encoder necessary?

(g) Let

$$q_\phi(z|x) = \frac{1}{\sqrt{2\pi\sigma_\phi^2}} \exp\left(-\frac{1}{2\sigma_\phi^2}(z-\mu_\phi)^2\right).$$

What is the value for the KL-divergence $\mathrm{KL}(q_\phi(z|x), q_\phi(z|x))$ and why?

(h) Further, let

$$p(z) = \frac{1}{\sqrt{2\pi\sigma_p^2}} \exp\left(-\frac{1}{2\sigma_p^2}(z-\mu_p)^2\right).$$

Note the difference of the means for $p(z)$ and $q_\phi(z|x)$ while their standard deviations are identical. Assume that $\sigma = \sigma_\phi = \sigma_p$. What is the value for the KL-divergence $\mathrm{KL}(q_\phi(z|x), p(z))$ in terms of μ_p, μ_ϕ and σ?

(i) Now, let $q_\phi(z|x)$ and $p(z)$ be arbitrary probability distributions. We want to find that $q_\phi(z|x)$ which maximizes

$$\sum_z q_\phi(z|x) \log p_\theta(x|z) - \mathrm{KL}(q_\phi(z|x), p(z))$$

subject to $\sum_z q_\phi(z|x) = 1$. Ignore the non-negativity constraints. State the Lagrangian and compute its stationary point, i.e., solve for $q_\phi(z|x)$ which depends on $p_\theta(x|z)$ and $p(z)$. Make sure to get rid of the Lagrange multiplier.

(j) Which of the following terms should $q_\phi(z|x)$ be equal to: (1) $p(z)$; (2) $p_\theta(x|z)$; (3) $p_\theta(z|x)$; (4) $p_\theta(x, z)$.

Solution.

2. Variational Auto-Encoders [Coding]

In this assignment, you will implement a Variational Autoencoder and train it on MNIST digits. Each datapoint x in the MNIST dataset is a 28×28 grayscale image (i.e., pixel values are between 0 and 1) of a handwritten digit in $\{0, \ldots, 9\}$, and a label indicating which number. The prior over each digit's latent representation z is a multivariate standard normal distribution, i.e., $z \sim \mathcal{N}(0, I)$. For all questions, we set the dimension of the latent space D_z to 2. Given the latent representation z for an image, the distribution over all 784 pixels in the image is given by a product of independent Bernoullis, whose characteristc probabilities are given by the output of a neural network $f_\theta(z)$ (the decoder):

$$p_\theta(x|z) = \prod_{d=1}^{784} \mathrm{Ber}(x_d | f_\theta(z)). \tag{3}$$

Relevant files: *HW5_vae.py*, *HW5_utils.py.*

(a) **Decoder Architecture**. Given a latent representation z, the decoder produces a 784-dimensional vector representing the Bernoulli distribution characteristic probability, i.e., the probability for every pixel in the image being labeled 1. Define the decoder parameters in the method *__init__* of the *Decoder* class and implement the corresponding *forward* function. The decoder architecture is a multi-layer perceptron (i.e., a fully-connected neural network), with two hidden layers, followed each by a non linearity: *tanh* after the first layer and *sigmoid* after the second layer. The hidden dimension is set to 500 units.

(b) **Distributions**.

  i. Implement the method *logpdf_diagonal_gaussian*, that given a latent representation z, a mean μ and the variance σ^2 outputs the log-likelihood of the normal distribution $\mathcal{N}(\mu, \sigma^2 I)$.

  ii. Implement a function *logpdf_bernoulli*, that given a sample x, a probability p outputs the log-likelihood of a Bernoulli distribution.

  iii. Implement the function *sample_diagonal_gaussian* which uses the reparametrization trick to sample z from Diagonal Gaussian $z \sim \mathcal{N}(\mu, \sigma^2 I)$.

  iv. Implement the function *sample_Bernoulli* which samples a configuration x from a Bernoulli distribution characterized by a probability p.

(c) **Variational Objective**. Complete the function *elbo* with the ELBO loss implementation corresponding to Eq. 2.

(d) **Training**. Train the model for 200 epochs. **Hint:** Run the *main* function and make sure the number of epochs is set-up correctly in *parse_args*.

(e) **Visualization**.

  i. **Samples from the generative model**. Complete the method *visualize_data_space* following the instructions:

  - Sample a z from the prior $p(z)$. Use *sample_diagonal_gaussian*.
  - Use the generative model to parameterize a Bernoulli distribution over x given z. Use *self.decoder* and *array_to_image*. Plot this distribution $p(x|z)$.
  - Sample x from the distribution $p(x|z)$. Plot this sample.
  - Repeat the steps above for 10 samples z from the prior. Concatenate all your plots into one 10×2 figure where the first column is the distribution over x and the second column is a sample from this distribution. Each row will be a new sample from the prior. Hint: use the function *concat_images*.
  - Attach the figure to your report.

  ii. **Latent space visualization**. Produce a scatter plot in the latent space, where each point in the plot represents a different image in the training set. Complete the method *visualize_latent_space* following the instructions:

- Encode each image in the training set. Use *self.encoder*.
- Plot the mean vector μ of $q_\phi(z|x)$ in the 2D latent space with a scatter plot. Make sure to color each point according to the class label (0 to 9).
- Attach the scatter plot to your report.

iii. **Interpolation between two classes**. Complete the method *visualize_inter_class_interpolation* following the instructions:

- Sample 3 pairs of data points (*self.train_images*) with different classes (*self.train_labels*).
- Encode the data in each pair, and take the mean vectors. Note that the encoder procuces a meam vector and a variance one.
- Interpolate between these mean vectors. We denote the output by z_α, with $\alpha \in [0, 1]$ and the interpolation step being 0.1. Hint: use the function *interpolate_mu*.
- Along the interpolation, plot the distributions $p(x|z_\alpha)$ in the same figure
- Use *concat_images* to concatenate these plots into one figure.
- Attach the plot to your report.

Solution.

3. Generative Adversarial Networks [Written]

Here we discuss distribution-comparison-related problems in Generative Adversarial Networks (GANs).

(a) What is the cost function for classical GANs? Use $D_\omega(x)$ as the discriminator and $G_\theta(x)$ as the generator.

(b) Assume arbitrary capacity for both discriminator and generator. In this case we refer to the discriminator using $D(x)$, and denote the distribution on the data domain induced by the generator via $p_G(x)$. State an equivalent problem to the one asked for in part (a), by using $p_G(x)$.

(c) Assuming arbitrary capacity, derive the optimal discriminator $D^*(x)$ in terms of $p_{\text{data}}(x)$ and $p_G(x)$.
Hint: you can think of fixing generator $G(\cdot)$ to find the optimal value for discriminator $D(\cdot)$.

(d) Assume arbitrary capacity and an optimal discriminator $D^*(x)$ from (c), show that the optimal generator $G^*(x)$ generates the distribution $p_G^* = p_{\text{data}}$, where $p_{\text{data}}(x)$ is the data distribution.
Hint: you may need the Jensen-Shannon divergence:

$$\text{JSD}(p_{\text{data}}, p_G) = \frac{1}{2}\text{KL}(p_{\text{data}}, M) + \frac{1}{2}\text{KL}(p_G, M),$$

where $M = \frac{1}{2}(p_{\text{data}} + p_G)$.

Solution.

4. Diffusion Model [Coding]

In this problem, you need to implement a Score-Matching model and use it to model a complicated distribution of points. The class `DiffusionModel` is used to represent the score function $s_\theta(x, \sigma)$.

A visualization of the dataset will appear when running the provided code.

(a) Implement the `ScoreMatching.denoising_loss` function following this equation:

$$\mathcal{L}(\theta) \triangleq \sum_x \sum_i^L \left[\left\| s_\theta(\tilde{x}, \sigma_i) + \frac{\tilde{x} - x}{\sigma_i^2} \right\|_2^2 \right],$$

where $x \sim p_{\text{data}}$ is a training sample; $\{\sigma_i\}_{i=1...L}$ is a set of predefined noise levels with $\sigma_1 < \sigma_2 ... < \sigma_L$; $\tilde{x} \sim \mathcal{N}(x, \sigma_i^2)$ is a noise-perturbed sample. The score function s_θ is trained to "denoise".

Implementation note: Instead of computing the loss at all noise levels which is slow, for each x, we can randomly sample a sigma and only evaluate the loss on that sigma. This allows us to remove the sum over L term. For every SGD training step, you can randomly select a noise level σ_i for each sample x. The `samples` argument has the shape (N, D) where $N = 2000$ (number of training samples) and $D = 2$. The predefined `sigma_chosen` variable also has shape (2000) and contains randomly sampled sigma values from `noise_levels` (i.e., $\{\sigma_i\}_{i=1}^L$).

(b) You are now ready to train the model. Run the main function and tune the `training_sigma` hyperparameter until the score function (plotted with `sm.plot_score()` looks good. **Attach a screenshot of it to the assignment submission**.

(c) Finally, implement Langevin Dynamics sampling with the formula:

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^L, \epsilon, T$.

Initialize $\tilde{x}_0$
for $i \leftarrow 1$ to L **do**
  $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$ ▷ α_i is the step size.
  for $t \leftarrow 1$ to T **do**
    Draw $z_t \sim \mathcal{N}(0, I)$
    $\tilde{x}_t \leftarrow \tilde{x}_{t-1} + \frac{\alpha_i}{2} s_\theta(\tilde{x}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\, z_t$
  end for
  $\tilde{x}_0 \leftarrow \tilde{x}_T$
end for
return $\tilde{x}_T$

Tune the `sampling_iterations` and `sampling_lr` hyperparameters in the main function until the generated samples roughly resemble the training data provided. Sample 2000 data samples (you can do it in a batch manner) and provide a scatter plot following the helper code.

Attach a screenshot of your generated samples to your submission.

Solution.