CS 446 / ECE 449 — Homework 5

your NetID here

Version 1.0

Instructions.

- Homework is due **Wednesday, Nov. 17th, at noon CST**; you have **3** late days in total for **all Homeworks**.
- Everyone must submit individually at gradescope under `hw5` and `hw5code`.
- The "written" submission at `hw5` **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 `hw5`, 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 `hw5code` 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 `hw5code`, only upload `hw5.py`. Additional files will be ignored.

1. Expectation Maximization

In this question, we expect you to do some computation related to the EM algorithm covered in the lecture.

Background. On the xy-plane, we have a rigid object, and our sensor can capture N key points, whose coordinates are: $\mathcal{P} = \{\boldsymbol{p}^{(1)}, \boldsymbol{p}^{(2)}, ..., \boldsymbol{p}^{(N)}\}$. ($N$ is a sufficiently large integer.) An unknown force then cause a translation $\mathbf{T}$ to this object, where $\mathbf{T}$ is encoded T_x and T_y, meaning how long the object has moved along the x-axis and y-axis. To calculate parameter $\mathbf{T}$, we use our sensor to capture the key points on the rigid object one more time, acquiring a set of N key points: $\mathcal{Q} = \{\boldsymbol{q}^{(1)}, \boldsymbol{q}^{(2)}, ..., \boldsymbol{q}^{(N)}\}$. (See Fig. 1 for an intuitive demonstration. The "L" is our rigid body, $N = 3$, and the blue and red dots are the key points.)

Assumption. During this process, we assume that an bi-jection mapping between $\mathcal{P}$ and $\mathcal{Q}$ exists (See Fig. 1, the key points are all the corners of "L.") However, this bijection mapping cannot be directly got from the sensors, as the orders of the points may be shuffled during perception.

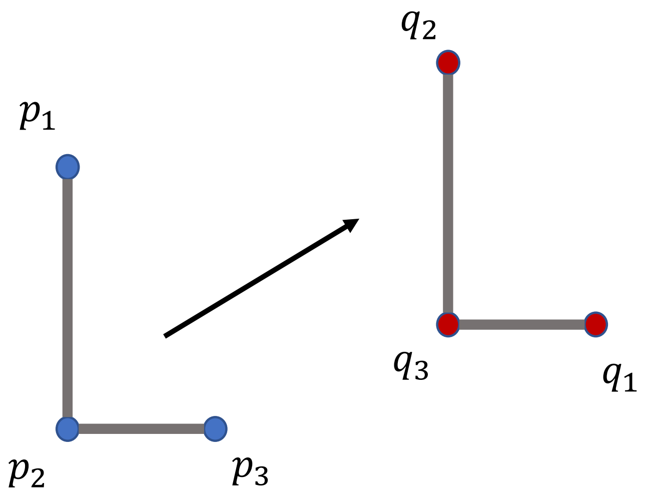

Figure 1

Objective. Using EM algorithm to estimate the translation parameter and the corresponding pairs of ke points, by introducing binary hidden variables $\boldsymbol{Z} \in \mathbb{R}^{N \times N}$, where $Z_{ij} = 1$ means $\boldsymbol{p}^{(i)}$ and $\boldsymbol{q}^{(j)}$ are a match.

Remark. We have the following remarks on this question.

- This questions set is for you to understand the process of EM with an intuitive example without annoying mathematical equations. However, to make it easy to understand, we use additional assumptions and simplifications. Please note the difference between this example and rigorous EM algorithm when you learn deeper machine learning courses in the future.
- You may find EM overkill for this simple question, and you are right about it. This question originates from a classical problem in computer vision called "point cloud registration," where this problem could be much more difficult when the sensor has noises and the bijection between $\mathcal{P}$ and $\mathcal{Q}$ does not exist. You may refer to "iterative closest points" (ICP) if you are interested in this.

(a) **Joint Probability.** If we know a pair of **matched** points $(\boldsymbol{p}^{(i)}, \boldsymbol{q}^{(j)})$, think of it as a single data sample, with the corresponding hidden state $Z_{ij} = 1$. Intuitively, based on this single pair, how likely parameter $\boldsymbol{T}$ is depends on $\|\boldsymbol{p}^{(i)} + \boldsymbol{T} - \boldsymbol{q}^{(j)}\|_2$. To make the math easier, we assume $\|\boldsymbol{p} + \boldsymbol{T} - \boldsymbol{q}\|_2$ follows from the Gaussian distribution $\mathcal{N}(0, \sigma)$ where σ is known, i.e.,

$$\mathbb{P}_{\boldsymbol{T}}((\boldsymbol{p}^{(i)}, \boldsymbol{q}^{(j)})|Z_{ij} = 1) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{\|\boldsymbol{p}^{(i)} + \boldsymbol{T} - \boldsymbol{q}^{(j)}\|_2^2}{2\sigma^2}) \tag{1}$$

(To avoid confusion in the notations, please use $\mathbb{P}$ to represent probability)

But we actually don't know which points are a match, or say we don't know the hidden states $\boldsymbol{Z}$, and want to use EM to help. We define that the matching is optimized from $\mathcal{P}$ to $\mathcal{Q}$, and not the reverse. In this way, the matching between each point $\boldsymbol{p} \in \mathcal{P}$ is independent, and two points $\boldsymbol{p} \in \mathcal{P}$ can match to the same $\boldsymbol{q} \in \mathcal{Q}$, but not the reverse.

Let $\mathbb{P}(Z_{ij} = 1) := \Pi_{ij}$ indicate the prior probability of $\boldsymbol{p}^{(i)}$ and $\boldsymbol{q}^{(j)}$ being a pair, where $\boldsymbol{\Pi} \in \mathbb{R}^{N\times N}$ are unknown parameters. Under our assumption, $0 \le \Pi_{ij} \le 1$, and $\sum_{k=1}^{N} \Pi_{ik} = 1, \forall\, i \in \{1, 2, ..., N\}$. Let us denote overall parameters $(\boldsymbol{T}, \boldsymbol{\Pi})$ as ψ like in the class.

Given a point $\boldsymbol{p}^{(i)}$, it has N possible matches $\boldsymbol{q}^{(j)}, j = 1, ..., N$. The probability contributed by $\boldsymbol{p}^{(i)}$ in the complete data likelihood is

$$\mathbb{P}(\boldsymbol{p}^{(i)}, \mathcal{Q}, \boldsymbol{Z}_{i,:}) = \mathbb{P}(\boldsymbol{p}^{(i)}, \mathcal{Q}|\boldsymbol{Z}_{i,:})\mathbb{P}(\boldsymbol{Z}_{i,:}) = \prod_{j=1}^{N} [\mathbb{P}_{\psi}((\boldsymbol{p}^{(i)}, \boldsymbol{q}^{(j)})|Z_{ij} = 1)\mathbb{P}(Z_{ij} = 1)]^{Z_{ij}} \tag{2}$$

Please write the log-likelihood $\log \mathbb{P}_{\psi}(\mathcal{P}, \mathcal{Q}, \boldsymbol{Z})$ **under the two following cases.**

i. General Case: Write out the general formula for $\log \mathbb{P}_{\psi}(\mathcal{P}, \mathcal{Q}, \boldsymbol{Z})$ following the "Log-likelihood of complete data" on Slides 7/27 in lecture 17.

ii. Special Case: $Z_{ii} = 1, i \in \{1, 2, 3.., N\}$. To get the full credits of this question, please full expand the Gaussian distributions.

(b) **Expectation. (1/2)** After doing the preparation work, we now start to apply EM algorithm to this question. In the E-step, our first objective is to compute $\boldsymbol{R}$, where $\boldsymbol{R} = [r_{ij}] \in \mathbb{R}^{N\times N}$, and $r_{ij} = \mathbb{P}_{\psi^{(t)}}(Z_{ij}|\boldsymbol{p}^{(i)}, \mathcal{Q})$.

Following the procedure of "the first line of the E-step on slides 8/27 in lecture 17," answer the following questions.

i. General Case: Derive the formula for r_{ij}. For the simplicity of notations, you can also use $\mathcal{N}(\cdot|\cdot,\cdot)$ like on the slides to denote Gaussian distribution.

ii. Special Case: For the N points in $\mathcal{P}$, the first $[\frac{N}{2}]$ points are matched with $\boldsymbol{q}^{(1)}$, and the rest are matched to $\boldsymbol{q}^{(2)}$. Write the values of $\boldsymbol{R}$.

(c) **Expectation. (2/2)** This question computes the values of $Q(\psi|\psi^{(t)})$. Please answer the following questions.

i. General Case: Derive the formula of $Q(\psi|\psi^{(t)})$ following the procedure on slides 8/27 in lecture 17. For the simplicity of notations, you can also use $\mathcal{N}(\cdot|\cdot,\cdot)$ to denote Gaussian distribution.

ii. Special Case: Same as the special case mentioned in problem "Expectation (1/2)," fully expand the formula of $Q(\psi|\psi^{(t)}$. To get full credits, your answer cannot have the variables r_{ij} and the notation for Gaussian distribution $\mathcal{N}(\cdot|\cdot,\cdot)$. (use $0 \times \log 0 = 0$ in your calculation)

(d) **Maximization.** On the basis of previous derivation, complete the maximization step and the update rule. Similar to the slides 9/27 in lecture 17, write out the formulas for $\boldsymbol{\Pi}^{(t+1)}$ and $\boldsymbol{T}^{(t+1)}$.

Hint. σ is fixed and you do not need to solve it.

i. General Case: Write the formulas for $\boldsymbol{\Pi}^{(t+1)}$ and $\boldsymbol{T}^{(t+1)}$. You may use N, $\boldsymbol{R}$, and the points in $\mathcal{P}$ and $\mathcal{Q}$.

ii. Special Case: Write the formulas for $\boldsymbol{T}^{(t+1)}$ for the special case in question "Expectation (1/2)." You may use N, and the points in $\mathcal{P}$ and $\mathcal{Q}$.

Solution.

2. Variational Auto-Encoders

We are training a variational auto-encoder (VAE). It contains the following parts: the input are vectors $\boldsymbol{x}$, the latent vector is $\boldsymbol{z}$, the encoder models the probability of $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, and the decoder is $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. Based on this notation, we will first look at several problems related to the structure of variational auto-encoder.

(a) We assume the latent vector $\boldsymbol{z} \in \mathbb{R}^2$ follows a multi-variate Gaussian distribution $\mathcal{N}$. Please compute the output dimension of the encoder $q_\phi(\cdot)$ under the following cases and briefly explain why. (If "output dimension" is not clear enough for you, think of it as "how many real numbers $r \in \mathbb{R}$ are needed to output for the sampling of latent vectors.")

- We assume $\mathcal{N}$ follows a multi-variate Gaussian distribution with an **identity matrix** as the covariance matrix.
- We assume $\mathcal{N}$ follows a multi-variate Gaussian distribution with an **diagonal matrix** as the covariance matrix.

(b) We then consider the problems related to the understanding of KL-Divergence.

i. Using the inequality of $\log(x) \le x - 1$, prove that $D_{KL}(p(x), q(x)) \ge 0$ holds for two arbitrary distributions $p(x)$ and $q(x)$.

ii. Consider a binary classification problem with input vectors $\boldsymbol{x}$ and labels $y \in \{0, 1\}$. The distribution of the ground truth label is denoted as $P(y)$. The expression of $P(y)$ is as Eq 3, where y_{gt} is the ground truth label.

$$P(y = y_{gt}) = 1, P(y = 1 - y_{gt}) = 0 \tag{3}$$

Suppose we are trying to predict the label of $\boldsymbol{x}$ with a linear model $\boldsymbol{w}$ and sigmoid function, then the distribution of y is denoted as $Q(y)$ and computed as Eq. 4.

$$Q(y = 0|\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^\top \boldsymbol{x})}, \quad Q(y = 1|\boldsymbol{x}) = \frac{\exp(-\boldsymbol{w}^\top \boldsymbol{x})}{1 + \exp(-\boldsymbol{w}^\top \boldsymbol{x})} \tag{4}$$

With the above information, compute the KL Divergence between the distributions of $P(y)$ and $Q(y|\boldsymbol{x})$, specifically $D_{KL}(P(y), Q(y|\boldsymbol{x})) = \mathbf{E}_{y\sim P(y)}[\log \frac{P(y)}{Q(y|\boldsymbol{x})}]$.
Expand your solution to the clearest form. To get full credits, your may only use $y_{gt}, \boldsymbol{w}, \boldsymbol{x}$ and related constants in your expression.

(c) VAE is a special branch of generative method in sampling the latent vectors $\widetilde{\boldsymbol{z}}$ from $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ instead of directly regressing the values of $\boldsymbol{z}$. Read an example implementation of VAE at `https://github.com/AntixK/PyTorch-VAE/blob/master/models/vanilla_vae.py` and answer the following questions:

i. Find the functions and lines related to the sampling of $\widetilde{\boldsymbol{z}}$ from $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. Specifying the names of the functions and the related lines can lead to full credits. Please note that if your range is too broad (in the extreme case, covering every line in the file) we cannot give your full credit.

ii. Suppose our latent variable is $\boldsymbol{z} \in \mathbb{R}^2$ sampled from a Gaussian distribution with mean $\boldsymbol{\mu} \in \mathbb{R}^2$ and a diagonal covariance matrix $\boldsymbol{\Sigma} = \texttt{Diag}\{\sigma_1^2, \sigma_2^2\}$. Then another random variable $\boldsymbol{v} \in \mathbb{R}^2$ is sampled from a Gaussian distribution $\mathcal{N}(0, \boldsymbol{I})$. Show that $\boldsymbol{V} = [\sigma_1, \sigma_2]^\top \circ \boldsymbol{v} + \boldsymbol{\mu}$ follows the same distribution as $\boldsymbol{z}$. ($\circ$ denotes Hadamard product, which means element-wide product; $\mathcal{N}(0, \boldsymbol{I})$ denotes the multi-variate Gaussian with zero mean and identity matrix as covariance.)

iii. Under the same setting of the Question ii, we can sample the latent vector $\widetilde{\boldsymbol{z}}$ by the process $\widetilde{\boldsymbol{z}} = [\sigma_1, \sigma_2]^\top \circ \widetilde{\boldsymbol{v}} + \boldsymbol{\mu}$, where $\widetilde{\boldsymbol{v}}$ is a sampled random variable from $\mathcal{N}(0, \boldsymbol{I})$. Consider the process of training, where we apply back-propagation to train the neural networks. Given the gradient on $\widetilde{\boldsymbol{z}}$ as $\widetilde{\boldsymbol{g}} \in \mathbb{R}^2$, which can be written as $[\widetilde{g}_1, \widetilde{g}_2]$. **What are the gradients of the output of the encoder: $\boldsymbol{\mu}, \sigma_1, \sigma_2$?** (Assume the KL-Divergence loss is not considered in this part.)
Note: To get full credit, you can use any constants and the variables of $\widetilde{\boldsymbol{v}} = [\widetilde{v}_1, \widetilde{v}_2]$, $\widetilde{\boldsymbol{g}} = [\widetilde{g}_1, \widetilde{g}_2]$, and $\boldsymbol{\mu}, \sigma_1, \sigma_2$.

iv. During reading the code, you might feel confused about why we are sampling $\widetilde{\boldsymbol{z}}$ in such a way, instead of generating a random value directly. But now, you could have some clues. Please briefly explain "Why we are sampling $\widetilde{\boldsymbol{z}}$ with $\mathcal{N}(0,1)$, instead of directly generating the values."

Solution.

3. Generative Adversarial Networks

Let's implement a Generative Adversarial Network(GAN) to create images of hand-written digits!

GAN consists of two parts: a generator network G and a discriminator network D. G is expected to generate a fake image from a random latent variable $\boldsymbol{z}$, and D is expected to distinguish fake images and real images. G and D are trained jointly with a minimax objective. In this question, we will use training data from MNIST to train our GAN, and let it produce some fake images that look like hand-written digits.

(a) First, let's implement the `Discriminator` network. It should take 32×32 gray-scale images as input, and output the probability of each image being a real one. Its architecture is summarized in Table 1.

Table 1: **Discriminator Architecture**

Layer Index	Layer Type	Input Channels	Output Channels	Kernel Size	Stride	Padding
1	Conv2d	1	16	3	1	1
2	LeakyReLU					
3	MaxPool			2	2	0
4	Conv2d	16	32	3	1	1
5	LeakyReLU					
6	MaxPool			2	2	0
7	Conv2d	32	64	3	1	1
8	LeakyReLU					
9	MaxPool			2	2	0
10	Conv2d	64	128	3	1	1
11	LeakyReLU					
12	MaxPool			4	4	0
13	Linear	128	1			
14	Sigmoid					

A few notes:

- All Conv2d and Linear layers have bias terms. You do not have to explicitly set `Conv2d(..., bias=True)`, since it is default in PyTorch.
- Also, you do not need to explicitly initialize the weights in Conv2d and Linear layers. The default initialization by PyTorch is good enough.
- LeakyReLU is a variant of ReLU activation, which has a smaller gradient for negative inputs. Set `negative_slope=0.2` for all LeakyReLU layers. More info about LeakyReLU at `https://pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html`.
- You need to reshape the tensor sometimes in the forward pass.
- Given a batch of images with shape `(batch_size, 1, 32, 32)`, the output of this network should be a tensor with shape `(batch_size)`, and the values in it are float numbers in $(0, 1)$. Our autograder will only be able to check the shape and range of the output, so be careful even if you have passed the test.

(b) Next, we can implement the `Generator` network. It should take 128-d vectors (sampled from a Gaussian distribution) as input, and output fake images. Its architecture is summarized in Table 2. We will make use of transposed convolutional layers. Given an input, a transposed convolutional layer can produce an output with a higher resolution. Thus, we can generate a 32×32 image from a vector by stacking such layers. A visualization of how transposed convolutional layers work can be found at `https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md`.

Table 2: **Generator Architecture**

Layer Index	Layer Type	Input Channels	Output Channels	Kernel Size	Stride	Padding
1	ConvTranspose2d	128	64	4	1	0
2	LeakyReLU					
3	ConvTranspose2d	64	32	4	2	1
4	LeakyReLU					
5	ConvTranspose2d	32	16	4	2	1
6	LeakyReLU					
7	ConvTranspose2d	16	1	4	2	1
8	Tanh					

A few notes:

- Again, all Conv2d and Linear layers have bias terms and are initialized by the default setup.
- Same LeakyReLU as above, with `negative_slope=0.2` for all LeakyReLU layers.
- You need to reshape the tensor sometimes in the forward pass.
- Given a batch of latent vectors with shape `(batch_size, 128)`, the output of this network should be a tensor with shape `(batch_size, 1, 32, 32)`, and the values in it are float numbers in $(-1, 1)$. Our autograder will only be able to check the shape and range of the output, so be careful even if you have passed the test.

(c) In class we have learned that to jointly train the generator and discriminator, we optimize them with a minimax objective:

$$V(G, D) := \frac{1}{N}\sum_{i=1}^{N} \log D(\boldsymbol{x}_i) + \frac{1}{N}\sum_{j=1}^{N} \log(1 - D(G(\boldsymbol{z}_j)))$$

$$\min_{G} \max_{D} V(G, D)$$

Here N is the batch size (set to 64 in our implementation), $\boldsymbol{x}_i$ is a real image, $\boldsymbol{z}_j$ is a random latent variable sampled from a Gaussian distribution, and $G(\boldsymbol{z}_j)$ is a fake image generated from it. Note that we are taking average to approximate the expectation, since we are using SGD to optimize G and D.

Please complete the function `calculate_V()` in `GAN`. You may (but not required to) use the binary cross entropy loss (see `https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html`) to simplify the implementation, but be careful about the sign and reduction method of BCELoss.

(d) We are ready to start training our GAN. The training pipeline is already provided in `train()`, and there is a `visualize()` function for your convenience. Train our GAN for 10 epochs, and **include the generated images after training in your PDF submission**.

Notes from TA:

- Training 10 epochs takes me about an hour on my laptop without GPU support. I can see interesting images after two or three epochs.
- You can make use of Google Colab(`https://colab.research.google.com/`), where you can access GPUs freely and accelerate the training. Remember to set `Runtime->Change runtime type->Hardware accelerator`.
- Some random seeds may lead to degenerated results. It's OK to try a few and manually set your random seed (`torch.manual_seed()`).

Solution.