EECS545 Machine Learning
Homework #5

Due date: Tuesday April 2, 2024 at 11:55pm

Reminder: While you are encouraged to discuss problems in a small group (up to 5 people), you should write your solutions and code **independently**. Do not share your solution with anyone else in the class. If you worked in a group for the homework, please include the names of your collaborators in the homework submission. Your answers (especially math derivaiton) should be as concise and clear as possible. Please address all questions to `http://piazza.com/class#winter2024/eecs545` with a reference to the specific question in the subject line (e.g., Homework 5, Q1(a): Can we assume...).
Submission Instruction: You should submit both **writeup** and **source code**. We may inspect your source code submission visually and rerun the code to check the results. Please be aware your points can be deducted if you don't follow the instructions listed below:

- Submit **writeup** to **Gradescope** (`https://www.gradescope.com/`)
  - Your writeup should contain your answers to **all** questions (typeset or hand-written), except for the implementation-only questions (marked with **(Autograder)**).
  - **For each subquestion, please select the associated pages from the pdf file in Gradescope. The graders are not required to look at pages other than the ones selected, and this may lead to some penalty when grading**.
  - Your writeup should be self-contained and self-explanatory. You should include the plots and figures in your writeup whenever asked, even when they are generated from the ipython notebook.
  - Please typeset (with LaTeX) or hand-write your solutions legibly. **Hand-writing that is difficult to read may lead to penalties**. If you prefer hand-writing, please use scanners or mobile scanning apps that provide shading corrections and projective transformations for better legibility; you **should *not*** just take photos and submit them without any image processing.
  - For all math derivations, you **should** provide detailed explanations as much as possible. If not, you may not be able to get a full credit if there are any mistakes or logical jumps.
- Submit **source code** to **Autograder** (`https://autograder.io/`)
  - Each ipython notebook file (i.e., `*.ipynb`) will walk you through the implementation task, producing the outputs and plots for *all* subproblems. You are required to write code on its matched python file (`*.py`). For instance, if you are working on `linear_regression.ipynb`, you are required to write code on `linear_regression.py`. Note that the outputs of your source code must match with your **writeup**.
  - We will read the data file in the `data` directory (i.e., `data/*.npy`) **from the same (current) working directory**: for example, `X_train = np.load('data/q3x.npy')`.
  - When you want to evaluate and test your implementation, please submit the `*.py` and `*.ipynb` files to Autograder for grading your implementations in the middle or after finish everything. You can partially test some of the files anytime, but please note that this also reduces your daily submission quota. You can submit your code up to **five** times per day.

– The autograder will give you information to help you troubleshoot your code. The following is the current list of error codes (will be updated accordingly if we encounter more common student implementation errors).

0:	Success
1:	The program exited with an unknown test failure.
2:	The program exited with an un-handled exception (usually a runtime error)
3-5:	Some other runtime error indicating an issue with the autograder code.
40:	NotImplementedError
50:	An uncategorized test failure different from the following error codes. Most likely indicates wrong outputs, but this may also include other assertion failures.
51:	There is an incorrect dtype on some output tensor.
52:	There is an incorrect shape on some output tensor.
53:	Some output tensors are not equal, or too different given the tolerance.
54:	Some output tensors are not equal with difference 1.0. Possibly due to rounding issues.
55:	The value of some input tensor was changed.
60:	The model's performance or converged loss is not good enough.
61:	The model might have diverged (e.g. encountered NaN or inf).
62-63:	The model has converged to a bad likelihood.

– Your program should run under an environment with following libraries:
  - Python 3.11 [1]
  - NumPy (for implementations of algorithms)
  - Matplotlib (for plots)
  - PyTorch (for implementing neural networks)

  Please do not use any other library unless otherwise instructed (no third-party libraries other than numpy and matplotlib are allowed). You should be able to load the data and execute your code on Autograder with the environment described above, but in your local working environment you might need to install them via `pip install numpy matplotlib torch torchvision`. For this assignment, you will need to submit the following files:
  - Q1 (KMeans, GMM)
    - `kmeans.py`
    - `gmm.py`
    - `kmeans_mmm.ipynb`
  - Q2 (EM)
    - No code submission
  - Q3 (PCA)
    - `pca.py`
    - `pca.ipynb`
  - Q4 (ICA)
    - `ica.py`
    - `ica.ipynb`
  - Q5 (CVAE)
    - `cvae.py`
    - `cvae.ipynb`

[1]We recommend using Miniconda (`https://docs.conda.io/en/latest/miniconda.html`). You can create `eecs545` environment with Python 3.11 as `conda create --name eecs545 python=3.11`. It is fine to use other python distributions and versions as long as they are supported, but we will run your program with Python 3.11 on Autograder.

Do not change the filename for the code and ipython notebook file because the Autograder may not find your submission. Please do not submit any other files except `*.py` and `*.ipynb` to Autograder.

– When you are done, please upload your work to Autograder (sign in with your UMich account). To receive the full credit, your code must run and terminate without error (i.e., with exit code 0) in the Autograder. Keep all the cell outputs in your notebook files (`*.ipynb`). We strongly recommend you run **Kernel → Restart Kernel and Run All Cells** (in the Jupyter Lab menu) before submitting your code to Autograder.

Change Log

- rev0 (2024/3/19): Initial Release

1 [25 points] K-means and GMM for image compression

In this problem, we will apply the K-means algorithm and Gaussian Mixture Models (GMM) to lossy image compression, by reducing the number of colors used in an image.

You are given two image files `mandrill-large.tiff` and `mandrill-small.tiff` of different resolution. Here, `mandrill-large.tiff` is a 512×512 image of a mandrill represented in 24-bit color. This means that, for each of the $512 \times 512 = 262,144$ pixels in the image, there are three 8-bit numbers (each ranging from 0 to 255) that represent the red, green, and blue intensity values for that pixel. The file `mandrill-small.tiff` contains a 128×128 version of `mandrill-large.tiff`. The straightforward representation of this image therefore takes about $262,144 \times 3 = 786,432$ bytes (a byte being 8 bits).

1.1 K-means Clustering

To compress the image, we will use K-means to reduce the image to $K = 16$ colors. More specifically, each pixel in the image is considered as a point in the three-dimensional (r, g, b)-space: $[0, 255]^3$. To compress the image, we will cluster these points in color-space into 16 clusters, and replace each pixel with the closest cluster centroid.

We use the Euclidean distance $\|x - y\|_2$ to measure the distance between two points x and y. In the pixel (RGB) space, the distance (or the *pixel error*) between (r, g, b) and (r', g', b') can be written as: $\sqrt{(r - r')^2 + (g - g')^2 + (b - b')^2}$.

Now, follow the instructions below.

(a) (6 pts) **(Autograder)** Start working on `kmeans_gmm.ipynb` and `kmeans.py`, and submit them when you are done implementing.

Treating each pixel's (r, g, b) values as an element of $\mathbb{R}^3$, implement and run K-means with 16 clusters on the pixel data from `mandrill-small.tiff` image, running 50 update steps. We provided the initial centroids as `initial_centroids` in the starter code to ensure a deterministic result.

You will need to implement (a general version of) K-means algorithm in `kmeans.train_kmeans()`, which will be graded with the provided sample data and some random data. In order to get full points, your implementation should be efficient and fast enough (otherwise you will get only partial points).

Hint: You may use `sklearn.metrics.pairwise_distances` function to compute the distance between centroids and data points, although it would be not difficult to implement this function (in a vectorized version) on your own.

(b) (3 pts) After training on the train image `mandrill-small.tiff`, read the test image `mandrill-large.tiff`, and replace each pixel's (r, g, b) values with the value of the closest cluster centroid.

Use the notebook's plotting code to display the original and compressed images side-by-side, and attach the plots to the **write-up**. (Note: you should have reasonable image quality/resolution to make the difference discernable). Also, measure and write down the mean pixel error between the original and compressed image.

(c) (1 pts) If we represent the image with these reduced 16 colors, by (approximately) what factor have we compressed the image (in terms of bits used to represent pixels) in terms of the data size? Include an explanation of why.

1.2 Gaussian Mixtures

Now let's repeat the same process with Gaussian mixtures (with *full* covariances), with $K = 5$ instead of K-means.

(d) (10 pts) **(Autograder)** Work on the notebook `kmeans_gmm.ipynb` to implement the EM algorithm for GMM. You will need to implement `gmm.train_gmm()` to train a GMM model, which will be graded with the provided sample data and some random data. In order to get full points, your implementation should be efficient and fast enough (otherwise you will get only partial points).

[Hint 1: You may use `scipy.stats.multivariate_normal()` to compute the *log*-likelihood of the data.] [Hint 2: You may use `scipy.special.logsumexp()` when computing $\gamma(z_{nk})$. You would need trick this because division by small probabilities can become computationally unstable when the likelihood values are too small. In practice, it is recommended to represent (possibly small) probabilities $\mathcal{N}(\mathbf{x}^{(n)} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ with log-likelihood. Note that $\mathcal{N}(\mathbf{x}^{(n)} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \exp[\log \mathcal{N}(\mathbf{x}^{(n)} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)]$.]

[Note: Do not use (and you don't need use) any other scipy or scikit-learn APIs.]

(e) (3 pts) Train Gaussian mixtures with $K = 5$ on the training data `mandrill-small.tiff`. Use the provided initial mean and covariance matrices, and prior distribution of latent cluster (`initial_mu`, `initial_sigma`, and `initial_pi`) to ensure deterministic output.

Report in the **write-up**: the log-likelihood of the training data after running 50 EM steps. In addition, report the GMM model parameters $\{(\pi_k, \boldsymbol{\mu}_k) : k = 1, ..., 5\}$. You do not need to write down $\boldsymbol{\Sigma}_k$. You can either write down the values, or simply attach the figure of visualization plots whose legend shows π_k and $\boldsymbol{\mu}_k$ values.

(f) (2 pts) After training on the train image `mandrill-small.tiff`, read the test image `mandrill-large.tiff` and replace each pixel's (r, g, b) values with the value of latent cluster mean, where we use the **MAP** (Maximum A Posteriori) estimation for the latent cluster-assignment variable for each pixel.

Use the notebook's plotting code to display the original and compressed images side-by-side, and attach the plots to the **write-up**. (Note: you should have reasonable image quality/resolution to make the difference discernable). Also, measure and write down the mean pixel error between the original and compressed image.

2 [20 points] Expectation Maximization for GDA with missing labels

In this problem, we will work on using the EM algorithm for Gaussian Discrimination Analysis (GDA) with missing labels.

Suppose that you are given dataset where some portion of the data is labeled and the other portion is unlabeled. We want to learn a generative model over this partially-labeled dataset.[2] In particular, suppose there are l examples with labels and u examples without labels, i.e., $\mathcal{D} = \left\{(\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(l)}, y^{(l)}), \mathbf{x}^{(l+1)}, \cdots, \mathbf{x}^{(l+u)}\right\}$.

We also make have the following assumptions:

- The data is real-valued and M-dimensional, i.e., $\mathbf{x} \in \mathbb{R}^M$
- The label y can take one of $\{0, 1\}$ (i.e., binary classification problem).
- We model the data following the same assumption as in Gaussian Discrimination Analysis, i.e.,

$$\begin{aligned} P(\mathbf{x}, y) &= P(y)P(\mathbf{x} \mid y) && (1)\\ P(y = j) &= \begin{cases} \phi & \text{if } j = 1 \\ 1 - \phi & \text{if } j = 0 \end{cases} && (2)\\ P(\mathbf{x} \mid y = j) &= \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j), j = 0 \text{ or } 1 && (3) \end{aligned}$$

where ϕ is a Bernoulli probability (i.e., $0 \leq \phi \leq 1$), and $\boldsymbol{\mu}_j$ and $\boldsymbol{\Sigma}_j$ are class-specific mean and covariance, respectively. For notational convenience, you can use $\phi_1 = \phi$ and $\phi_0 = 1 - \phi$.

Further, $\mathcal{N}(x; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ is the multivariate Gaussian distribution which is defined as:

$$p(\mathbf{x} \mid y = j; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) = \mathcal{N}(x; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) = \frac{1}{(2\pi)^{\frac{M}{2}} |\boldsymbol{\Sigma}_j|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_j)^\top \boldsymbol{\Sigma}_j^{-1} (\mathbf{x} - \boldsymbol{\mu}_j)\right) \tag{4}$$

Since we have unlabeled data, our goal is to maximize the following hybrid objective function:

$$\mathcal{J} = \sum_{i=1}^{l} \log p(\mathbf{x}^{(i)}, y^{(i)}) + \lambda \sum_{i=l+1}^{l+u} \log p(\mathbf{x}^{(i)}) \tag{5}$$

where λ is a hyperparameter that controls the weight of labeled and unlabeld data.

As we don't explicitly model the distribution $p(\mathbf{x})$, we use the law of total probability and rewrite the object function as:

$$\mathcal{J} = \sum_{i=1}^{l} \log p(\mathbf{x}^{(i)}, y^{(i)}) + \lambda \sum_{i=l+1}^{l+u} \log \sum_{j \in \{0,1\}} p(\mathbf{x}^{(i)}, y^{(i)} = j) \tag{6}$$

This way the unlabeled training examples is using the same models as the labeled samples. Now we will be using EM algorithm to optimize this objective function.

When deriving the solution, please show all the necessary steps in derivation and try to explain them to make the derivation as clearly understandable as possible.

(Hint) You can use the fact:

$$p(\mathbf{x}^{(i)}, y^{(i)}) = \prod_{j \in \{0,1\}} \left[\frac{\phi_j}{(2\pi)^{\frac{M}{2}} |\boldsymbol{\Sigma}_j|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}^{(i)} - \boldsymbol{\mu}_j)^\top \boldsymbol{\Sigma}_j^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_j)\right)\right]^{\mathbb{I}[y^{(i)} = j]} \tag{7}$$

[2]this type of learning formalism is called semi-supervised learning, which is a broad research field in machine learning.

2.1 Lower bound [3 points]

(a) Derive the variational lower bound $\mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \phi)$ of the objective $\mathcal{J}$. Specifically, show that any arbitrary probability distribution $q_i(y^{(i)} = j)$, the lower bound of the objective function can be written as:

$$\mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \phi) = \sum_{i=1}^{l} \log p(\mathbf{x}^{(i)}, y^{(i)}) + \lambda \sum_{i=l+1}^{l+u} \sum_{j \in \{0,1\}} Q_{ij} \log \frac{p(\mathbf{x}^{(i)}, y^{(i)} = j)}{Q_{ij}} \tag{8}$$

where $Q_{ij} \triangleq q_i(y^{(i)} = j)$ is a simplified shorthand notation. [Hint: you may want to use Jensen's Inequality.]

2.2 E-step [2 points]

(b) Write down the E-step; specifically, define the distribution $Q_{ij} = q_i(y^{(i)} = j)$.

2.3 M-step for $\boldsymbol{\mu}_k$ [6 points]

(c) Derive the M-step update rule for $\boldsymbol{\mu}_k$ where $k = 0$ or 1, while holding Q_i's (which you obtained in (a)) fixed. Also, explain in words (English) what *intuitively* $\boldsymbol{\mu}_k$ looks like in terms of $\mathbf{x}^{(i)}$'s (each of labeled and unlabeled) and pseudo-counts.

2.4 M-step for ϕ [6 points]

(d) Derive the M-step update rule for $\phi \in \mathbb{R}$, while holding Q_i's (which you obtained in (a)) fixed. Also, explain in words (English) what *intuitively* ϕ looks like in terms of $\mathbf{x}^{(i)}$'s (each of labeled and unlabeled) and pseudo-counts.

2.5 M-step for Σ_k [3 points]

(e) Finally, let's think about the M-step update rule for $\boldsymbol{\Sigma}_k$ where $k = 0$ or 1. Since we know the derivation is very similar to the case of GDA (and GMM M-step), we do not require you to repeat the similar the step as you have already worked on other two M-step update rules. Write down the M-step update rule for $\boldsymbol{\Sigma}_k$, without derivation, based on your guess and the analogy we have seen. Also, explain in words (English) what *intuitively* $\boldsymbol{\Sigma}_k$ looks like in terms of $\mathbf{x}^{(i)}$'s (each of labeled and unlabeled) and pseudo-counts.

3 [20 points] PCA and eigenfaces

(a) (8 pts) In lecture, we derived PCA from the "maximizing variance" viewpoint. In this problem, we will take the "minimizing squared error" viewpoint. Let $K \in \{1, \dots, D\}$ be arbitrary and let $\mathbf{x}^{(n)} \in \mathbb{R}^D$. Let $\mathcal{U} = \{\mathbf{U} = [\mathbf{u}_1 \cdots \mathbf{u}_K] \in \mathbb{R}^{D \times K} \mid \{\mathbf{u}_i\}_{i=1}^K\text{'s are orthonormal vectors}\}$, where $\mathbf{u}_i$ is the i-th column vector of $\mathbf{U}$.

Let's define the objective function for minimizing the distortion error:

$$\mathcal{J} = \frac{1}{N}\sum_{n=1}^{N} \left\| \mathbf{x}^{(n)} - \mathbf{U}\mathbf{U}^\top \mathbf{x}^{(n)} \right\|^2 = \frac{1}{N}\sum_{n=1}^{N} \left\| \mathbf{x}^{(n)} - \sum_{i=1}^{K} \mathbf{u}_i \mathbf{u}_i^\top \mathbf{x}^{(n)} \right\|^2 \tag{9}$$

Here, $\mathbf{U}\mathbf{U}^\top\mathbf{x}^{(n)}$ is called a projection of $\mathbf{x}^{(n)}$ into the subspace spanned by $\mathbf{u}_i$'s, and we can denote the projection $\widetilde{\mathbf{x}}^{(n)} = \mathbf{U}\mathbf{U}^\top\mathbf{x}^{(n)}$ as in the lecture.

Specifically, <u>show that</u>:

$$\mathcal{J} = \sum_{i=1}^{D} \lambda_i - \sum_{i=1}^{K} \mathbf{u}_i^\top \mathbf{S} \mathbf{u}_i \tag{10}$$

where $\mathbf{S}$ is the data covariance matrix $\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N} (\mathbf{x}^{(n)} - \overline{\mathbf{x}})(\mathbf{x}^{(n)} - \overline{\mathbf{x}})^\top$, $\overline{\mathbf{x}}$ is the data mean vector, and $\lambda_1 \geq \ldots \geq \lambda_d$ are the (ordered) eigenvalues of $\mathbf{S}$. Since the first term is a constant, the above equation implies that minimizing the squared error after projection is equivalent to maximizing the variance, as we have shown in the lecture slides.

With further simplification, <u>show that</u> the minimum distortion error corresponds to the sum of the $D-K$ smallest eigenvalues of $\mathbf{S}$, i.e.,

$$\min_{\mathbf{U} \in \mathcal{U}} \mathcal{J} = \sum_{k=K+1}^{D} \lambda_k. \tag{11}$$

and that the $\mathbf{u}_i$'s that minimize $\mathcal{J}$ are indeed the K eigenvectors of $\mathbf{S}$ corresponding to the (ordered) eigenvalues $\{\lambda_i\}_{k=1}^K$. After showing Eq.(10), it is okay to use the fact (without proof) that the optimal solution $\mathbf{u}_i$'s that maximize $\sum_{i=1}^K \mathbf{u}_i^\top \mathbf{S}\mathbf{u}_i$ is to pick the top-K eigenvectors of $\mathbf{S}$ (i.e., $\mathbf{u}_1, ..., \mathbf{u}_K$ corresponding to the largest K eigenvalues of $\mathbf{S}$ in descending order), as we already have seen in the lecture.

[Hint 1: You may assume that the data is zero-centered, i.e., $\overline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^N \mathbf{x}^{(n)} = \mathbf{0} \in \mathbb{R}^D$ without loss of generality. Be sure to mention it if you make such an assumption.]

[Hint 2: You can rewrite the objective as $\mathcal{J} = \dfrac{1}{N}\|\mathbf{X} - \mathbf{U}\mathbf{U}^\top\mathbf{X}\|_F^2$, where $\mathbf{X} \in \mathbb{R}^{D\times N}$ is the matrix that stacks all the data points $\{\mathbf{x}^{(n)}\}_{n=1}^N$ as column vectors, and the $\|\cdot\|_F$ denotes the Frobenious norm:

$$\|A\|_F = \sqrt{\sum_{i,j} (A_{ij})^2}, \tag{12}$$

and use the fact $\|A\|_F^2 = \mathrm{tr}(A^\top A)$ and $\mathrm{tr}(\mathbf{S}) = \sum_i \lambda_i$. Also note that there are many possible approaches for proving the claim, so you do not have to use this fact if you take a different approach.]

Now, you will apply PCA to face images. The principal components (eigenvectors) of the face images are called eigenfaces.

(b) (4 pts) **(Autograder)** Work on the provided code `pca.ipynb` and `pca.py` to implement PCA. Your code will be graded by the correctness on the sample face dataset and some other randomly-generated dataset.

(c) (3 pts) By regarding each image as a vector in a high dimensional space, perform PCA on the face images (sort the eigenvalues in descending order). In the write-up, report the eigenvalues corresponding to the first 10 principal components, and plot **all** the eigenvalues (in sorted order) where x-axis is the index of corresponding principal components and y-axis is the eigenvalue. Use log scale for the y-axis.

(d) (3 pts) Plot and attach to your write-up: a 2×5 array of subplots showing the first 10 principal components/eigenvectors ("eigenfaces") (sorted according to the descending eigenvalues) as images, treating the mean of images as the first principal component. Comment on what facial or lighting variations some of the different principal components are capturing (Note: you don't need to comment for all the images. Just pick a few that capture some salient aspects of image).

(e) (2 pts) Eigenfaces are a set of bases for all the images in the dataset (every image can be represented as a linear combination of these eigenfaces). Suppose we have L eigenfaces in total. Then we can use the L coefficients (of the bases, i.e. the eigenfaces) to represent an image. Moreover, we can use the first $K(< L)$ eigenfaces to reconstruct the face image approximately (correspondingly use K coefficients to represent the image). In this case, we reduce the dimension of the representation of images from L to K. To determine the proper K to use, we will check the percentage of variance that has been preserved (recall that the basic idea of PCA is preserving variance). Specifically, we define total variance

$$v(K) = \sum_{i=1}^{K} \lambda_i$$

where $1 \leq K < L$ and $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_L$ are eigenvalues. Then the percentage of total variance is

$$\frac{v(K)}{v(L)}$$

How many principal components are needed to represent 95% of the total variance? How about 99%? What is the percentage of reduction in dimension in each case?

4 [10 points] Independent Component Analysis

In this problem, you will implement maximum-likelihood Independent Component Analysis (ICA) for blind audio separation. As we learned in the lecture, the maximum-likelihood ICA minimizes the following loss:

$$\ell(W) = \sum_{i=1}^{N} \left(\sum_{j=1}^{m} \log g' \left(w_j^\top x^{(i)} \right) + \log |W| \right), \tag{13}$$

where N is the number of time steps, m is the number of independent sources, W is the transformation matrix representing a concatenation of w_j's, and $g(s) = 1/\left(1 + e^{-s}\right)$ is the sigmoid function. This link has some nice demos of blind audio separation: `https://cnl.salk.edu/~tewon/Blind/blind_audio.html`.

We provided the starter code `ica.py` and the data `ica_data.dat`, which contains mixed sound signals from multiple microphones. Run the provided notebook `ica.ipynb` to load the data and run your ICA implementation from `ica.py`.

(a) (6 points) (Autograder) Implement ICA by filling in the `ica.py` file.

(b) (4 points) Run your ICA implementation in the `ica.ipynb` notebook. To make sure your code is correct, you should listen to the resulting unmixed sources. (Some overlap in the sources may be present, but the different sources should be pretty clearly separated.)

Report the W matrix you found and submit the notebook `ica.ipynb` (along with `ica.py`) to the autograder. Make sure the audio tracks are audible in the notebook before submitting. You do **not** need to submit your unmixed sound files (`ica_unmixed_track_X.wav`).

5 [25 points] Conditional Variational Autoencoders

In this problem, you will implement a conditional variational autoencoder (CVAE) from [1] and train it on the MNIST dataset.

(a) **[5 points]** Derive the variational lower bound of a conditional variational autoencoder. Show that:

$$\begin{aligned}\log p_\theta(\mathbf{x}|\mathbf{y}) &\geq \mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{y}) \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}\left[\log p_\theta(\mathbf{x}|\mathbf{z}, \mathbf{y})\right] - D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y}) \| p_\theta(\mathbf{z}|\mathbf{y})\right), \end{aligned} \tag{14}$$

where $\mathbf{x}$ is a binary vector of dimension d, $\mathbf{y}$ is a one-hot vector of dimension c defining a class, $\mathbf{z}$ is a vector of dimension m sampled from the posterior distribution $q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y})$. The posterior distribution is modeled by a neural network of parameters ϕ. The generative distribution $p_\theta(\mathbf{x}|\mathbf{y})$ is modeled by another neural network of parameters θ. Similar to the VAE that we learned in the class, we assume the conditional independence on the componenets of $\mathbf{z}$: i.e., $q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y}) = \prod_{j=1}^m q_\phi(z_j|\mathbf{x}, \mathbf{y})$, and $p_\theta(\mathbf{z}|\mathbf{y}) = \prod_{j=1}^m p_\theta(z_j|\mathbf{y})$.

(b) **[8 points]** Derive the analytical solution to the KL-divergence between two Gaussian distributions $D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y}) \| p_\theta(\mathbf{z}|\mathbf{y})\right)$. Let us assume that $p_\theta(\mathbf{z}|\mathbf{y}) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and show that:

$$D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y}) \| p_\theta(\mathbf{z}|\mathbf{y})\right) = -\frac{1}{2}\sum_{j=1}^{m}\left(1 + \log\left(\sigma_j^2\right) - \mu_j^2 - \sigma_j^2\right), \tag{15}$$

where μ_j and σ_j are the outputs of the neural network that estimates the parameters of the posterior distribution $q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y})$.

You can assume without proof that

$$D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y}) \| p_\theta(\mathbf{z}|\mathbf{y})\right) = \sum_{j=1}^{m} D_{KL}\left(q_\phi(z_j|\mathbf{x}, \mathbf{y}) \| p_\theta(z_j|\mathbf{y})\right). \tag{16}$$

This is a consequence of conditional independence of the components of $\mathbf{z}$.

(c) **[12 points]** Fill in code for CVAE network as a `nn.Module` class called `CVAE` in the starter code `cvae.py` and the notebook `cvae.ipynb`:

- Implement the `recognition_model` function $q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y})$.
- Implement the `generative_model` function $p_\theta(\mathbf{x}|\mathbf{z}, \mathbf{y})$.
- Implement the `forward` function by inferring the Gaussian parameters using the recognition model, sampling a latent variable using the reparametrization trick and generating the data using the generative model.
- Implement the variational lowerbound `loss_function` $\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{y})$.
- Train the CVAE and visualize the generated image for each class (i.e., 10 images per class).
- Repeat the image generation 10 times with different random noise. In the write-up, attach and submit 10 $\times$ 10 array of images showing all the generated images, where the images in the same row are generated from the same random noise, and images in the same column are generated from the the same class label.
- The hyperparameters and training setups provided in the code should work well for learning a CVAE on the MNIST dataset, but please feel free to make any changes as needed and you think appropriate to make CVAE work. Please discuss (if any) there are some notable changes you have made.

If trained successfully, you should be able to sample images $\mathbf{x}$ that look like MNIST digits reflecting the given label $\mathbf{y}$, and the noise vector $\mathbf{z}$.

References

[1] Kihyuk Sohn, Xinchen Yan, and Honglak Lee. Learning structured output representation using deep conditional generative models. In *NeurIPS*. 2015.