1. [25 points] K-means and GMM for Image Compression

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

input image files:

1. `mandrill-large.tiff`, $512 \times 512 = 262144$ pixels size, 24-bit color (3×8 bits color channels, 8 bits represent 0~255, RGB colors).

   So the size of each picture is 262144×3 B
2. `mandrill-small.tiff`: 128×128 pixels version of `mandrill-large.tiff`.

1.1 K-means

(a) (auto) implement k-means

work on `keans_gmm.ipynb` 和 `kmean.py`

Treat every pixel as $(r, g, b) \in \mathbb{R}^3$, implement and run k-means with 16 clusters, on `mandrill-small.tiff`, running 50 updates steps.

Initial centriods is in `initial_centrids` , so the result is deterministic.

We will 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 not be difficult to implement this function (in a vectorized version) on your own.

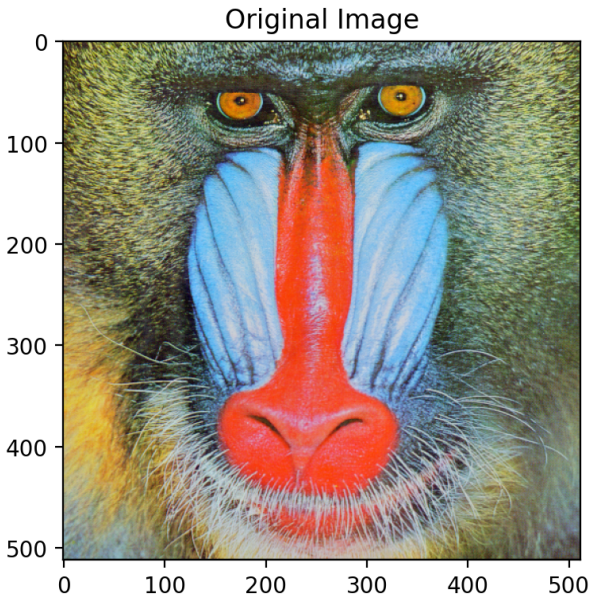
(b) test on `mandrill-large.tiff`

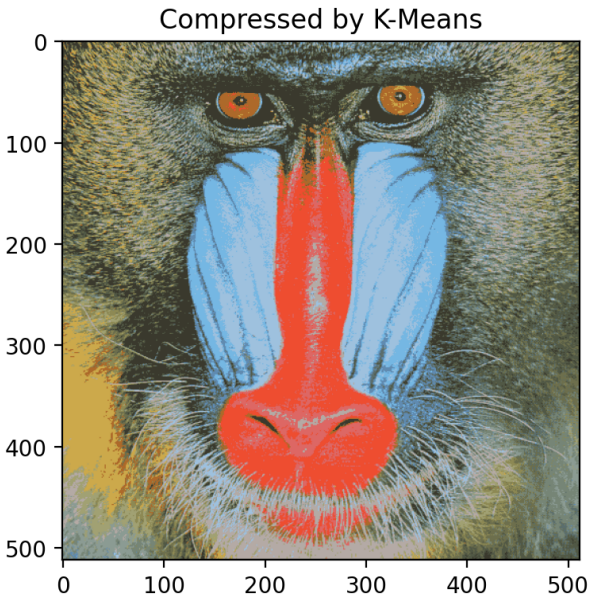
After training, we test on `mandrill-large.tiff`, replace (r, g, b) of every pixel with the value of the closest cluster centroid.

Attach the plots original and compressed images side-by-side to the write-up. (Note: you should have reasonable image quality/resolution to make the difference discernible).

Also, measure and write down the **mean pixel error** between the original and compressed image.

> **Sol**:
>
> Plots as below





mean pixel error as below:

```
# ~~START DELETE~~
kmeans.euclidean_distance(img_compressed, img_large).mean()
# ~~END DELETE~~
```

✓ 0.0s Python

```
14.60802961518135
```

(c) explain: what factor have we compressed

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.

Sol:

Originally we have 256^3 possible number of (r, g, b) colors to represent, taking over $3 \log_2 256 = 5128 \times 3 = 24$ Bits, for one pixel.

And now we have 16 colors, taking over $\log_2 16 = 4$ bits, for one pixel.

Storing the information of the 16 colors as basis, takes over $16 \times 24 = 384$ bits, (this amount is quite little compared to the size of an image.)

Thus the factor of compression is approximately:

$$= \frac{24 \times 512^2}{4 \times 512^2 + 384} \approx 6$$

1.2 Gaussian Mixtures

Now we use GMM to repeat the clustering task, setting $K = 5$.

(d) implement the EM algorithm for GMM

(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 $\gamma\left(z_{nk}\right)$. 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 $\mathcal{N}\left(\mathbf{x}^{(n)} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)$ with log-likelihood. Note that

$$N(x^{(n)} \mid \mu_k, \Sigma_k) = \exp[\log N(x^{(n)} \mid \mu_k, \Sigma_k)]$$

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

> **E-step** (compute responsibilities):
>
> For each point x_n, and cluster k:
>
> $$\gamma(z_{nk}) = \frac{\pi_k \cdot \mathcal{N}(x_n \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}(x_n \mid \mu_j, \Sigma_j)}$$
>
> Use **log-space trick**:
>
> - compute `log p(x_n | k)` using `scipy.stats.multivariate_normal.logpdf`
> - then compute log-sum-exp over $k$for normalization (using `scipy.special.logsumexp`)
> - exponentiate to get $\gamma(z_{nk})$
>
> **M-step** (update parameters):
>
> Let $N_k = \sum_n \gamma(z_{nk})$
>
> - $\pi_k = \frac{N_k}{N}$
> - $\mu_k = \frac{1}{N_k} \sum_n \gamma(z_{nk}) x_n$
> - $\Sigma_k = \frac{1}{N_k} \sum_n \gamma(z_{nk})(x_n - \mu_k)(x_n - \mu_k)^T$

(e) train GMM on `mandrill-small.tiff` image

(3 points) Train GMM on `mandrill-small.tiff` using $K = 5$. Provided initial parameters:

- initial mean（`initial_mu`）
- covariance matrices（`initial_sigma`）
- prior distribution of latent cluster（`initial_pi`）

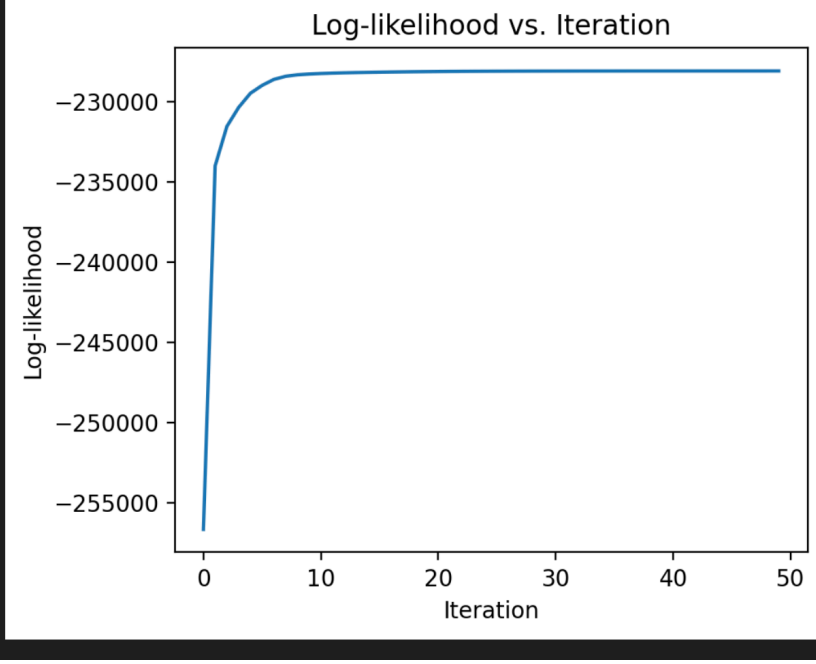
report:

- log-likelihood of training data after running 50 EM steps
- Parameters $\{(\pi_k, \mu_k) \mid k = 1, \ldots, 5\}$

You do not need to write down Σ_k. You can choose either write down the values, or attach visualization plots.

> log-likelihood of training data after running 50 EM steps:

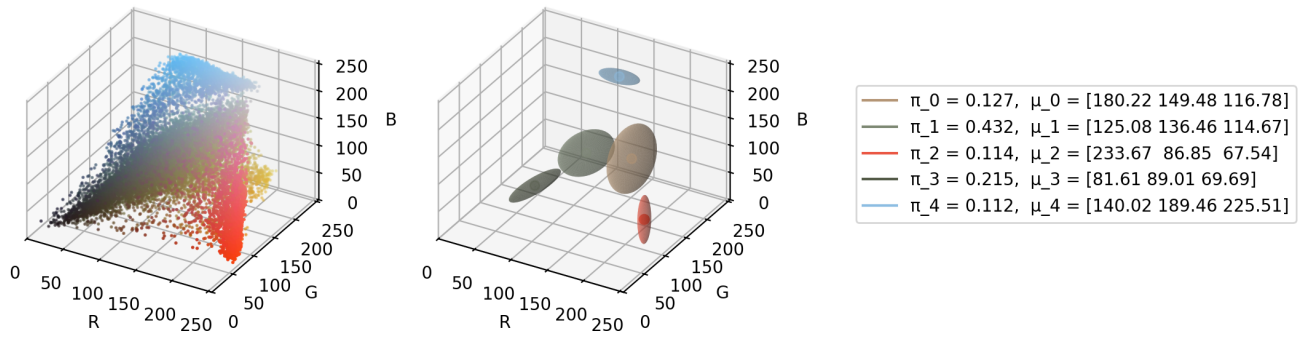
```
log likelihood after running 50 iterations:  -228103.42574313737
```



> Parameter values:

```
pi = array([0.13, 0.43, 0.11, 0.21, 0.11])
mu = array([[180.22, 149.48, 116.78],
       [125.08, 136.46, 114.67],
       [233.67,  86.85,  67.54],
       [ 81.61,  89.01,  69.69],
       [140.02, 189.46, 225.51]])
sigma = array([[[ 521.36,  290.61, -210.66],
        [ 290.61, 1047.03,  842.88],
        [-210.66,  842.88, 2511.74]],

       [[ 742.09,  526.24,   40.68],
        [ 526.24,  607.66,  454.3 ],
        [  40.68,  454.3 , 1226.68]],

       [[ 129.72, -108.61, -238.98],
        [-108.61,  259.27,  460.1 ],
        [-238.98,  460.1 , 1035.17]],

       [[ 459.17,  518.45,  324.74],
        [ 518.45,  692.35,  486.85],
        [ 324.74,  486.85,  463.54]],

       [[ 612.96,  146.43,  -90.39],
        [ 146.43,   89.24,   32.62],
        [ -90.39,   32.62,   82.26]]])
```

Plots:



(f) test on `mandrill-large.tiff` image

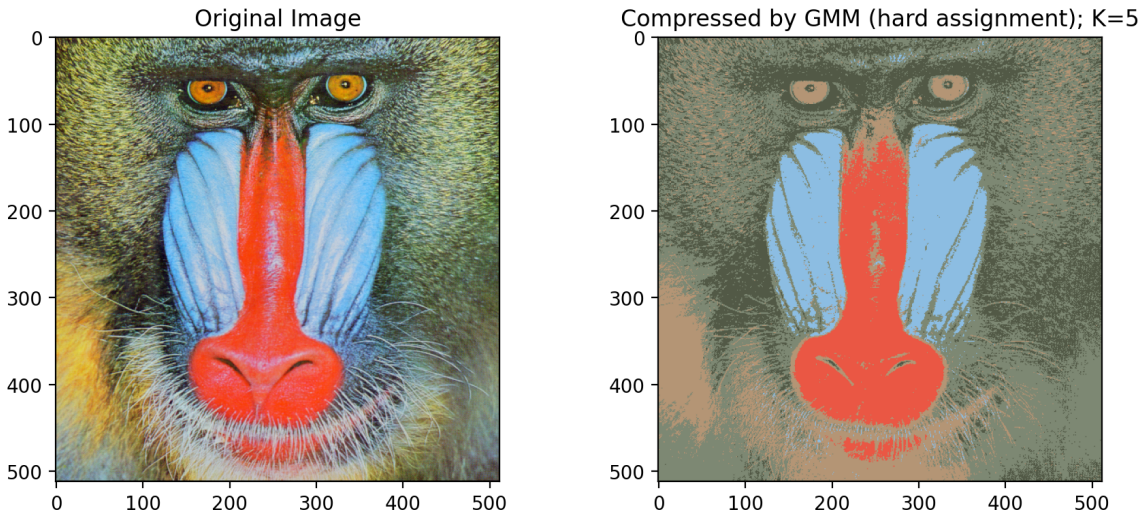
(2 pts) After training on the train `image mandrill-small.tiff`, read the test image `mandrill-large.tiff` and replace each pixel's (r, g, b) values with the value of latent cluster mean, where we use the MAP (Maximum A Posteriori) estimation for the latent cluster-assignment variable for each pixel.

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.

Sol:

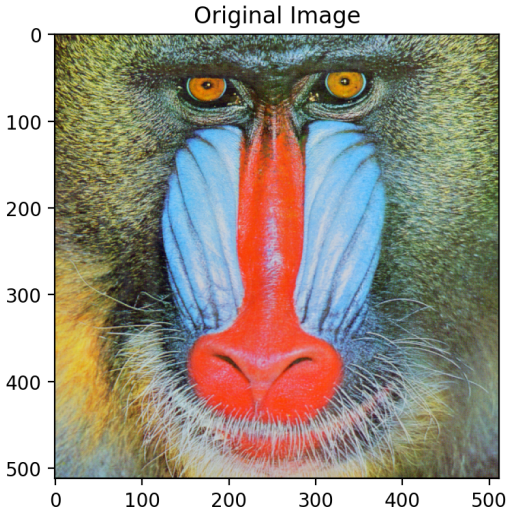
Plot:

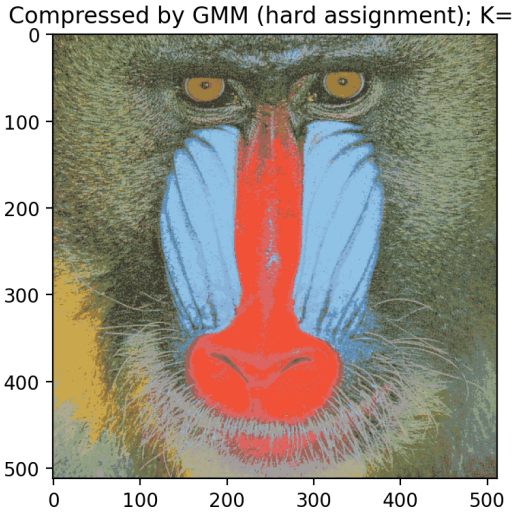


mean pixel error between the original and compressed image:

```
16.365135581692456
```

Plot for $K = 16$:


Original Image
0
100
200
300
400
500
0
100
200
300
400
500


Compressed by GMM (hard assignment); K=16
0
100
200
300
400
500
0
100
200
300
400
500

2. [20 points] EM 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]

> This type of learning formalism is called semi-supervised learning, which is a broad research field in machine learning.

In particular, suppose there are l examples with labels and u examples without labels, i.e.,
$\mathcal{D} = \left\{\left(\mathbf{x}^{(1)}, y^{(1)}\right), \cdots, \left(\mathbf{x}^{(l)}, y^{(l)}\right), \mathbf{x}^{(l+1)}, \cdots, \mathbf{x}^{(l+u)}\right\}$.

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.,

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

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

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

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

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

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:

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

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.

(Hint) You can use the fact:

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

(a) lower bound derivation [3 points]

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

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

where $Q_{ij} \triangleq q_i \left(y^{(i)} = j\right)$ is a simplified shorthand notation.

[**Hint: you may want to use Jensen's Inequality.**]

> Proof:
>
> The objective is:
>
> $$\mathcal{J} = \sum_{i=1}^{l} \log p\left(\mathbf{x}^{(i)}, y^{(i)}\right) + \lambda \sum_{i=l+1}^{l+u} \log \sum_{j \in \{0,1\}} p\left(\mathbf{x}^{(i)}, y^{(i)} = j\right)$$
>
> We define a distribution over the latent variable $y^{(i)} (l+1 \leq i \leq l+u)$ as:
>
> $$q_i\left(y^{(i)} = j\right) = Q_{ij}, \quad j \in \{0, 1\}, \quad \text{where} \sum_{j=0}^{1} Q_{ij} = 1$$
>
> Since $\log$ is a concave function, $-\log$ is convex. Then by Jensen's ineq we have:
>
> $$-\log \mathbb{E}[X] \leq \mathbb{E}[-\log X]$$
>
> Writing the second term into expectation form, we have:
>
> $$\begin{aligned}
> \log \sum_{j \in \{0,1\}} p\left(\mathbf{x}^{(i)}, y^{(i)} = j\right) &= \log \sum_{j \in \{0,1\}} Q_{ij} \cdot \frac{p\left(\mathbf{x}^{(i)}, y^{(i)} = j\right)}{Q_{ij}} \\
> &= \log \sum_{j \in \{0,1\}} q\left(y^{(i)}\right) \cdot \frac{p\left(\mathbf{x}^{(i)}, y^{(i)} = j\right)}{q\left(y^{(i)}\right)} \\
> &= \log \mathbb{E}_{y^{(i)} \sim q}\left[\frac{p(\mathbf{x}, y)}{q(y)}\right] \\
> &= -\left(-\log \mathbb{E}_{y^{(i)} \sim q}\left[\frac{p(\mathbf{x}, y)}{q(y)}\right]\right)
> \end{aligned}$$
>
> Since by Jensen's ineq we have:
>
> $$-\log \mathbb{E}_{y^{(i)} \sim q}\left[\frac{p(\mathbf{x}, y)}{q(y)}\right] \leq \mathbb{E}_{y^{(i)} \sim q}\left[-\log \frac{p(\mathbf{x}, y)}{q(y)}\right]$$
>
> Then reversing it by adding a negative sign:
>
> $$-\left(-\log \mathbb{E}_{y^{(i)} \sim q}\left[\frac{p(\mathbf{x}, y)}{q(y)}\right]\right) \geq -\mathbb{E}_{y^{(i)} \sim q}\left[-\log \frac{p(\mathbf{x}, y)}{q(y)}\right] = \mathbb{E}_{y^{(i)} \sim q}\left[\log \frac{p(\mathbf{x}, y)}{q(y)}\right]$$
>
> Thus
>
> $$\begin{aligned}
> \log \sum_{j \in \{0,1\}} p\left(\mathbf{x}^{(i)}, y^{(i)} = j\right) &\geq \mathbb{E}_{y \sim q(y)}\left[\log \frac{p(\mathbf{x}, y)}{q(y)}\right] \\
> &= \sum_{j \in \{0,1\}} Q_{ij} \log \frac{p\left(\mathbf{x}^{(i)}, y^{(i)} = j\right)}{Q_{ij}}
> \end{aligned}$$
>
> Since this holds for each , we have:
>
> $$\mathcal{J} \geq \sum_{i=1}^{l} \log p\left(\mathbf{x}^{(i)}, y^{(i)}\right) + \lambda \sum_{i=l+1}^{l+u} \sum_{j \in \{0,1\}} Q_{ij} \log \frac{p\left(\mathbf{x}^{(i)}, y^{(i)} = j\right)}{Q_{ij}}$$

We define

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

Then we establish the (variational) lower bound:

$$\mathcal{J} \geq \mathcal{L}(\mu, \Sigma, \phi)$$

(b) E-step

[2 points] Write down the E-step. Specifically, define the distribution $Q_{ij} = q_i(y^{(i)} = j)$

For the E-step, we update the distribution of the latent variable to be the same as posterior, in order to make the KL-divergence 0 and thus maximize the variational lower bound $\mathcal{L}$ under current parameters:

$$q^{new}(y) := p(y|x, \theta)$$

Here to be concrete, we set for each $l+1 \leq i \leq l+u$ and $j = 0, 1$:

$$Q_{ij}^{new} := p(y^{(i)} = j \mid \mathbf{x}^{(i)}; \mu_j, \Sigma_j, \phi)$$

So we compute the posterior $p(y^{(i)} = j \mid \mathbf{x}^{(i)}; \mu_j, \Sigma_j, \phi)$ for each $l+1 \leq i \leq l+u$ and $j = 0, 1$:

Using Bayes' theorem:

$$p(y^{(i)} = j \mid \mathbf{x}^{(i)}; \mu_j, \Sigma_j, \phi) = \frac{p\left(y^{(i)} = j\right) \cdot p\left(\mathbf{x}^{(i)} \mid y^{(i)} = j\right)}{\sum_{k \in \{0,1\}} p\left(y^{(i)} = k\right) \cdot p\left(\mathbf{x}^{(i)} \mid y^{(i)} = k\right)}$$

Plug in the model assumptions:

- $p(y = k) = \phi_k$
- $p(\mathbf{x} \mid y = k) = \mathcal{N}\left(\mathbf{x}; \mu_k, \Sigma_k\right)$

Thus, we get:

$$Q_{ij} = \frac{\phi_j \cdot \mathcal{N}\left(\mathbf{x}^{(i)}; \mu_j, \Sigma_j\right)}{\sum_{k \in \{0,1\}} \phi_k \cdot \mathcal{N}\left(\mathbf{x}^{(i)}; \mu_k, \Sigma_k\right)}$$

(where $\phi_0 = 1 - \phi,\ \phi_1 = \phi$)

(c) M-step for μ_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.

Sol:

For M step we want:

$$\theta^{\text{new}} := \text{argmax}_\theta \, \mathcal{L}(q, \theta)$$

For the parameter $\mu_k, k = 0, 1$, we check both the labelled and unlabelled part of variational lower bound $\mathcal{L}$.
The labelled part: $\sum_{i=1}^{l} \log p\left(x^{(i)}, y^{(i)}\right)$, where for each i,

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

Thus μ_k contributes to $\mathcal{L}$ only when $y^{(i)} = k$. And thus the contribution is:

$$\begin{aligned}
&\sum_{i=1}^{l} \mathbb{I}\left[y^{(i)}=k\right] \log \phi_j \mathcal{N}\left(\mathbf{x}^{(i)}; \mu_k, \Sigma_k\right) \\
&= \sum_{i=1}^{l} \mathbb{I}\left[y^{(i)}=k\right] \log \mathcal{N}\left(\mathbf{x}^{(i)}; \mu_k, \Sigma_k\right) + \sum_{i=1}^{l} \mathbb{I}\left[y^{(i)}=k\right] \log \phi_j
\end{aligned}$$

And since $\sum_{i=1}^{l} \mathbb{I}\left[y^{(i)}=k\right] \log$ does not depend on μ_k also, the contribution is thus

$$\sum_{i=1}^{l} \mathbb{I}\left[y^{(i)}=k\right] \log \mathcal{N}\left(\mathbf{x}^{(i)}; \mu_k, \Sigma_k\right)$$

The unlabeled part: $\lambda \sum_{i=l+1}^{l+u} \sum_{j\in\{0,1\}} Q_{ij} \log \frac{p(x^{(i)}, y^{(i)}=j)}{Q_{ij}}$. Thus for this part, the contribution of μ_k to $\mathcal{L}$ is:

$$\begin{aligned}
\lambda \sum_{i=l+1}^{l+u} Q_{ik} \log \frac{p(x^{(i)}, y^{(i)}=k)}{Q_{ik}} &= \lambda \sum_{i=l+1}^{l+u} Q_{ik} \log \frac{\phi_k \mathcal{N}\left(\mathbf{x}^{(i)}; \mu_k, \Sigma_k\right)}{Q_{ik}} \\
&= \lambda \sum_{i=l+1}^{l+u} Q_{ik} \left(\log \mathcal{N}\left(\mathbf{x}^{(i)}; \mu_k, \Sigma_k\right) + \log \phi_k - \log Q_{ik} \right)
\end{aligned}$$

But since for the M-step we fix the distribution of the latent variable, Q_{ik} **is constant**, so the contribution is:

$$\lambda \sum_{i=l+1}^{l+u} Q_{ik} \log \mathcal{N}\left(\mathbf{x}^{(i)}; \mu_k, \Sigma_k\right)$$

So the total contribution of μ_k to $\mathcal{L}$ is:

$$\mathcal{L}_{\mu_k} := \sum_{i=1}^{l} \mathbb{I}\left[y^{(i)}=k\right] \log \mathcal{N}\left(\mathbf{x}^{(i)}; \mu_k, \Sigma_k\right) + \lambda \sum_{i=l+1}^{l+u} Q_{ik} \log \mathcal{N}\left(\mathbf{x}^{(i)}; \mu_k, \Sigma_k\right)$$

Now we optimize this w.r.t. μ_k. Taking the gradient of the above w.r.t. μ_k, knowing:

$$\log \mathcal{N}\left(x; \mu_k, \Sigma_k\right) = -\frac{1}{2}\left(\mathbf{x}-\mu_k\right)^\top \Sigma_k^{-1}\left(\mathbf{x}-\mu_k\right) + \text{const}$$

Taking gradient:

$$\nabla_{\mu_k} \log \mathcal{N}\left(\mathbf{x}; \mu_k, \Sigma_k\right) = \Sigma_k^{-1}\left(\mathbf{x}-\mu_k\right)$$

So setting derivative of total contribution to zero:

$$\nabla_{\mu_k} \mathcal{L}_{\mu_k} := 0 \Longrightarrow \sum_{i=1}^{l} \mathbb{I}\left[y^{(i)}=k\right] \Sigma_k^{-1}\left(\mathbf{x}^{(i)}-\mu_k\right) + \lambda \sum_{i=l+1}^{l+u} Q_{ik} \Sigma_k^{-1}\left(\mathbf{x}^{(i)}-\mu_k\right) = 0$$

Multiplying Σ_k on the left we get:

$$\begin{aligned}
&\sum_{i=1}^{l} \mathbb{I}\left[y^{(i)}=k\right]\left(\mathbf{x}^{(i)}-\mu_k\right) + \lambda \sum_{i=l+1}^{l+u} Q_{ik}\left(\mathbf{x}^{(i)}-\mu_k\right) = 0 \\
&\sum_{i=1}^{l} \mathbb{I}\left[y^{(i)}=k\right]\mathbf{x}^{(i)} + \lambda \sum_{i=l+1}^{l+u} Q_{ik}\mathbf{x}^{(i)} = \left(\sum_{i=1}^{l} \mathbb{I}\left[y^{(i)}=k\right] + \lambda \sum_{i=l+1}^{l+u} Q_{ik}\right)\mu_k
\end{aligned}$$

Then we get the optimal μ_k to update:

$$\mu_k = \frac{\sum_{i=1}^{l} \mathbb{I}\left[y^{(i)} = k\right] \mathbf{x}^{(i)} + \lambda \sum_{i=l+1}^{l+u} Q_{ik} \mathbf{x}^{(i)}}{\sum_{i=1}^{l} \mathbb{I}\left[y^{(i)} = k\right] + \lambda \sum_{i=l+1}^{l+u} Q_{ik}}$$

Intuition of what μ_k looks like in termes of $x^{(i)}$'s and pseudo-counts:

μ_k is estimated by the weighted average of $\mathbf{x}^{(i)}$ that are in the class k, where for the labelled part, real counts is applied ($\sum_{i=1}^{l} \mathbb{I}\left[y^{(i)} = k\right]$, "how many data points is in class k) and for the unlabelled part, pseudo-counts is applied ($\sum_{i=l+1}^{l+u} Q_{ik}$, "how many data points are expected to be in class k by prob modeling).

And we use hyperparameter λ to control whether labeled and unlabeld data is more important in this learning.

If we only look at the labelled part, the μ_k is same as that we are doing GDA. If we only look at the unlabelled part, the μ_k is same as that we are doing GMM.

(d) M-step for ϕ

[6 points] [6 points] 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.

Sol:

$$\begin{aligned}
\mathcal{L} &= \sum_{i=1}^{l} \log p\left(x^{(i)}, y^{(i)}\right) + \lambda \sum_{i=l+1}^{l+u} \sum_{j \in \{0,1\}} Q_{ij} \log \frac{p\left(x^{(i)}, y^{(i)} = j\right)}{Q_{ij}} \\
&= \sum_{i=1}^{l} \left(\log \frac{\phi_{y(i)}}{(2\pi)^{\frac{M}{2}} |\Sigma_j|^{\frac{1}{2}}} - \frac{1}{2} \left(\mathbf{x}^{(i)} - \mu_j\right)^{\top} \Sigma_j^{-1} \left(\mathbf{x}^{(i)} - \mu_j\right) \right) + \lambda \sum_{i=l+1}^{l+u} [Q_{i1} \log \phi + Q_{i0} \log(1-\phi) \\
&= \sum_{i=1}^{l} \left(\log \phi_{y(i)} - \log(2\pi)^{\frac{M}{2}} |\Sigma_j|^{\frac{1}{2}} - \frac{1}{2} \left(\mathbf{x}^{(i)} - \mu_j\right)^{\top} \Sigma_j^{-1} \left(\mathbf{x}^{(i)} - \mu_j\right) \right) + \lambda \sum_{i=l+1}^{l+u} [Q_{i1} \log \phi + Q_{i0} l
\end{aligned}$$

Removing the terms that ϕ does not depend on, i.e. $-\log(2\pi)^{\frac{M}{2}} |\Sigma_j|^{\frac{1}{2}} - \frac{1}{2} \left(\mathbf{x}^{(i)} - \mu_j\right)^{\top} \Sigma_j^{-1} \left(\mathbf{x}^{(i)} - \mu_j\right)$ for each i, then we get the total contribution of ϕ to $\mathcal{L}$ is:

$$\begin{aligned}
\mathcal{L}_\phi &:= \sum_{i=1}^{l} \log \phi_{y(i)} + \lambda \sum_{i=l+1}^{l+u} [Q_{i1} \log \phi + Q_{i0} \log(1-\phi)] \\
&= \sum_{i=1}^{l} \left(\mathbb{I}\left[y^{(i)} = 1\right] \log \phi + \mathbb{I}\left[y^{(i)} = 0\right] \log(1-\phi) \right) + \lambda \sum_{i=l+1}^{l+u} \left(Q_{i1} \log \phi + Q_{i0} \log(1-\phi)\right) \\
&= \left(\sum_{i=1}^{l} \mathbb{I}\left[y^{(i)} = 1\right] + \lambda \sum_{i=l+1}^{l+u} Q_{i1} \right) \log \phi + \left(\sum_{i=1}^{l} \mathbb{I}\left[y^{(i)} = 0\right] + \lambda \sum_{i=l+1}^{l+u} Q_{i0} \right) \log(1-\phi)
\end{aligned}$$

We set:

$$A := \sum_{i=1}^{l} \mathbb{I}\left[y^{(i)} = 1\right] + \lambda \sum_{i=l+1}^{l+u} Q_{i1}, \quad B := \sum_{i=1}^{l} \mathbb{I}\left[y^{(i)} = 0\right] + \lambda \sum_{i=l+1}^{l+u} Q_{i0}$$

To maximize $\mathcal{L}_\phi$ over ϕ, we set $\nabla_\phi \mathcal{L}_\phi := 0$, get:

$$\nabla_\phi \mathcal{L}_\phi = \frac{A}{\phi} - \frac{B}{1-\phi} = 0$$
$$A(1-\phi) = B\phi$$
$$A = A\phi + B$$
$$\phi = \frac{A}{A+B}$$

Then we get the optimal μ_k to update:

$$\phi = \frac{\sum_{i=1}^{l} \mathbb{I}\left[y^{(i)} = 1\right] + \lambda \sum_{i=l+1}^{l+u} Q_{i1}}{l + \lambda u}$$

Intuition of what ϕ looks like in termes of $x^{(i)}$'s and pseudo-counts:

ϕ, the estimated prior probability of class 1, is estimated by the fraction of examples out of all labeled and unlabeled points that are believed to be in class 1. priority of labeled and unlabeled in the model is controlled by λ .

Labeled examples contribute hard counts (via $\mathbb{I}\left[y^{(i)} = 1\right]$), while unlabeled examples contribute soft counts via Q_{i1}. The denominator is the total number of effective examples, including both labeled and scaled unlabeled.

(e) M-step for Σ_k

[3 points] Finally, let's think about the M-step update rule for $\mathbf{\Sigma}_k$ 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 $\mathbf{\Sigma}_k$, without derivation, based on your guess and the analogy we have seen. Also, explain in words (English) what intuitively $\mathbf{\Sigma}_k$ looks like in terms of $\mathbf{x}^{(i)}$'s (each of labeled and unlabeled) and pseudo-counts.

Sol:

By similar reasoning we can get:

$$\Sigma_k = \frac{\sum_{i=1}^{l} \mathbb{I}\left[y^{(i)} = k\right] \left(\mathbf{x}^{(i)} - \mu_k\right)\left(\mathbf{x}^{(i)} - \mu_k\right)^\top + \lambda \sum_{i=l+1}^{l+u} Q_{ik} \left(\mathbf{x}^{(i)} - \mu_k\right)\left(\mathbf{x}^{(i)} - \mu_k\right)^\top}{\sum_{i=1}^{l} \mathbb{I}\left[y^{(i)} = k\right] + \lambda \sum_{i=l+1}^{l+u} Q_{ik}}$$

Intuition of what Σ_k looks like in termes of $x^{(i)}$'s and pseudo-counts:

Σ_k is esimated by the (not strictly) data covariance matrix of sample data points in class k.

For labeled examples, it literarily sample from data points that are belong to class k; for unlabeled examples, it takes $\sum_{i=l+1}^{l+u} Q_{ik}$ through probability modeling as pseudo-counts. The denominator is the total effective counts (hard and pseudo, importance weighted by λ between labelled and unlablled) of class k, ensuring it's a proper average.

Then the whole matrix can be viewed as data covariance matrix (but not strictly, since it has pseudo counts).

3. [20 points] PCA and eigenfaces

(a) derive PCA from "minimizing squared error" viewpoint

(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 $K \in \{1, \ldots, D\}$ be arbitrary and let $\mathbf{x}^{(n)} \in \mathbb{R}^D$. Let $\mathcal{U} = \Big\{ \mathbf{U} = [\mathbf{u}_1 \cdots \mathbf{u}_K] \in \mathbb{R}^{D \times K} \mid \{\mathbf{u}_i\}_{i=1}^K \text{ 's are orthonormal vectors } \Big\}$, 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, show that:

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

where $\mathbf{S}$ is the data covariance matrix $\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \left(\mathbf{x}^{(n)} - \overline{\mathbf{x}}\right) \left(\mathbf{x}^{(n)} - \overline{\mathbf{x}}\right)^\top$, $\overline{\mathbf{x}}$ 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, show that 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., $\mathbf{u}_1, \ldots, \mathbf{u}_K$ 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 $\mathcal{J} = \frac{1}{N} \left\| \mathbf{X} - \mathbf{U}\mathbf{U}^\top \mathbf{X} \right\|_F^2$, where $\mathbf{X} \in \mathbb{R}^{D \times N}$ is the matrix that stacks all the data points $\left\{\mathbf{x}^{(n)}\right\}_{n=1}^N$ as column vectors, and the $\| \cdot \|_F$ denotes the Frobenious norm:

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

and use the fact $\|A\|_F^2 = \operatorname{tr}\left(A^\top A\right)$ and $\operatorname{tr}(\mathbf{S}) = \sum_i \lambda_i$. 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.

> **Proof**:

First, **WLOG, we can assume that the data is zero-centered**, i.e.,

$$\overline{x} = \frac{1}{N}\sum_{n=1}^{N} x^{(n)} = \mathbf{0}$$

This is because the directions of maximum variance depend on how the data is spread, not on where it's located; if the data is not zero-centered, shifting all data points by a constant vector (i.e. the mean) doesn't change the shape of the data. More concretely, even if the data is not zero-centered, the normalized data which is used for the data covariance matrix, will still be zro-centered:

$$S = \frac{1}{N}\sum_{n=1}^{N}\left(x^{(n)} - \overline{x}\right)\left(x^{(n)} - \overline{x}\right)^T$$

Since we are projecting each data point $x^{(n)} \in \mathbb{R}^D$ onto a K-dimensional subspace spanned by orthonormal basis vectors $\mathbf{u}_1, \ldots, \mathbf{u}_K$. The projection of $x^{(n)}$ is:

$$\widetilde{x}^{(n)} = UU^T x^{(n)}$$

Then the projection error for each data point is:

$$\left\| x^{(n)} - UU^T x^{(n)} \right\|^2$$

So the total average error is:

$$\mathcal{J} = \frac{1}{N}\sum_{n=1}^{N}\left\| x^{(n)} - UU^T x^{(n)} \right\|^2$$

Define the data matrix $X = \left[x^{(1)}, \ldots, x^{(N)}\right] \in \mathbb{R}^{D\times N}$. Then by def of **Frobenius norm** we have:

$$\mathcal{J} = \frac{1}{N}\left\| X - UU^T X \right\|_F^2$$

Using the identity:

$$\|A\|_F^2 = \operatorname{tr}\left(A^T A\right)$$

We have:

$$\mathcal{J} = \frac{1}{N}\operatorname{tr}\left[\left(X - UU^T X\right)^\top \left(X - UU^T X\right)\right]$$

Let's denote $P := UU^T$ as the projection matrix. Then:

$$\begin{aligned}\mathcal{J} &= \frac{1}{N}\operatorname{tr}\left[((I-P)X)^T (I-P)X\right] \\ &= \frac{1}{N}\operatorname{tr}\left[X^T (I-P)^T (I-P)X\right]\end{aligned}$$

Note that (as the property of projection matrix) we have:

$$P^T = \left(UU^T\right)^T = UU^T = P \quad \text{and} \quad P^2 = \left(UU^T UU^T\right) = UU^T = P$$

Thus we also have:

$$(I-P)^T = I^T - P^T = I - P$$

so

$$(I-P)^T (I-P) = I - 2P + P^2 = I - P$$

This simplifies $\mathcal{J}$ to

$$\begin{aligned}\mathcal{J} = \frac{1}{N}\text{tr}\left[X^T(I-P)X\right] &= \frac{1}{N}\text{tr}\left[X^TX - X^TPX\right] \\ &= \frac{1}{N}\left(\text{tr}\left(X^TX\right) - \text{tr}\left(X^TPX\right)\right) \quad \text{by linearity of trace} \\ &= \frac{1}{N}\left(\text{tr}\left(X^TX\right) - \text{tr}\left(X^TUU^TX\right)\right) \\ &= \frac{1}{N}\left(\text{tr}\left(X^TX\right) - \text{tr}\left(U^TXX^TU\right)\right) \quad \text{since } \text{tr}(AB) = \text{tr}(BA\end{aligned}$$

Define the data covariance matrix:

$$S := \frac{1}{N}\sum_{n=1}^{N}\left(x^{(n)} - \bar{x}\right)\left(x^{(n)} - \bar{x}\right)^T = \frac{1}{N}XX^T \quad \text{(since data is zero-centered)}$$

Then since $\text{tr}\left(X^TX\right) = \text{tr}\left(XX^T\right)$, we have:

$$\mathcal{J} = \text{tr}(S) - \text{tr}\left(U^TSU\right)$$

Now, trace of a symmetric matrix equals the sum of its eigenvalues, so denoting $\lambda_1 \geq \cdots \geq \lambda_D$ as the eigenvalues of S, we have $\text{tr}(S) = \sum_{i=1}^{D}\lambda_i$; and we know that $\text{tr}\left(U^TSU\right) = \sum_{i=1}^{K}\mathbf{u}_i^TS\mathbf{u}_i$, thus we simplify $\mathcal{J}$ to be:

$$\mathcal{J} = \sum_{i=1}^{D}\lambda_i - \sum_{i=1}^{K}\mathbf{u}_i^TS\mathbf{u}_i$$

To minimize $\mathcal{J}$ is to maximize $\sum_{i=1}^{K}\mathbf{u}_i^TS\mathbf{u}_i$ over $\{\mathbf{u}_i\}$.
Now we use the fact (from spectral theorem) that, the optimal solution $\mathbf{u}_i$'s to maximize $\sum_{i=1}^{K}\mathbf{u}_i^TS\mathbf{u}_i$ is to pick the top-K eigenvectors of S. Note each $\mathbf{u}_i$ is unit, to make them orthonormal (thus $\mathbf{u}_i^T\mathbf{u}_i = 1$ for each i)

So we have:

$$S\mathbf{u}_i = \lambda_i\mathbf{u}_i \implies \mathbf{u}_i^TS\mathbf{u}_i = \lambda_i\mathbf{u}_i^T\mathbf{u}_i = \lambda_i \tag{1}$$

This completes the proof that:

$$\mathcal{J}_{\min} = \sum_{i=1}^{D}\lambda_i - \sum_{i=1}^{K}\lambda_i = \sum_{k=K+1}^{D}\lambda_k$$

(b) implement PCA

(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.

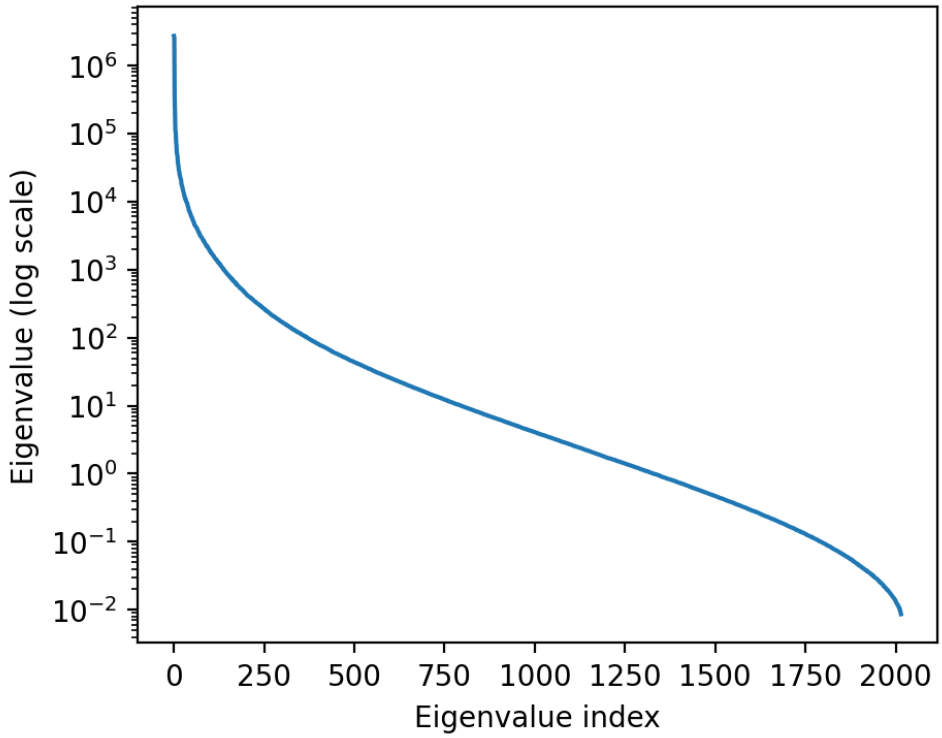
(c) perform PCA on face images

(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.

First 10 principal components:

```
[2719333.89451627 2611865.95964519  365515.29797577  211149.83657024
  110603.22720593  104715.79567702   78512.9353901    67032.06261281
   52592.80467658   49197.08114141]
```

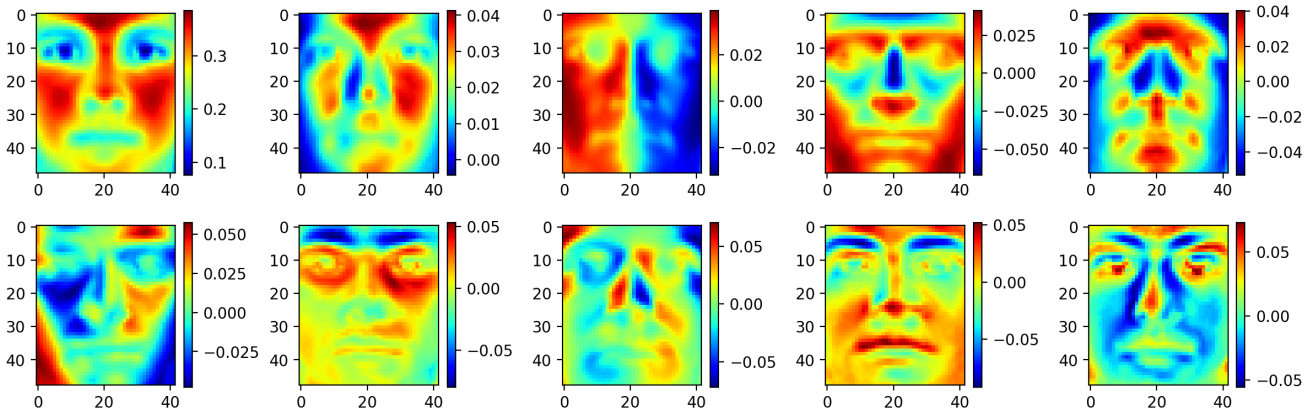
Plot of eigenvalues (in descending order):



(d) plot eigenfaces

(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).

Eigenfaces in descending eigenvalues:



My comment:

The first component in first row captures that a face has two eyes (and eyes are similar among all faces);

The third component in first row captures the symmetry of the face;

> The last component in second row captures smiling;
>
> The last but one component in second row captures the behavior of slightly raising head up.

(e) calculate: how many principle components are needed to represent 95% total variance

(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?

```
# ~~START DELETE NO TODO~~
cumul_ev = np.cumsum(eigenvalues)
variance_por = cumul_ev / cumul_ev[-1]

index_1 = (variance_por >= 0.95).nonzero()[0][0] + 1
index_2 = (variance_por >= 0.99).nonzero()[0][0] + 1
print(index_1, 1 - index_1 / 48/42)
print(index_2, 1 - index_2 / 48/42)
# ~~END DELETE~~
```

[30] ✓ 0.0s

```
43 0.9786706349206349
167 0.9171626984126984
```

> Upon computation, **43 principal components** are needed to represent **95% of the total variance**, with a **reduction of 97.87% in dimension**.
>
> **167 principal components** are needed to represent **99% of the total variance**, with a **reduction of 91.72% in dimension**.

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) implement ICA

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

> details:
>
> - $y = Wx$ are the independent components.
> - $\log g'(y) = \log(\sigma(y)(1 - \sigma(y)))$
> - The gradient of the sum over $\log g'(y_j)$ gives $(1 - 2\sigma(y_j))x^T$
> - Plus the gradient of $\log |\det W| = (W^{-1})^T$

(b) report W

(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`).

```
W solution:
 [[ 72.15081922  28.62441682  25.91040458 -17.2322227  -21.191357  ]
 [ 13.45886116  31.94398247  -4.03003982 -24.0095722   11.89906179]
 [ 18.89688784  -7.80435173  28.71469558  18.14356811 -21.17474522]
 [ -6.0119837   -4.15743607  -1.01692289  13.87321073  -5.26252289]
 [ -8.74061186  22.55821897   9.61289023  14.73637074  45.28841827]]
CPU times: user 1min 28s, sys: 10 s, total: 1min 38s
Wall time: 20.8 s
```

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) derive variational lower bound of a conditional VAE

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

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

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 $q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y})$. The posterior distribution is modeled by a neural network of parameters ϕ. The generative distribution $p_\theta(\mathbf{x} \mid \mathbf{y})$ 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., $q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y}) = \prod_{j=1}^m q_\phi\left(z_j \mid \mathbf{x}, \mathbf{y}\right)$, and $p_\theta(\mathbf{z} \mid \mathbf{y}) = \prod_{j=1}^m p_\theta\left(z_j \mid \mathbf{y}\right)$.

> **Proof:**
>
> We want to optimize: $\log p_\theta(\mathbf{x} \mid \mathbf{y})$.
>
> By introducing latent variable $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y})$, we get
>
> $$\log p_\theta(\mathbf{x} \mid \mathbf{y}) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}\left[\log p_\theta(\mathbf{x} \mid \mathbf{y})\right]$$
>
> since $p_\theta(\mathbf{x} \mid \mathbf{y})$ does not depend on $\mathbf{z}$.
>
> And we do deduction to get:
>
> $$\begin{aligned}\log p_\theta(\mathbf{x} \mid \mathbf{y}) &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}\left[\log \frac{p_\theta(\mathbf{x}, \mathbf{z} \mid \mathbf{y})}{p_\theta(\mathbf{z} \mid \mathbf{x}, \mathbf{y})}\right] \quad \text{by Bayes' rule} \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}\left[\log \left(\frac{p_\theta(\mathbf{x}, \mathbf{z} \mid \mathbf{y})}{p_\theta(\mathbf{z} \mid \mathbf{x}, \mathbf{y})} \cdot \frac{q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y})}{q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y})}\right)\right] \quad \text{multiplying const 1} \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}\left[\log \left(\frac{p_\theta(\mathbf{x}, \mathbf{z} \mid \mathbf{y})}{q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y})} \cdot \frac{q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y})}{p_\theta(\mathbf{z} \mid \mathbf{x}, \mathbf{y})}\right)\right] \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}\left[\log \left(\frac{p_\theta(\mathbf{x} \mid \mathbf{z}, \mathbf{y}) p_\theta(\mathbf{z} \mid \mathbf{y})}{q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y})} \cdot \frac{q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y})}{p_\theta(\mathbf{z} \mid \mathbf{x}, \mathbf{y})}\right)\right] \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z},\mathbf{x}|\mathbf{y})}\left[\log p_\theta(\mathbf{x}, \mathbf{z} \mid \mathbf{y})\right] + \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}\left[\log \frac{p_\theta(\mathbf{z} \mid \mathbf{y})}{q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y})}\right] + \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}\left[\log \frac{q_\theta(\mathbf{z} \mid \mathbf{x}, \mathbf{y})}{p_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y})}\right]\end{aligned}$$
>
> Here, the first term is the expected joint log-likelihood under the approximate posterior.
> The second term is:
>
> $$\mathbb{E}_{q_\phi}\left[\log \frac{p_\theta(\mathbf{z} \mid \mathbf{y})}{q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y})}\right] = -D_{\mathrm{KL}}\left(q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y}) \| p_\theta(\mathbf{z} \mid \mathbf{y})\right)$$
>
> The third term is:
>
> $$\mathbb{E}_{q_\phi}\left[\log \frac{q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y})}{p_\theta(\mathbf{z} \mid \mathbf{x}, \mathbf{y})}\right] = D_{\mathrm{KL}}\left(q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y}) \| p_\theta(\mathbf{z} \mid \mathbf{x}, \mathbf{y})\right) \geq 0$$
>
> So putting it all together we have:

$$\log p_\theta(\mathbf{x} \mid \mathbf{y}) = \underbrace{\mathbb{E}_{q_\phi}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z}, \mathbf{y})\right] - D_{\mathrm{KL}}\left(q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y}) \| p_\theta(\mathbf{z} \mid \mathbf{y})\right)}_{:=\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{y})} + D_{\mathrm{KL}}\left(q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y}) \| p_\theta(\mathbf{z} \mid \mathbf{x}, \mathbf{y})\right)$$

And since the KL divergence $D_{\mathrm{KL}}(q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y}) \| p_\theta(\mathbf{z} \mid \mathbf{x}, \mathbf{y}))$ is non-negative, we finally get:

$$\log p_\theta(\mathbf{x} \mid \mathbf{y}) \geq \mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{y})$$

(b) Derive the analytical KL-divergence between two Gaussian distributions

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

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

where μ_j and σ_j are the outputs of the neural network that estimates the parameters of the posterior distribution $q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{y})$.
You can assume without proof that

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

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

Proof:

Since we know

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

It suffices to compute each $\sum_{j=1}^{m} D_{KL}\left(q_\phi\left(z_j \mid \mathbf{x}, \mathbf{y}\right) \| p_\theta\left(z_j \mid \mathbf{y}\right)\right)$.

We know that:

$$q(z_j) = \mathcal{N}\left(\mu_j, \sigma_j^2\right) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{\left(z_j - \mu_j\right)^2}{2\sigma_j^2}\right)$$

And

$$p(z_j) = \mathcal{N}(0, 1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z_j^2}{2}\right)$$

Then we can compute the log ratio:

$$\begin{aligned}\log \frac{q(z)}{p(z)} &= \log\left(\frac{1}{\sqrt{2\pi\sigma_j^2}}\exp\left(-\frac{(z-\mu_j)^2}{2\sigma_j^2}\right)\cdot\frac{\sqrt{2\pi}}{1}\exp\left(\frac{z^2}{2}\right)\right)\\ &= \log\left(\frac{1}{\sqrt{\sigma_j^2}}\exp\left(-\frac{(z-\mu_j)^2}{2\sigma_j^2}+\frac{z^2}{2}\right)\right)\\ &= -\frac{1}{2}\log\sigma_j^2+\frac{1}{2}\left(z^2-\frac{(z-\mu_j)^2}{\sigma_j^2}\right)\end{aligned}$$

Taking expectation to get the KL conv:

$$D_{\mathrm{KL}}(q\|p)=\mathbb{E}_{z\sim q}\left[\log\frac{q(z)}{p(z)}\right]=-\frac{1}{2}\log\sigma_j^2+\frac{1}{2}\mathbb{E}_{z\sim q}\left[z^2-\frac{(z-\mu_j)^2}{\sigma_j^2}\right]$$

And we compute easily that:

$$\mathbb{E}_{z\sim q}\left[z^2\right]=\mathrm{Var}(z)+\mathbb{E}[z]^2=\sigma_j^2+\mu_j^2,\quad \mathbb{E}_{z\sim q}\left[(z-\mu_j)^2\right]=\sigma_j^2$$

Thus putting it together we have:

$$D_{\mathrm{KL}}(q\|p)=-\frac{1}{2}\log\sigma_j^2+\frac{1}{2}\left(\sigma_j^2+\mu_j^2-1\right)=\frac{1}{2}\left(-\log\sigma_j^2+\sigma_j^2+\mu_j^2-1\right)$$

So back to the original sum, we have:

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

(c) implement CVAE

[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 $q_\phi(\mathbf{z}\mid\mathbf{x},\mathbf{y})$.
- Implement the generative_model function $p_\theta(\mathbf{x}\mid\mathbf{z},\mathbf{y})$.
- 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 10×10 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 $\mathbf{z}$.

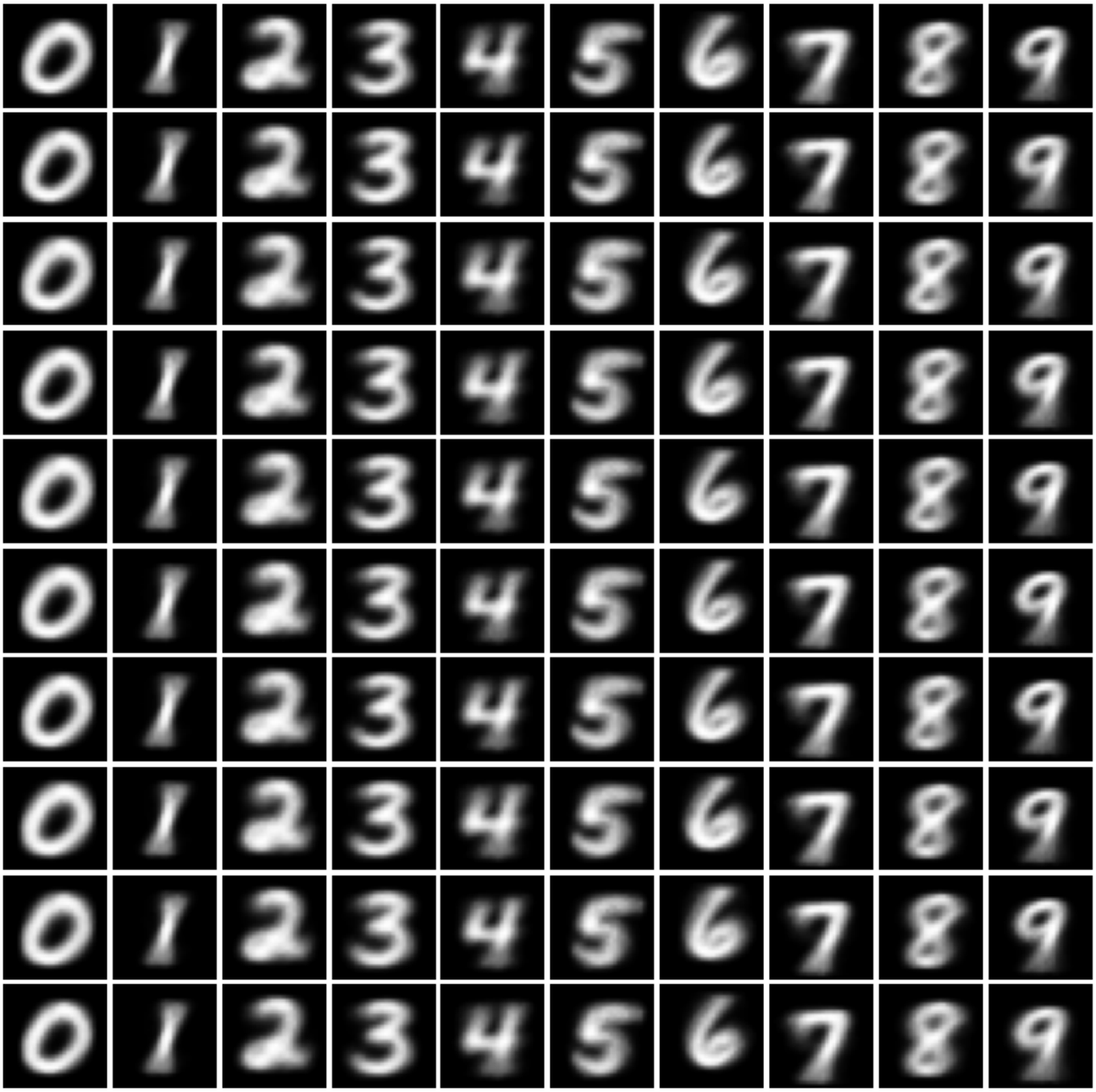
> My recognition model uses two hidden layers (size ignoring batch size: input+num_classes -> hidden_units, hidden_units -> hidden units) activated by relu function, and a linear output layer, outputing $\mu_{z|x}$ and $\Sigma_{z|x}$ (size ignoring batch size: hidden_units -> latent_size).
>
> My generation model also uses two hidden layers (size ignoring batch size: latent_size+num_classes -> hidden_units, hidden_units -> hidden_units) activated by relu function, and a sigmoid output layer, generating $\hat{x}$.

```
CVAE(
  (fc1): Linear(in_features=794, out_features=400, bias=True)
  (fc2): Linear(in_features=400, out_features=400, bias=True)
  (fc_mu): Linear(in_features=400, out_features=20, bias=True)
  (fc_logvar): Linear(in_features=400, out_features=20, bias=True)
  (fc3): Linear(in_features=30, out_features=400, bias=True)
  (fc4): Linear(in_features=400, out_features=400, bias=True)
  (fcout): Linear(in_features=400, out_features=784, bias=True)
)
```

> We minimized our loss_function $-\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{y})$, from 1.599241 to 0.240457.

```
Train Epoch: 10 [6400/10000 (64%)]      Loss: 0.244923
Train Epoch: 10 [9600/10000 (96%)]      Loss: 0.240457
CPU times: user 1min 12s, sys: 11.1 s, total: 1min 23s
Wall time: 24.5 s
```

> My generated images that look like MNIST digits reflecting the given label and the 10 different noise vectors:



The generating model performs very well on MNIST.