Sharif University of Technology
Electrical Engineering Department

Introduction to Machine Learning
Project Phase 2

Amir Hossein Moraveji - Amir Hossein Yari
99104232 - 99102507

June 30, 2023

Contents

Simulation Question 1)

At first we load dataset and convert it to numpy array. after that choose 100 samples of each digit(generally 1000 samples).

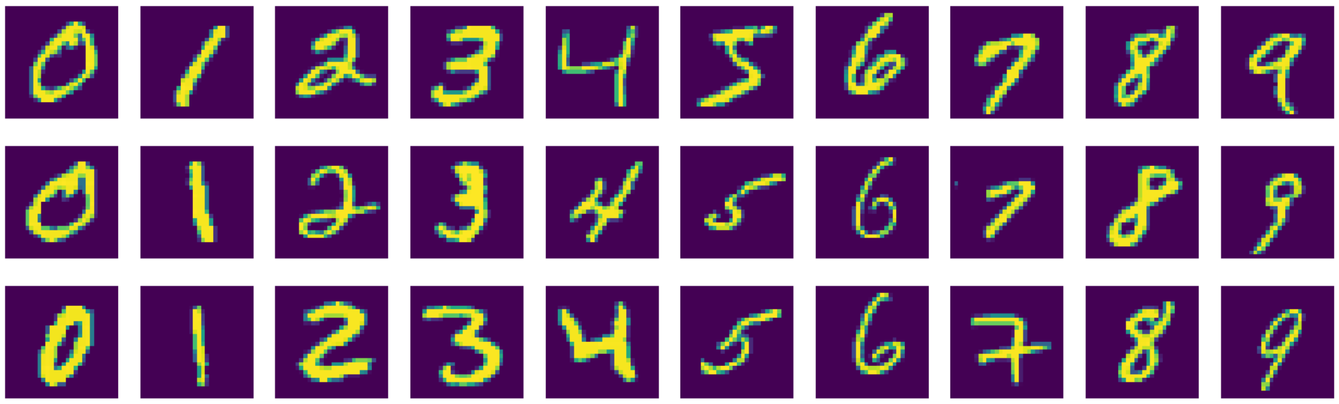
```
# Loading the dataset
train_set = MNIST(root='.', train=True, download=True)

data_all = train_set.data.numpy()
targets_all = train_set.targets.numpy()

print('all data : ',type(data_all) , data_all.shape)
print('all targets : ',type(targets_all) , targets_all.shape)

# choose 1000 samples with 100 samples for each digit
data = np.zeros((1000,28,28))
targets = np.zeros((1000,))
for i in  range(10):
idx = (targets_all==i)
data[0+100*i:100*(i+1),:,:] = data_all[idx][:100,:,:]
targets[0+100*i:100*(i+1)] = targets_all[idx][:100]

print('chosen data : ',type(data) , data.shape)
print('chosen targets : ',type(targets) , targets.shape)
```

Output:

```
all data :  <class 'numpy.ndarray'> (60000, 28, 28)
all targets :  <class 'numpy.ndarray'> (60000,)
chosen data :  <class 'numpy.ndarray'> (1000, 28, 28)
chosen targets :  <class 'numpy.ndarray'> (1000,)
```

By plotting a number of images, we can see a part of our dataset.

```
# take a look at some examples from the dataset
plt.figure(figsize=(20,6))
for i in  range(10):
plt.subplot(3,10,i+1)
plt.imshow(data[0+i*100])
plt.axis('off')
plt.subplot(3,10,i+11)
plt.imshow(data[1+i*100])
plt.axis('off')
plt.subplot(3,10,i+21)
plt.imshow(data[2+i*100])
plt.axis('off')
```



These functions perform extraction operations.(Extract patch of data happens next)

```
# extracting patches and vectorizing them
def extract_1(img , m):
  L = len(img)
  n = L-m+1
  n2 = n**2
  m2 = m**2
  p = np.zeros((n2,m2))
  for i in range(n):
    for j in range(n):
      p[i*n+j,:] = img[i:i+m,j:j+m].reshape((1,m2))/255
  return p
def extract(data , m):
  n = (28-m+1)**2
  N = len(data)
  patches = np.zeros((N*n,m**2))
  for i in range(N):
    patches[i*n:(i+1)*n,:] = extract_1(data[i,:,:] , m)
  return patches
```

Simulation Question 2)

This is my own EM algorithm.

```
# ----- No Need to run this box -----
# EM Algorithm ( written by us )
def mvn_prob(vector, mean, covariance):
  D = len(vector)
  vector = vector.reshape((D,1))
  mean = mean.reshape((D,1))
  diff = vector - mean
  prob = np.exp(-0.5 * diff.T @ np.linalg.inv(covariance) @ diff) / ((2 * np.pi)**(D/2) *
                                                np.linalg.det(covariance)**(1/2))
  return prob

def E_step(pi , mu , sigma , data ):
  N = len(data)
  K = len(mu)
  pi = pi.reshape((K,1))
  q = np.zeros((N,K))
  for n in range(N):
    denom = 0
    for k in range(K):
      p = mvn_prob(data[n,:], mu[k,:], sigma[k,:,:])
      denom += pi[k]*p
    for k in range(K):
      q[n,k] = pi[k]*mvn_prob(data[n,:], mu[k,:], sigma[k,:,:])/denom
  return q

def M_step(q , data ):
  N = len(data)
  D = len(data[0,:])
  K = len(q[0,:])
  pi = q.mean(axis=0)
  mu = np.zeros((K,D))
  sigma = np.zeros((K,D,D))
  for k in range(K):
    den = pi[k] if pi[k]>0 else 0.0001
    mu[k,:] = np.mean(np.diag(q[:,k]) @ data , axis=0) / den
    for n in range(N):
      d = data[n,:] - mu[k,:]
      sigma[k,:,:] += q[n,k]*(d.reshape((D,1)) @ d.reshape((1,D)))
    sigma[k,:,:] += np.diag(0.0001*np.ones(D))
    sigma[k,:,:] /= (den*N)
  return pi , mu , sigma

def EM_loop(data , step , pi , mu , sigma):
  q = E_step(pi , mu , sigma , data)
  pi_new , mu_new , sigma_new = M_step(q , data)
  step -= 1
  if step==0 :
    return pi_new , mu_new , sigma_new
  else :
    return EM_loop(data , step , pi_new , mu_new , sigma_new)

def initialize(data , K):
  D = len(data[0,:])
  pi = np.ones((1,K)) / K
  M = data.max()
  m = data.min()
  mu = (np.random.rand(K,D)-0.5)*(M-m)/4 + (M+m)/2
  sigma = np.zeros((K,D,D))
  for k in range(K):
    r = np.random.rand(D)
    sigma[k,:,:] = np.diag(r)
  return pi , mu , sigma

def EM_algorithm(data , K , step):
  pi , mu , sigma = initialize(data , K )
  pi , mu , sigma = EM_loop(data , step , pi , mu , sigma)
  return pi , mu , sigma
```

Because of Its runtime is long, we use GaussianMixture of python as below.

```
  # python GMM
  def train(input_data , K):
    gm = GaussianMixture(n_components=K, init_params="random_from_data").fit(input_data)
    return gm
```

Theory Question 1)

Calculate $\boldsymbol{\mu}_{\mathbf{Z},\mathbf{Y}}$:

$p_{\mathbf{Y},\mathbf{Z}} = p_{\mathbf{Y}|\mathbf{Z}} \times p_{\mathbf{Z}}$

$$\mathbf{X} = \begin{bmatrix} \mathbf{Z} \\ \mathbf{Y} \end{bmatrix}$$

$$E[\mathbf{X}] = \begin{bmatrix} E[\mathbf{Z}] \\ E[\mathbf{Y}] \end{bmatrix} = \begin{bmatrix} E[\mathbf{Z}] \\ E[E[\mathbf{Y}|\mathbf{Z}]] \end{bmatrix} = \begin{bmatrix} E[\mathbf{Z}] \\ E[\mathbf{W}\mathbf{Z}+\mathbf{b}] \end{bmatrix} = \begin{bmatrix} E[\mathbf{Z}] \\ \mathbf{W}\,E[\mathbf{Z}]+\mathbf{b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu}_{\mathbf{Z}} \\ \mathbf{W}\,\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b} \end{bmatrix} \checkmark$$

Calculate $\boldsymbol{\Sigma}_{\mathbf{Z},\mathbf{Y}}$:

$$\boldsymbol{\Sigma}_{\mathbf{X}} = E[(\mathbf{X}-E[\mathbf{X}])(\mathbf{X}-E[\mathbf{X}])^T] = E[\mathbf{X}\mathbf{X}^T] - E[\mathbf{X}]\,E[\mathbf{X}^T]$$

$$= E\begin{bmatrix} \mathbf{Z}\mathbf{Z}^T & \mathbf{Z}\mathbf{Y}^T \\ \mathbf{Y}\mathbf{Z}^T & \mathbf{Y}\mathbf{Y}^T \end{bmatrix} - \begin{bmatrix} \boldsymbol{\mu}_{\mathbf{Z}}\boldsymbol{\mu}_{\mathbf{Z}}^T & \boldsymbol{\mu}_{\mathbf{Z}}(\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})^T \\ (\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})\boldsymbol{\mu}_{\mathbf{Z}}^T & (\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})(\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}$$

$$\boldsymbol{\Sigma}_{11} = E[\mathbf{Z}\mathbf{Z}^T] - \boldsymbol{\mu}_{\mathbf{Z}}\boldsymbol{\mu}_{\mathbf{Z}}^T = \boldsymbol{\Sigma}_{\mathbf{Z}}$$

$$\boldsymbol{\Sigma}_{12} = E[\mathbf{Z}\mathbf{Y}^T] - \boldsymbol{\mu}_{\mathbf{Z}}(\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})^T = E[\mathbf{Z}(\mathbf{W}\mathbf{Z}+\mathbf{b})^T] - \boldsymbol{\mu}_{\mathbf{Z}}(\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})^T$$
$$= E[\mathbf{Z}\mathbf{Z}^T\mathbf{W}^T + \mathbf{Z}\mathbf{b}^T] - \boldsymbol{\mu}_{\mathbf{Z}}\boldsymbol{\mu}_{\mathbf{Z}}^T\mathbf{W}^T - \boldsymbol{\mu}_{\mathbf{Z}}\mathbf{b}^T = (E[\mathbf{Z}\mathbf{Z}^T] - \boldsymbol{\mu}_{\mathbf{Z}}\boldsymbol{\mu}_{\mathbf{Z}}^T)\mathbf{W}^T + (E[\mathbf{Z}] - \boldsymbol{\mu}_{\mathbf{Z}})\mathbf{b}^T$$
$$= \boldsymbol{\Sigma}_{\mathbf{Z}}\mathbf{W}^T$$

$$\boldsymbol{\Sigma}_{21} = (\boldsymbol{\Sigma}_{12})^T = \mathbf{W}\boldsymbol{\Sigma}_{\mathbf{Z}}$$

$$\boldsymbol{\Sigma}_{22} = E[\mathbf{Y}\mathbf{Y}^T] - (\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})(\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})^T = E[E[\mathbf{Y}\mathbf{Y}^T|\mathbf{Z}]] - (\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})(\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})^T$$
$$= E[E[\mathbf{Y}\mathbf{Y}^T|\mathbf{Z}] - E[\mathbf{Y}|\mathbf{Z}]\,E[\mathbf{Y}|\mathbf{Z}]^T + E[\mathbf{Y}|\mathbf{Z}]\,E[\mathbf{Y}|\mathbf{Z}]^T] - (\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})(\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})^T$$
$$= E[\boldsymbol{\Sigma}_{\mathbf{Y}|\mathbf{Z}} + (\mathbf{W}\mathbf{Z}+\mathbf{b})(\mathbf{W}Z+\mathbf{b})^T] - (\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})(\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})^T$$
$$= \boldsymbol{\Sigma}_{\mathbf{Y}|\mathbf{Z}} + \mathbf{W}\,E[\mathbf{Z}\mathbf{Z}^T]\,\mathbf{W}^T + \mathbf{b}\,E\mathbf{Z}^T\mathbf{W}^T + \mathbf{W}\,E[\mathbf{Z}]\,\mathbf{b}^T + \mathbf{b}\mathbf{b}^T - (\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})(\mathbf{W}\boldsymbol{\mu}_{\mathbf{Z}}+\mathbf{b})^T$$
$$= \boldsymbol{\Sigma}_{\mathbf{Y}|\mathbf{Z}} + \boldsymbol{W}(\boldsymbol{E}(\boldsymbol{Z}\boldsymbol{Z}^T) - \boldsymbol{\mu}_{\mathbf{Z}}\boldsymbol{\mu}_{\mathbf{Z}}^T)\boldsymbol{W}^T = \boldsymbol{\Sigma}_{\mathbf{Y}|\mathbf{Z}} + \boldsymbol{W}\boldsymbol{\Sigma}_{\mathbf{Z}}\boldsymbol{W}^T$$

$$\Rightarrow \boldsymbol{\Sigma}_{\mathbf{Z},\mathbf{Y}} = \begin{bmatrix} \boldsymbol{\Sigma}_{\mathbf{Z}} & \boldsymbol{\Sigma}_{\mathbf{Z}}\boldsymbol{W}^T \\ \boldsymbol{W}\boldsymbol{\Sigma}_{\mathbf{Z}} & \boldsymbol{\Sigma}_{\mathbf{Y}|\mathbf{Z}} + \boldsymbol{W}\boldsymbol{\Sigma}_{\mathbf{Z}}\boldsymbol{W}^T \end{bmatrix} \checkmark$$

Calculate $\boldsymbol{\mu}_{\mathbf{Z}|\mathbf{Y}}$ and $\boldsymbol{\Sigma}^{-1}_{\mathbf{Z}|\mathbf{Y}}$:

$$p_{\mathbf{Z}|\mathbf{Y}} = \frac{p_{\mathbf{Y},\mathbf{Z}}}{p_{\mathbf{Y}}}$$

$$\log(p_{\mathbf{Y},\mathbf{Z}}(\mathbf{y},\mathbf{z})) = -\frac{1}{2}((\mathbf{z}-\boldsymbol{\mu}_{\mathbf{Z}})^T\boldsymbol{\Sigma}_{\mathbf{Z}}^{-1}(\mathbf{z}-\boldsymbol{\mu}_{\mathbf{Z}}) + (\mathbf{y}-\boldsymbol{W}\mathbf{z}-\mathbf{b})^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}_{\mathbf{Y}|\mathbf{Z}}(\mathbf{y}-\boldsymbol{W}\mathbf{z}-\mathbf{b}))$$

$$= -\frac{1}{2}\begin{bmatrix}\mathbf{z}\\ \mathbf{y}\end{bmatrix}^T \begin{bmatrix}\boldsymbol{\Sigma}_{\mathbf{Z}}^{-1} + \boldsymbol{W}^T\boldsymbol{\Sigma}^{-1}_{\mathbf{Y}|\mathbf{Z}}\boldsymbol{W} & -\boldsymbol{W}^T\boldsymbol{\Sigma}^{-1}_{\mathbf{Y}|\mathbf{Z}}\\ -\boldsymbol{\Sigma}^{-1}_{\mathbf{Y}|\mathbf{Z}}\boldsymbol{W} & \boldsymbol{\Sigma}^{-1}_{\mathbf{Y}|\mathbf{Z}}\end{bmatrix}\begin{bmatrix}\mathbf{z}\\ \mathbf{y}\end{bmatrix}$$

$$\Rightarrow \boldsymbol{\Sigma}^{-1}_{\mathbf{Z}|\mathbf{Y}} = \boldsymbol{\Lambda} = \begin{bmatrix}\boldsymbol{\Sigma}_{\mathbf{Z}}^{-1} + \boldsymbol{W}^T\boldsymbol{\Sigma}^{-1}_{\mathbf{Y}|\mathbf{Z}}\boldsymbol{W} & -\boldsymbol{W}^T\boldsymbol{\Sigma}^{-1}_{\mathbf{Y}|\mathbf{Z}}\\ \\ -\boldsymbol{\Sigma}^{-1}_{\mathbf{Y}|\mathbf{Z}}\boldsymbol{W} & \boldsymbol{\Sigma}^{-1}_{\mathbf{Y}|\mathbf{Z}}\end{bmatrix}$$

From MVN we know that:

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\Sigma}_{1|2}(\boldsymbol{\Lambda}_{11}\boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{12}(\mathbf{y}_2 - \boldsymbol{\mu}_2))$$
$$\boldsymbol{\Sigma}_{1|2} = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} = \boldsymbol{\Lambda}_{11}^{-1}$$

$$\Rightarrow \boldsymbol{\Sigma}^{-1}_{\mathbf{Z}|\mathbf{Y}} = \boldsymbol{\Sigma}_{\mathbf{Z}}^{-1} + \boldsymbol{W}^T\boldsymbol{\Sigma}^{-1}_{\mathbf{Y}|\mathbf{Z}}\boldsymbol{W} \quad \checkmark$$

$$\boldsymbol{\mu}_{\mathbf{Z}|\mathbf{Y}} = \boldsymbol{\Sigma}_{\mathbf{Z}|\mathbf{Y}}[(\boldsymbol{\Sigma}_{\mathbf{Z}}^{-1} + \boldsymbol{W}^T\boldsymbol{\Sigma}^{-1}_{\mathbf{Y}|\mathbf{Z}}\boldsymbol{W})\boldsymbol{\mu}_{\mathbf{Z}} + \boldsymbol{W}^T\boldsymbol{\Sigma}^{-1}_{\mathbf{Y}|\mathbf{Z}}(\mathbf{y} - \boldsymbol{\mu}_{\mathbf{Y}})]$$

$$= \boldsymbol{\Sigma}_{\mathbf{Z}|\mathbf{Y}}[\boldsymbol{\Sigma}_{\mathbf{Z}}^{-1}\boldsymbol{\mu}_{\mathbf{Z}} + \boldsymbol{W}^T\boldsymbol{\Sigma}^{-1}_{\mathbf{Y}|\mathbf{Z}}(\boldsymbol{W}\boldsymbol{\mu}_{\mathbf{Z}} + \mathbf{y} - \boldsymbol{\mu}_{\mathbf{Y}})] = \boldsymbol{\Sigma}_{\mathbf{Z}|\mathbf{Y}}[\boldsymbol{\Sigma}_{\mathbf{Z}}^{-1}\boldsymbol{\mu}_{\mathbf{Z}} + \boldsymbol{W}^T\boldsymbol{\Sigma}^{-1}_{\mathbf{Y}|\mathbf{Z}}(\mathbf{y} - \mathbf{b})] \quad \checkmark$$

Theory Question 2)

The posterior distribution will not be a simple Gaussian distribution in general. Instead, it will be a GMM. let's consider the Bayesian framework. Given the prior distribution $p(\mathbf{Z})$ as a GMM and the likelihood $p(\mathbf{Y}|\mathbf{Z})$ as a normal distribution, the posterior distribution $p(\mathbf{Z}|\mathbf{Y})$ is obtained by applying Bayes theorem:

$$p(\mathbf{Z}|\mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{Z})\ p(\mathbf{Z})}{p(\mathbf{Y})}$$

Since $p(\mathbf{Y}|\mathbf{Z})$ is a normal distribution and $p(\mathbf{Z})$ is a GMM, the product $p(\mathbf{Y}|\mathbf{Z}) \times p(\mathbf{Z})$ is a mixture of Gaussians. When we divide the mixture of Gaussians by the marginal likelihood $p(\mathbf{Y})$, the resulting posterior distribution $p(\mathbf{Z}|\mathbf{Y})$ remains a GMM. This is because the normalizing constant $p(\mathbf{Y})$ ensures that the resulting distribution is a valid probability distribution, and it retains the mixture structure.

$$p_{\mathbf{Z}}(\mathbf{z}) = \sum_{k=1}^{K} \pi_k\ \mathcal{N}(\mathbf{z}|\mu_{\mathbf{Z}_k}, \mathbf{\Sigma}_{\mathbf{Z}_k}) = \sum_{k=1}^{K} \pi_k\ p_{\mathbf{Z}_k}(\mathbf{z})$$

$$p_{\mathbf{Y}|\mathbf{Z}}(\mathbf{y}|\mathbf{z}) = \mathcal{N}(\mathbf{y}|\boldsymbol{W}\mathbf{z} + \mathbf{b}, \mathbf{\Sigma}_{\mathbf{Y}|\mathbf{Z}})$$

$$p_{\mathbf{Z}|\mathbf{Y}}(\mathbf{z}|\mathbf{y}) = \frac{p_{\mathbf{Y}|\mathbf{Z}}(\mathbf{y}|\mathbf{z}) p_{\mathbf{Z}}(\mathbf{z})}{p_{\mathbf{Y}}(\mathbf{y})} = \frac{\sum_{k=1}^{K} \pi_k\ p_{\mathbf{Z}_k}(\mathbf{z})\ p_{\mathbf{Y}|\mathbf{Z}}(\mathbf{y}|\mathbf{z})}{p_{\mathbf{Y}}(\mathbf{y})} = \sum_{k=1}^{K} \frac{\pi_k}{p_{\mathbf{Y}}(\mathbf{y})} \mathcal{N}(\begin{bmatrix} \mathbf{z} \\ \mathbf{y} \end{bmatrix} |\boldsymbol{\mu}_{\mathbf{Y},\mathbf{Z}_k}, \mathbf{\Sigma}_{\mathbf{Y},\mathbf{Z}_k})$$

$$= \sum_{k=1}^{K} \pi_k\ \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\mathbf{Z}|\mathbf{Y}_k}, \mathbf{\Sigma}_{\mathbf{Z}|\mathbf{Y}_k})$$

$$\mathbf{\Sigma}_{\mathbf{Z}|\mathbf{Y}_k}^{-1} = \mathbf{\Sigma}_{\mathbf{Z}_k}^{-1} + \boldsymbol{W}^T \mathbf{\Sigma}_{\mathbf{Y}_k}^{-1} \boldsymbol{W} \quad \text{where} \quad \mathbf{\Sigma}_{\mathbf{Y}_k} = \mathbf{\Sigma}_{\mathbf{Y}|\mathbf{Z}} + \boldsymbol{W} \mathbf{\Sigma}_{\mathbf{Z}_k} \boldsymbol{W}^T$$

$$\mu_{\mathbf{Z}|\mathbf{Y}_k} = \mathbf{\Sigma}_{\mathbf{Z}|\mathbf{Y}_k}[\boldsymbol{W}^T \mathbf{\Sigma}_{\mathbf{Y}_k}^{-1}(\mathbf{y} - \mathbf{b}) + \mathbf{\Sigma}_{\mathbf{Z}_k}^{-1} \mu_{\mathbf{Z}_k}] \quad \text{where} \quad \mathbf{\Sigma}_{\mathbf{Y}_k} = \mathbf{\Sigma}_{\mathbf{Y}|\mathbf{Z}} + \boldsymbol{W} \mathbf{\Sigma}_{\mathbf{Z}_k} \boldsymbol{W}^T$$

Theory Question 3)

1. **Flexibility**: GMM allows for modeling multimodal distributions, meaning it can capture multiple modes or clusters in the data. This flexibility is crucial in situations where the data exhibits multiple underlying patterns or when there are distinct subgroups within the dataset. By using a GMM as a prior for Z, we can effectively model such complex data structures.

2. **Nonparametric Representation**: GMM can be seen as a nonparametric approximation to the true underlying distribution of the data. It achieves this by approximating the data distribution as a weighted sum of Gaussian components. This nonparametric property makes GMMs suitable for situations where the true data distribution is unknown or cannot be accurately represented by a single parametric distribution.

3. **Uncertainty Modeling**: GMM provides a natural way to model uncertainty. Each component in the mixture model represents a separate source of uncertainty, and the weights associated with each component indicate their relative importance. This allows the model to capture different levels of uncertainty in different regions of the data space, which is valuable in many real-world scenarios.

4. **Latent Variable Modeling**: GMMs are commonly used in latent variable models, where Z represents the underlying latent variables that explain the observed data. GMMs offer a convenient framework for capturing the complex relationships between the observed data and the latent variables by modeling the conditional distribution of the observed data given the latent variables.

5. **Computational Efficiency**: GMMs have efficient algorithms for parameter estimation, such as the expectation-maximization (EM) algorithm. These algorithms make it computationally feasible to estimate the parameters of the GMM and infer the latent variables Z from the observed data.

Simulation Question 3)

```
    def denoise(img , W , sigma2_y , gm ): # sigma2_y = (sigma(y))**2
      D = len(W)
      m = int(np.sqrt(D))
      patches = extract_1(img , m)
      N = len(patches)
      l = 28-m+1
      p = gm.predict_proba(patches)
      idx = np.argmax(p,axis=1)
      sigma_z = gm.covariances_
      mu_z = gm.means_
      sigma_y = np.eye(D)*sigma2_y
      denoised_img = np.zeros((28,28))
      count = np.zeros((28,28))
      for n in range(N):
        k = idx[n]
        sigma_z_inv = np.linalg.inv(sigma_z[k,:,:])
        sigma_y_k = sigma_y + W @ sigma_z[k,:,:] @ W.T
        sigma_y_inv = np.linalg.inv(sigma_y_k)
        sigma_zy = np.linalg.inv(sigma_z_inv + W.T @ sigma_y_inv @ W)
        mu_zy = sigma_zy @ (W.T @ sigma_y_inv @ patches[n,:].reshape((D,1)) + sigma_z_inv @
                                            mu_z[k,:].reshape((D,1)))
        a = n//l
        b = n%l
        denoised_img[a:a+m , b:b+m] += mu_zy.reshape((m,m))
        count[a:a+m , b:b+m] += np.ones((m,m))
      for i in range(28):
        for j in range(28):
          denoised_img[i,j] /= count[i,j]
          denoised_img[i,j] *= 255
      return denoised_img
```

Denoise function by giving the corrupted image, W, variance of y and the trained model, First extracts patches of the corrupted image. Then finds the most probable K for each patch with the given model. Next it calculates mean($z|y,k$) and adds the reshaped mean to the result matrix and counts the number of adds to each cell. Finally it divides the result by counts to find the mean value. The result numbers are between 0 and 1 so we also multiply them with 255.

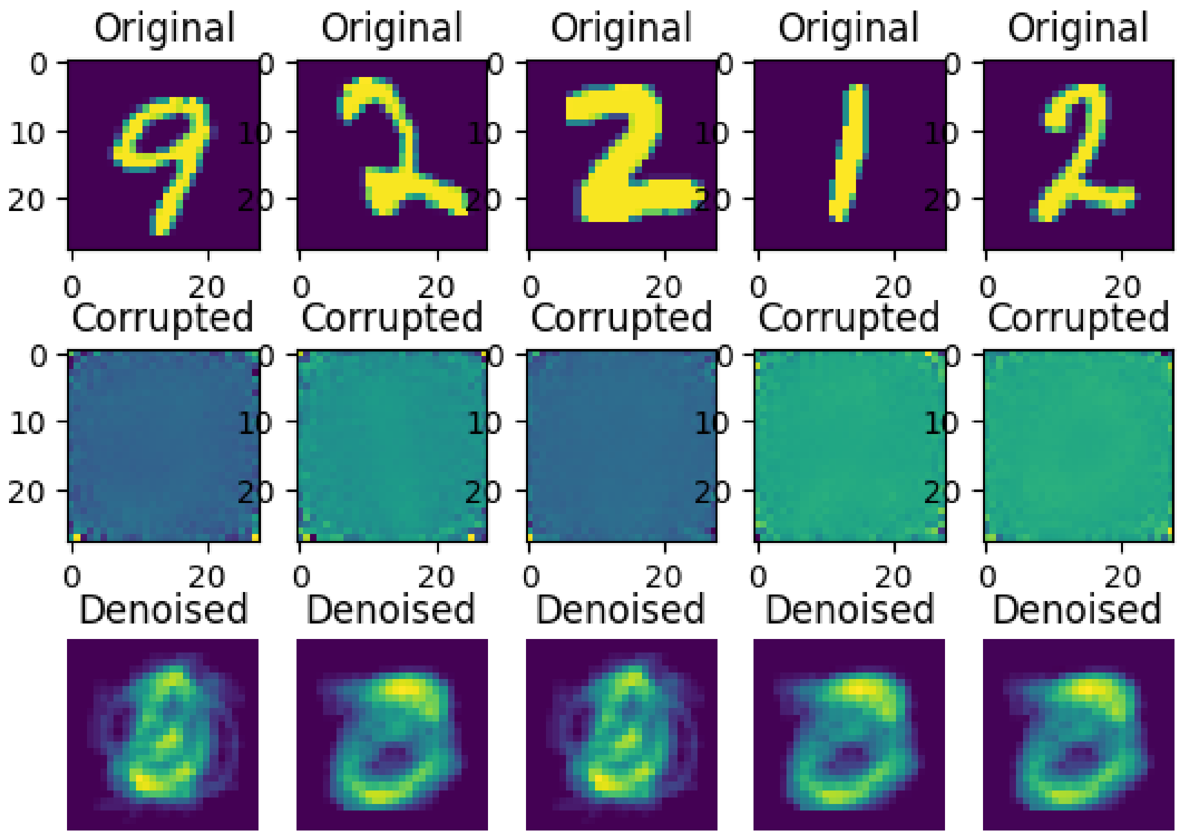
Simulation Question 4)

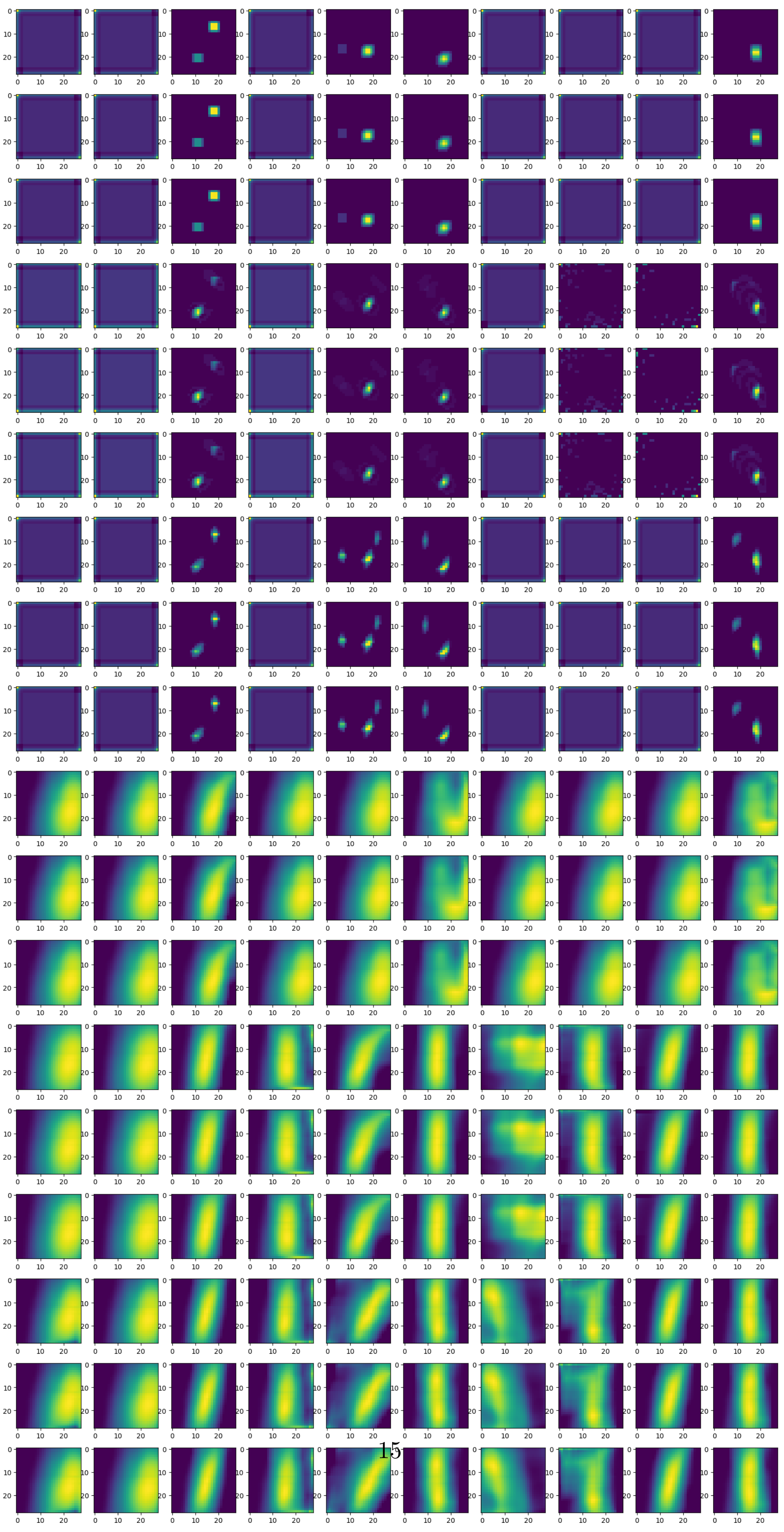
```
def read_zip(zip_filename):
  # Opening the zip file
  with zipfile.ZipFile(zip_filename, "r") as zfile:
    # Getting the list of filenames in the zip file
    list_of_filenames = zfile.namelist()
    original = []
    corrupted = []
    W = None
    for filename in list_of_filenames:
      # Reading the file data as bytes
      data = zfile.read(filename)
      # Checking if the file is an image or a numpy file
        if filename.endswith(".png"):
        # Decoding the image data using cv2.imdecode
        img = cv2.imdecode(np.frombuffer(data, np.uint8), 1)
        # Checking if the image belongs to original or corrupted folder
          if "original" in filename:
            original.append(img)
          elif "corrupted" in filename:
            corrupted.append(img)
        elif filename.endswith(".npy"):
          W = np.load(BytesIO(data))
  return original, corrupted, W

# Initialize m, K, sigma
m_arr = [4, 8, 12, 16, 20, 28]
K_arr = [4, 8, 12, 16, 32]
sigma_arr = [5, 10, 20, 30, 50]

# Find best m
cmp_image = np.zeros(len(m_arr))
i1 = -1
K = 8
sigma = 10
for i in(m_arr):
  i1 += 1
  # Read zip and initialize original and corrupted images and W.npy
  original, corrupted, W = read_zip("MNIST-m=%s.zip" %(i))
  example_image_org = original[10][:,:,0]
  example_image_cor = corrupted[10][:,:,0]
  # training
  patches = extract(data , i)
  gm = train(patches , K)
  # denoise
  denoised_img = denoise(example_image_cor , W , sigma , gm )
  cmp_image[i1] = mean_squared_error(example_image_org, denoised_img)
m_best = np.unravel_index(cmp_image.argmin(), cmp_image.shape)
print("best m =  ", m_arr[m_best[0]])

# Find best K
cmp_image = np.zeros([len(K_arr)])
i1 = -1
m = m_arr[m_best[0]]
# Read zip and initialize original and corrupted images and W.npy
```

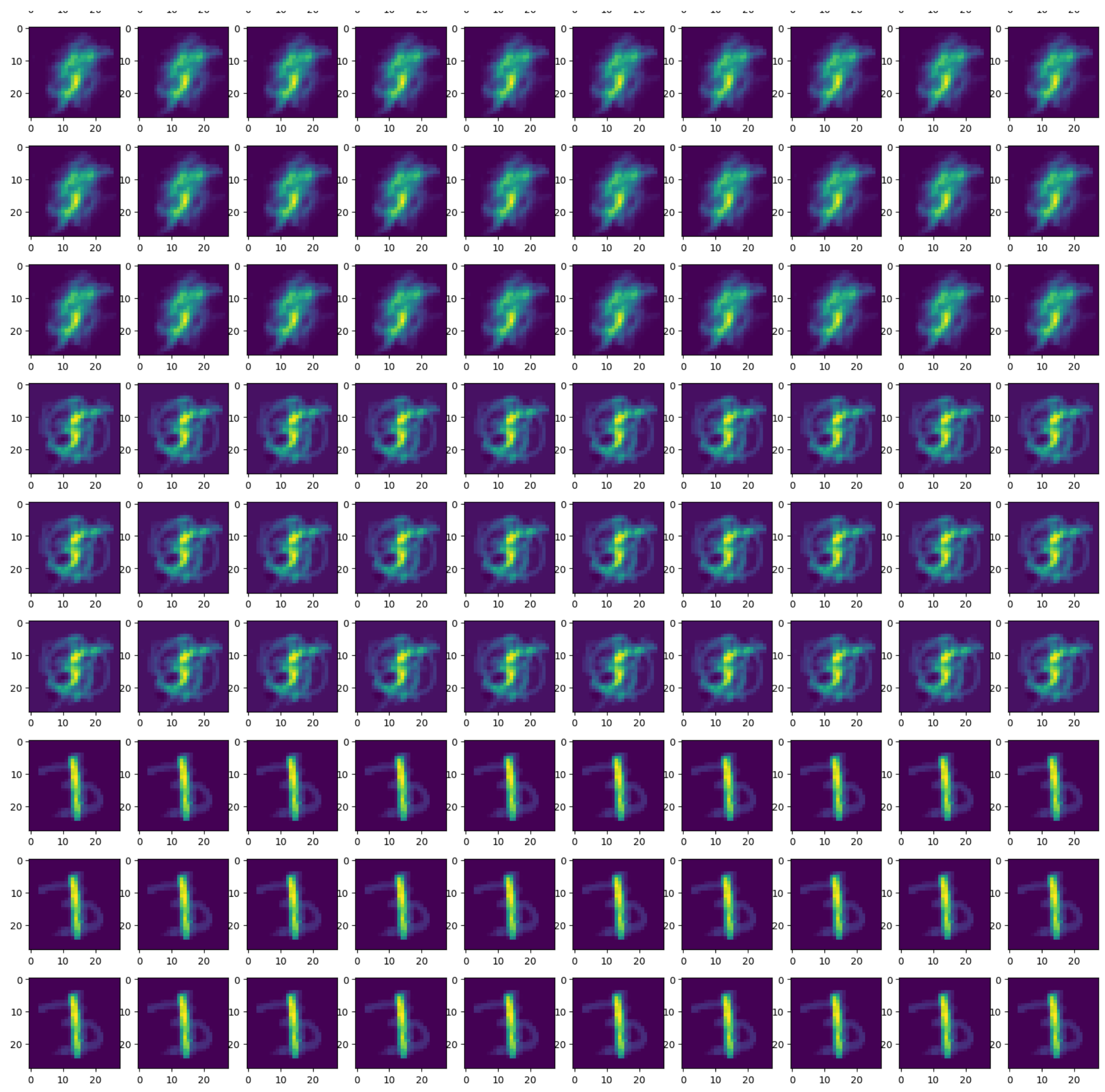
```
original, corrupted, W = read_zip("MNIST-m=%s.zip" %(m))
example_image_org = original[10][:,:,0]
example_image_cor = corrupted[10][:,:,0]
sigma = 10
patches = extract(data , m)
for i in(K_arr):
  i1 += 1
  # training
  gm = train(patches , i)
  # denoise
  denoised_img = denoise(example_image_cor , W , sigma , gm )
  cmp_image[i1] = mean_squared_error(example_image_org, denoised_img)
K_best = np.unravel_index(cmp_image.argmin(), cmp_image.shape)
print("best K = %s " % (K_arr[K_best[0]]))

# Find best sigma
cmp_image = np.zeros([len(sigma_arr)])
i1 = -1
K = K_arr[K_best[0]]
# training
patches = extract(data , m)
gm = train(patches , K)
for i in(sigma_arr):
  i1 += 1
  # denoise
  denoised_img = denoise(example_image_cor , W , i , gm )
  cmp_image[i1] = mean_squared_error(example_image_org, denoised_img)
sigma_best = np.unravel_index(cmp_image.argmin(), cmp_image.shape)
print("best sigma = %s " % (sigma_arr[sigma_best[0]]))

# denoise some corrupted images with best m,K,sigma and plot it
sigma = sigma_arr[sigma_best[0]]
# training
pi = gm.weights_
mu_z = gm.means_
sigma_z = gm.covariances_

# plotting
plt.figure()
i = 0
for img in(corrupted[:5]):
  # denoise image
  denoised_img = denoise(img[:,:,0] , W , sigma , gm)
  plt.subplot(3,5,i+1)
  plt.imshow(original[i][:,:,0])
  plt.title("Original")
  plt.subplot(3,5,i+6)
  plt.imshow(corrupted[i][:,:,0])
  plt.title("Corrupted")
  plt.subplot(3,5,i+11)
  plt.imshow(denoised_img)
  plt.title("Denoised")
  plt.axis("off")
  i += 1
```

Function of read zip by reading the zip file and then checking the type of the existing files and the folder in which the file is located, values the desired variables, and returns them at the end.

```
    best m =   28
    best K = 4
    best sigma = 5
```




0
10
20
0
10
20


0
10
20