Student: Quoc Tuan Vinh, Ngo
ID: 704526

CS-E4820 – Machine Learning: Advanced Probabilistic Methods
Homework Assignment 8

Problem 1. *"ELBO for the simple model ½"*

(a) From the question, we have:

$$\mathcal{L}(q) = \int q(\boldsymbol{Z}) \log \frac{p(\boldsymbol{X}, \boldsymbol{Z})}{q(\boldsymbol{Z})} d\boldsymbol{Z} = E_q[\log p(\boldsymbol{X}, \boldsymbol{Z})] - E_q[\log p(\boldsymbol{Z})]$$

Where **X** denotes observed variables, and **Z** denotes all unobservables (including all **z**). We also have that:

$$\log p(\boldsymbol{X}, \boldsymbol{Z}) = \log p(\boldsymbol{x}, \boldsymbol{z}, \tau, \theta) = \log p(\tau) + \log p(\theta) + \log p(\boldsymbol{z}|\tau) + \log p(\boldsymbol{x}|\boldsymbol{z}, \theta)$$

$$\log p(\boldsymbol{Z}) = \log p(\boldsymbol{z}, \tau, \theta | \boldsymbol{x}) \approx \log q(\tau) + \log q(\theta) + \log q(\boldsymbol{z})$$

Therefore:

$$\begin{aligned}
\mathcal{L}(q) = {} & E_q[\log p(\boldsymbol{X}, \boldsymbol{Z})] - E_q[\log p(\boldsymbol{Z})] \\
& = E_{q(\boldsymbol{z},\tau,\theta)}[\log p(\tau) + \log p(\theta) + \log p(\boldsymbol{z}|\tau) + \log p(\boldsymbol{x}|\boldsymbol{z}, \theta)] \\
& - E_{q(\boldsymbol{z},\tau,\theta)}[\log q(\tau) + \log q(\theta) + \log q(\boldsymbol{z})] \\
& = E_{q(\tau)}[\log p(\tau)] + E_{q(\theta)}[\log p(\theta)] + E_{q(\boldsymbol{z})q(\tau)}[\log p(\boldsymbol{z}|\tau)] \\
& + E_{q(\boldsymbol{z})q(\theta)}[\log p(\boldsymbol{x}|\boldsymbol{z}, \theta)] - E_{q(\boldsymbol{z})}[\log q(\boldsymbol{z})] - E_{q(\tau)}[\log q(\tau)] \\
& - E_{q(\theta)}[\log q(\theta)]
\end{aligned}$$

(b) Derive the 2nd term $E_{q(\theta)}[\log p(\theta)]$ of the ELBO

Since $p(\theta) = N(\theta|0, \beta_0^{-1}) \propto \exp\left(-\frac{\beta_0}{2}\theta^2\right)$

$$E_{q(\theta)}[\log p(\theta)] \propto E_{q(\theta)}\left(-\frac{\beta_0}{2}\theta^2\right) = -\frac{\beta_0}{2} E_{q(\theta)}(\theta^2)$$

We know that $q(\theta) = N(\theta|m_2, \beta_2^{-1})$, and $E(X^2) \Leftrightarrow E(X^2) = Var(X) + E(X)^2$ so:

$$E_{q(\theta)}[\log p(\theta)] \propto -\frac{\beta_0}{2} E_{q(\theta)}(\theta^2) = -\frac{\beta_0}{2} (\beta_2^{-1} + m_2^2)$$

(c) Find out the 7th term $E_{q(\theta)}[\log q(\theta)]$

Similarly to 6th term, this is basically the negative entropy of $q(\theta) = N(\theta|m_2, \beta_2^{-1})$, and according to Wiki[1], it is as following:

$$E_{q(\theta)}[\log q(\theta)] = -\frac{1}{2}\log(2\pi e \beta_2^{-1})$$

---

[1] https://en.wikipedia.org/wiki/Normal_distribution

Problem 2. *"ELBO for the simple model 2/2"*

(a) *Derive the 4th term $E_{q(\mathbf{z})q(\theta)}[log\, p(\boldsymbol{x}|\mathbf{z},\theta)]$ of the ELBO*

We know that: $p(\boldsymbol{x}|\mathbf{z},\theta) = \prod_{n=1}^{N} N(x_n|0,1)^{z_{n1}} N(x_n|\theta,1)^{z_{n2}}$

Therefore:

$$E_{q(\mathbf{z})q(\theta)}[log\, p(\boldsymbol{x}|\mathbf{z},\theta)] = E_{q(\mathbf{z})q(\theta)}\left[log \prod_{n=1}^{N} N(x_n|0,1)^{z_{n1}} N(x_n|\theta,1)^{z_{n2}}\right]$$

$$= \sum_{n=1}^{N} E_{q(\mathbf{z})q(\theta)}[z_{n1}\log N(x_n|0,1) + z_{n2}\log N(x_n|\theta,1)]$$

$$= \sum_{n=1}^{N}\{E_{q(\mathbf{z})}(z_{n1}) \times \log N(x_n|0,1) + E_{q(\mathbf{z})}(z_{n2}) \times E_{q(\theta)}(\log N(x_n|\theta,1))\}$$

$$= \sum_{n=1}^{N}\left\{r_{n1} \times \left[\log\frac{1}{\sqrt{2\pi}} + (\frac{-x_n^2}{2})\right]\right.$$

$$\left. + r_{n2} \times \left[\log\frac{1}{\sqrt{2\pi}} + E_{q(\theta)}\frac{-(x_n-\theta)^2}{2}\right]\right\}$$

$$= \log\frac{1}{\sqrt{2\pi}}\sum_{n=1}^{N}(r_{n1}+r_{n2}) - \frac{1}{2}\sum_{n=1}^{N} r_{n1}x_n^2 - \frac{1}{2}\sum_{n=1}^{N} r_{n2} \times E_{q(\theta)}\,(x_n-\theta)^2$$

$$= -\frac{N}{2}\log 2\pi - \frac{1}{2}\sum_{n=1}^{N} r_{n1}x_n^2 - \frac{1}{2}\sum_{n=1}^{N} r_{n2} \times E_{q(\theta)}\,(x_n^2 + \theta^2 - 2\theta x_n)$$

With the last term, we can have the following:

$E_{q(\theta)}\,(x_n^2 + \theta^2 - 2\theta x_n) = x_n^2 + E_{q(\theta)}(\theta^2) - 2\,x_n E_{q(\theta)}(\theta)$

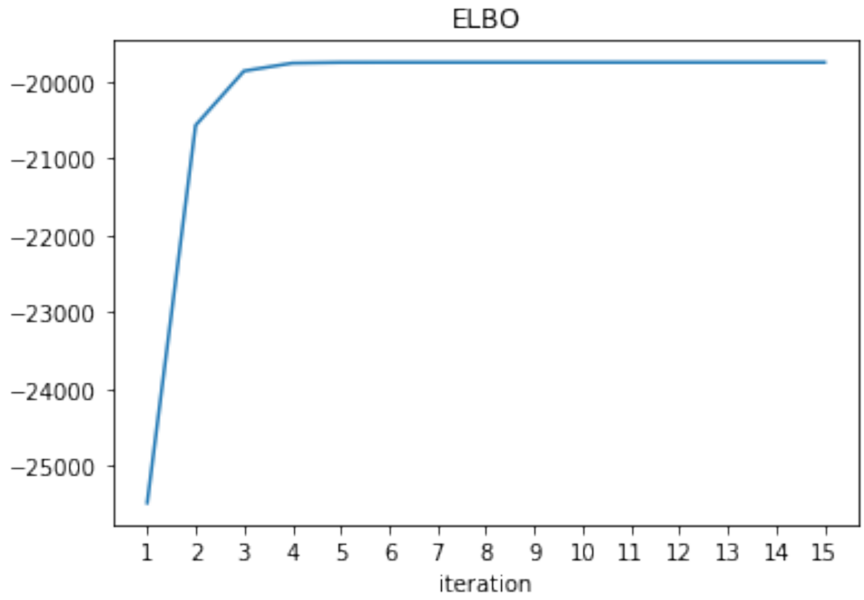
Remember that: $E_{q(\theta)}(\theta^2) = \beta_2^{-1} + m_2^2$ and $E_{q(\theta)}(\theta) = m_2$
Therefore:

$$E_{q(\theta)}\,(x_n^2 + \theta^2 - 2\theta x_n) = x_n^2 x_n^2 + \beta_2^{-1} + m_2^2 - 2\,x_n m_2 = (x_n - m_2)^2 + \beta_2^{-1}$$

And:

$$E_{q(\mathbf{z})q(\theta)}[log\, p(\boldsymbol{x}|\mathbf{z},\theta)] = -\frac{N}{2}\log 2\pi - \frac{1}{2}\sum_{n=1}^{N} r_{n1}x_n^2 - \frac{1}{2}\sum_{n=1}^{N} r_{n2} \times [(x_n - m_2)^2 + \beta_2^{-1}]$$

(b) Implement terms 2,4,7



```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(123123123)

# Compute ELBO for the model described in simple_elbo.pdf
def compute_elbo(alpha_tau, beta_tau, r1, r2, m2, beta2, alpha0, beta0, x):

    from scipy.special import psi, gammaln # digamma function, logarithm of gamma function

    # E[log p(tau)]
    term1 = (alpha0 - 1) * (psi(alpha_tau) + psi(beta_tau) - 2 * psi(alpha_tau + beta_tau))

    # E[log p(theta)]
    term2 = -1/2 * beta0 * ( beta2**(-1) + m2**2)# EXERCISE

    # E[log p(z|tau)]
    N2 = np.sum(r2); N1 = np.sum(r1); N = N1 + N2
    term3 = N2 * psi(alpha_tau) + N1 * psi(beta_tau) - N * psi(alpha_tau + beta_tau)

    # E[log p(x|z,theta)]
    term4 = -N/2 * np.log(2*np.pi) - 1/2 * np.sum(r1 * x**2) - 1/2 * np.sum(r2 *((x - m2)**2 + 
beta2**(-1) )) # EXERCISE

    # Negative entropy of q(z)
    term5 = np.sum(r1 * np.log(r1)) + np.sum(r2 * np.log(r2))

    # Negative entropy of q(tau)
    term6 = (gammaln(alpha_tau + beta_tau) - gammaln(alpha_tau) - gammaln(beta_tau)
      + (alpha_tau - 1) * psi(alpha_tau) + (beta_tau - 1) * psi(beta_tau)
      - (alpha_tau + beta_tau - 2) * psi(alpha_tau + beta_tau))

    # Negative entropy of q(theta)
    term7 = -1/2 * np.log(2*np.pi*np.e* beta2**(-1))# EXERCISE

    elbo = term1 + term2 + term3 + term4 - term5 - term6 - term7

    return elbo


# Simulate data
```

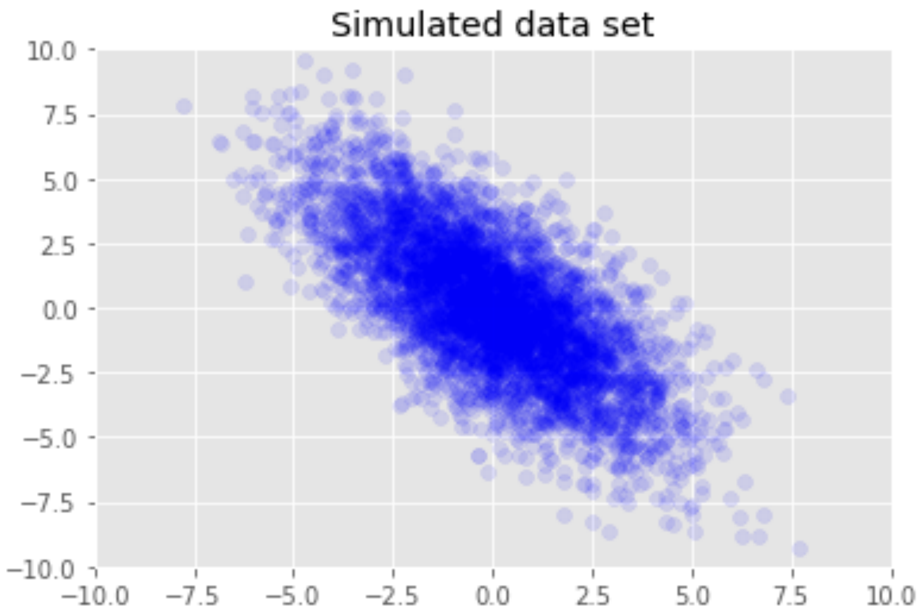
```
theta_true = 4
tau_true = 0.3
n_samples = 10000
z = (np.random.rand(n_samples) < tau_true)  # True with probability tau_true
x = np.random.randn(n_samples) + z * theta_true

# Parameters of the prior distributions.
alpha0 = 0.5
beta0 = 0.2

n_iter = 15 # The number of iterations
elbo_array = np.zeros(n_iter) # To track the elbo

# Some initial value for the things that will be updated
E_log_tau = -0.7   # E(log(tau))
E_log_tau_c = -0.7  # E(log(1-tau))
E_log_var = 4 * np.ones(n_samples)  # E((x_n-theta)^2)
r2 = 0.5 * np.ones(n_samples)  # Responsibilities of the second cluster.

for i in range(n_iter):

    # Updated of responsibilites, factor q(z)
    log_rho1 = E_log_tau_c - 0.5 * np.log(2 * np.pi) - 0.5 * (x ** 2)
    log_rho2 = E_log_tau - 0.5 * np.log(2 * np.pi) - 0.5 * E_log_var
    max_log_rho = np.maximum(log_rho1, log_rho2)  # Normalize to avoid numerical problems when
exponentiating.
    rho1 = np.exp(log_rho1 - max_log_rho)
    rho2 = np.exp(log_rho2 - max_log_rho)
    r2 = rho2 / (rho1 + rho2)
    r1 = 1 - r2

    N1 = np.sum(r1)
    N2 = np.sum(r2)

    # Update of factor q(tau)
    from scipy.special import psi # digamma function
    E_log_tau = psi(N2 + alpha0) - psi(N1 + N2 + 2*alpha0)
    E_log_tau_c = psi(N1 + alpha0) - psi(N1 + N2 + 2*alpha0)

    # Update of factor q(theta)
    x2_avg = 1 / N2 * np.sum(r2 * x)
    beta_2 = beta0 + N2
    m2 = 1 / beta_2 * N2 * x2_avg
    E_log_var = (x - m2) ** 2 + 1 / beta_2

    # Keep track of the current estimates
    tau_est = (N2 + alpha0) / (N1 + N2 + 2*alpha0)
    theta_est = m2

    # Compute ELBO
    alpha_tau = N2 + alpha0
    beta_tau = N1 + alpha0
    elbo_array[i] = compute_elbo(alpha_tau, beta_tau, r1, r2, m2, beta_2, alpha0, beta0, x)

# Plot ELBO as a function of iteration
plt.plot(np.arange(n_iter) + 1, elbo_array)
plt.xticks(np.arange(n_iter) + 1)
plt.xlabel("iteration")
plt.title("ELBO")
plt.show()
```

Problem 3. *“Factor analysis in Edward”* *(a) Modify the code that simulates data set*

```
def build_toy_dataset(N, D, K):
    x_train = np.zeros((D, N))
    w = np.random.normal(0.0, 2.0, size=(D, K))
    z = np.random.normal(0.0, 1.0, size=(K, N))
    mean = np.dot(w, z)
    psi = np.exp(np.random.normal(size = D)) # ?
    epsilon = np.diag(psi) # ?
    for d in range(D):
        for n in range(N):
            x_train[d, n] = np.random.normal(mean[d, n], epsilon[d,d]) # ?

    print('True principal axes')
    print(w)
    return x_train, epsilon

N = 5000
D = 2 # Data dimension
K = 1 # Latent dimension

x_train, true_noise_std = build_toy_dataset(N, D, K)

plt.scatter(x_train[0, :], x_train[1, :], color='blue', alpha=0.1)
plt.axis([-10, 10, -10, 10])
plt.title('Simulated data set')
plt.show()
```

We first need to create a psi vector with length = D for the diagonal. To do so, I generated a normally distributed vector (mean = 0, std =1, which is the default setup of *np.random.normal*), and took the exponential for each element to ensure that the values are positive. Then the final noise matrix is created by making a diagonal matrix from the psi vector above. Simulated the data set with seed 123, we have below results:

```
True principal axes
[[-2.17126121]
 [ 1.99469089]]
```



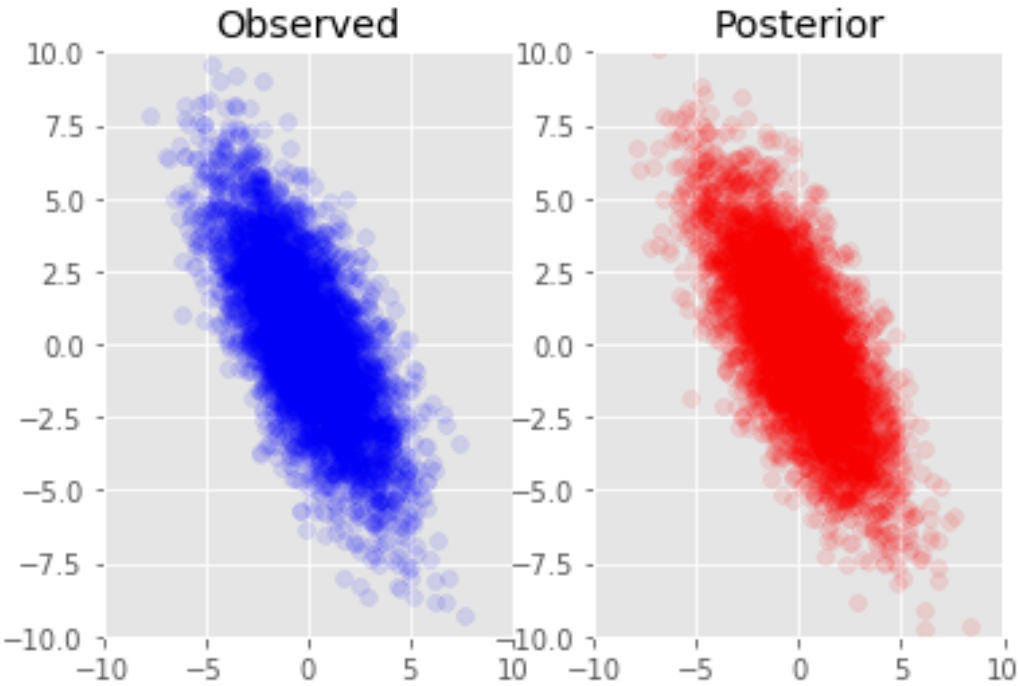
(b) Modify the model description:

```
# MODEL
np.random.seed(123)
noise_std = tf.diag(tf.exp(tf.Variable(tf.random_normal([D])))) #?
w = Normal(loc=tf.zeros([D, K]), scale=2.0 * tf.ones([D, K])) # prior on w
z = Normal(loc=tf.zeros([N, K]), scale=tf.ones([N, K])) # prior on z
x = Normal(loc=tf.matmul(w, z, transpose_b=True), scale= noise_std @ tf.ones([D,N])) # likelihood
# ?
# transpose_b=True transposes the second argument


# INFERENCE
qw = Normal(loc=tf.Variable(tf.random_normal([D, K])),
scale=tf.nn.softplus(tf.Variable(tf.random_normal([D, K]))))
qz = Normal(loc=tf.Variable(tf.random_normal([N, K])),
scale=tf.nn.softplus(tf.Variable(tf.random_normal([N, K]))))

inference = ed.KLqp({w: qw, z: qz}, data={x: x_train}) # Note: noise std is not updated
inference.run(n_iter=500, n_print=100, n_samples=10)

# CRITICISM
print('Inferred principal axes:')
print(qw.mean().eval())

x_post = ed.copy(x, {w: qw, z: qz}) # Simulate x_post similarly to x, but use learned z and w
x_gen = x_post.sample().eval()

print('Inferred noise_std')
print(noise_std.eval())

print('True noise_std')
print(true_noise_std)

# VISUALIZATION
def visualise(x_data, y_data, ax, color, title):
    ax.scatter(x_data, y_data, color=color, alpha=0.1)
    ax.axis([-10, 10, -10, 10])
    ax.set_title(title)

fig = plt.figure()
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)

visualise(x_train[0, :], x_train[1, :], ax1, 'blue', 'Observed')
visualise(x_gen[0, :], x_gen[1, :], ax2, 'red', 'Posterior')

plt.show()
```

The noise_std is also modified with the exact same logic as in (*a*). In x, the scale is also adjusted accordingly. Finally, I printed out the true noise_std and inferred noise_std.

```
500/500 [100%] ██████████████████████████████ Elapsed: 23s | Loss: 21461.45
9
Inferred principal axes:
[[-1.79451835]
 [ 2.28938532]]
```

```
Inferred noise_std
[[ 1.19358337  0.        ]
 [ 0.          1.48720729]]
True noise_std
[[ 0.32089966  0.        ]
 [ 0.          1.90115179]]
```



Comparing between true and inferred parameters, I can witness some certain de viations. That is, the values are not very close to each other. Therefore, I have an impression that Edward does not do very well in this case.