Student: Quoc Tuan Vinh, Ngo
ID: 704526

CS-E4820 – Machine Learning: Advanced Probabilistic Methods
Homework Assignment 5

Problem 1. *"EM for missing observations"*

From question, we have: $X_i \sim \mathcal{N}_2(0,\ \Sigma)$, where $\Sigma = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}$.
Then we have three observations X_1, X_2, X_3 that are stacked vertically to make a matrix, in which the first coordinate of second observation is unknown. Denote Z as the missing observation. Here, we would like to learn the unknown ρ using EM-algorithm with the following E-step:

(1) Write complete data log-likelihood:

$$\ell(\rho) = \log \prod_{i=1}^{3} p(x_i|\rho) = \sum_{i=1}^{3} \log p(x_i|\rho) = \sum_{i=1}^{3} \log \mathcal{N}_2(x_i \mid 0, \Sigma)$$
$$= \sum_{i=1}^{3} \log \frac{\det \Sigma^{-\frac{1}{2}}}{\sqrt{2\pi}}\, e^{-\frac{1}{2} x_i^T \Sigma^{-1} x_i}$$
$$= -\frac{1}{2}\, x_1^T \Sigma^{-1} x_1 - \frac{1}{2}\, x_2^T \Sigma^{-1} x_2 - \frac{1}{2}\, x_3^T \Sigma^{-1} x_3 - \frac{3}{2}\log(1-\rho^2) + const$$

(2) Compute posterior distribution of the missing observation

In general, for $X \sim \mathcal{N}_2(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $X = (X_1, X_2)^T$, $\boldsymbol{\mu} = (\mu_1, \mu_2)^T$ and $\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}$, we have

$$X_1|X_2 = x_2 \sim \mathcal{N}\left(\mu_1 + \frac{\sigma_1}{\sigma_2}\rho(x_2 - \mu_2), (1-\rho^2)\sigma_1^2\right),$$

with ρ being the correlation coefficient.

Thus:
$$p(Z|x_{22}) = \mathcal{N}(Z|\, 0 + \frac{1}{1}\rho(x_{22} - 0), (1-\rho^2)1^2) = \mathcal{N}(Z|\, \rho x_{22}, (1-\rho^2))$$

Therefore, $E(Z) = \rho x_{22}$, and $E(Z^2) = \rho^2 {x_{22}}^2 + 1 - \rho^2$

(3) Evaluate expectation of $\ell(\rho)$

Since $\ell(\rho)$ only depends on expectation of $x_2^T \Sigma^{-1} x_2$, we can have the following:

$$
\begin{aligned}
Q(\rho,\rho_0) &= E_{Z|X,\rho_0}\ell(\rho) \\
&= E\left(-\frac{1}{2}\,x_1^T\Sigma^{-1}x_1\; -\frac{1}{2}\,x_2^T\Sigma^{-1}x_2\; -\frac{1}{2}\,x_3^T\Sigma^{-1}x_3 - \frac{3}{2}\log(1-\rho^2) + const\right) \\
&= -\frac{1}{2}E(\,x_2^T\Sigma^{-1}x_2\,)\; + const = -\frac{1}{2}E(trace(\Sigma^{-1}x_2x_2^T) + const \\
&= -\frac{1}{2}\,trace(\Sigma^{-1}\begin{bmatrix} E(Z^2) & E(Z)x_{22} \\ E(Z)x_{22} & x_{22}^2 \end{bmatrix}) + const \\
&= -\frac{1}{2}\,trace\left(\begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}^{-1}\cdot\begin{bmatrix} \rho^2{x_{22}}^2 + 1-\rho^2 & \rho{x_{22}}^2 \\ \rho{x_{22}}^2 & x_{22}^2 \end{bmatrix}\right) + const \\
&= -\frac{1}{2}\,trace\left(\frac{1}{1-\rho^2}\begin{bmatrix} 1 & -\rho \\ -\rho & 1 \end{bmatrix}\cdot\begin{bmatrix} \rho^2{x_{22}}^2 + 1-\rho^2 & \rho{x_{22}}^2 \\ \rho{x_{22}}^2 & x_{22}^2 \end{bmatrix}\right) + const \\
&= -\frac{1}{2}trace\left(\frac{1}{1-\rho^2}\begin{bmatrix} 1-\rho^2 & \dots \\ \dots & -\rho^2{x_{22}}^2 + x_{22}^2 \end{bmatrix}\right) + const \\
&= -\frac{1}{2}\,(1+\,x_{22}^2)\; + const
\end{aligned}
$$

Problem 2. *"Extension of the simple example from lecture"*

$$p(x_n \mid \theta,\tau) = (1-\tau)\mathcal{N}(x_n|0,1) + \tau\,\mathcal{N}(x_n|\theta,1)$$

(a) *Write down complete log-likelihood´*

First we formulate the model using the **latent variable representation**, and introduce variables $\mathbf{z} = (z_1,\dots,z_N)$ which explicitly specify the component responsible for generating observation x_n. In detail:

$$z_n = (z_{n1}, z_{n2})^T = \begin{cases} (1,0)^T, & (x_n \text{ is from } N(x_n|0,1)) \\ (0,1)^T, & (x_n \text{ is from } N(x_n|\theta,1)) \end{cases}.$$

$$
\begin{aligned}
\log p(\boldsymbol{x},\boldsymbol{z}|\theta,\tau) &= \log\left\{\prod_{n=1}^{N} p(x_n,z_n|\theta,\tau)\right\} = \sum_{n=1}^{N}\log p(x_n,z_n|\theta,\tau) \\
&= \sum_{n=1}^{N}\log(\,(1-\tau)^{z_{n1}}\,\times\,\mathcal{N}(x_n|0,1)^{z_{n1}}\,\times\tau^{z_{n2}}\,\times\,\mathcal{N}(x_n|\theta,1)^{z_{n2}}) \\
&= \sum_{n=1}^{N}(z_{n1}\log\mathcal{N}(x_n|0,1)\; +\; z_{n2}\log\mathcal{N}(x_n|\theta,1) + \; z_{n1}\log(1-\tau) \\
&+\; z_{n2}\log\tau)
\end{aligned}
$$

Derive the EM-algorithm:

E-step 1: Compute the posterior distribution of the latent variables, given current estimates of θ_0 and τ_0:

$$\gamma(z_{n2})\; \equiv p\,(z_{n2}=1|x_n,\theta_0,\tau_0) = \frac{\tau_0\,\mathcal{N}(x_n|\theta_0,1)}{(1-\,\tau_0)\mathcal{N}(x_n|0,1) + \,\tau_0\,\mathcal{N}(x_n|\theta_0,1)}$$

E-step 2: Evaluate the expectation of the complete log-likelihood over the posterior distribution of the latent variables:

$$
\begin{aligned}
Q(\theta,\tau,\theta_0,\tau_0) &= E_{z|\,x,\theta_0,\tau_0}(\log p(x,z|\theta,\tau)) \\
&= \sum_{n=1}^{N}(E(z_{n1})\log\mathcal{N}(x_n|0,1) + E(z_{n2})\log\mathcal{N}(x_n|\theta,1) \\
&+ E(z_{n1}\log(1-\tau)) + E(z_{n2}\log\tau)\,) \\
&= \sum_{n=1}^{N}\{\big(1-\gamma(z_{n2})\big)\log\mathcal{N}(x_n|0,1) + \gamma(z_{n2})\log\mathcal{N}(x_n|\theta,1) \\
&+ \big(1-\gamma(z_{n2})\big)\log(1-\tau) + \gamma(z_{n2})\log\tau\,)\}
\end{aligned}
$$

M-Step: Maximize $Q(\theta,\tau,\theta_0,\tau_0)$ with respect to θ,τ

First, derive the $Q(\theta,\tau,\theta_0,\tau_0)$ with respect to θ

$$
\begin{aligned}
&\frac{d}{d\theta}Q(\theta,\tau,\theta_0,\tau_0) \\
&= \frac{d}{d\theta}\sum_{n=1}^{N}(\,(1-\gamma(z_{n2})\log\mathcal{N}(x_n|0,1) + \gamma(z_{n2})\log\mathcal{N}(x_n|\theta,1) + (1 \\
&- \gamma(z_{n2})\log(1-\tau) + \gamma(z_{n2})\log\tau\,) = \sum_{n=1}^{N}\gamma(z_{n2})(x_n-\theta)
\end{aligned}
$$

Set $\frac{d}{d\theta}Q(\theta,\tau,\theta_0,\tau_0)=0$, we get:

$$\theta = \frac{\sum_{n=1}^{N}\gamma(z_{n2})x_n}{\sum_{n=1}^{N}\gamma(z_{n2})} = \frac{1}{N_2}\sum_{n=1}^{N}\gamma(z_{n2})x_n$$

where $N_2=\sum_{n=1}^{N}\gamma(z_{n2})$ being interpreted as the effective number of observations assigned to component 2.

Secondly, derive the $Q(\theta,\tau,\theta_0,\tau_0)$ with respect to τ:

$$
\begin{aligned}
&\frac{d}{d\tau}Q(\theta,\tau,\theta_0,\tau_0) \\
&= \frac{d}{d\tau}\sum_{n=1}^{N}(\,(1-\gamma(z_{n2})\log\mathcal{N}(x_n|0,1) + \gamma(z_{n2})\log\mathcal{N}(x_n|\theta,1) + (1 \\
&- \gamma(z_{n2})\log(1-\tau) + \gamma(z_{n2})\log\tau\,)
\end{aligned}
$$

Using Lagrange multiplier and set the derivation to 0, we can obtain:

$$\tau = \frac{N_2}{N}$$

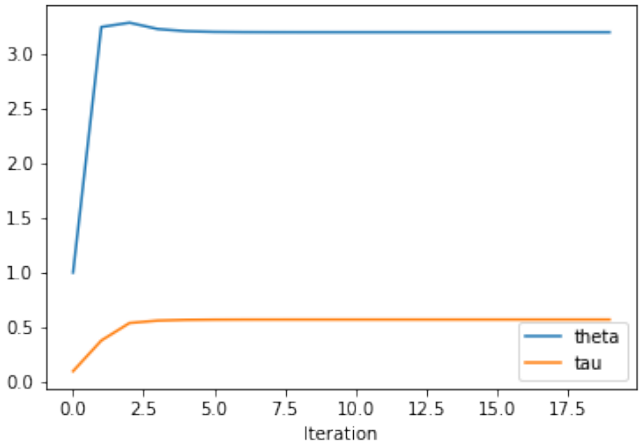
(b) *Implement EM-algorithm*

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


### Simulate data:

np.random.seed(0)
```

```
theta_true = 3
tau_true = 0.5
n_samples = 100

x = np.zeros(n_samples)
for i in range(n_samples):
    # Sample from N(0,1) or N(theta_true,1)
    if np.random.rand() < 1 - tau_true:
        x[i] = np.random.normal(0, 1)
    else:
        x[i] = np.random.normal(theta_true, 1)


### The EM algorithm:

n_iter = 20
theta = np.zeros(n_iter)
tau = np.zeros(n_iter)

# Initial guesses for theta and tau
theta[0] = 1
tau[0] = 0.1

for it in range(1, n_iter):
    # The current estimates for theta and tau,
    # computed in the previous iteration
    theta_0 = theta[it-1]
    tau_0 = tau[it-1]

    # E-step: compute the responsibilities r1 and r2
    r1_unnorm = (1 - tau_0)*scipy.stats.norm.pdf(x, 0, 1)# EXERCISE
    r2_unnorm = (tau_0) * scipy.stats.norm.pdf(x, theta_0, 1) # EXERCISE
    r1 = r1_unnorm / (r1_unnorm + r2_unnorm)
    r2 = r2_unnorm / (r1_unnorm + r2_unnorm)

    # M-step: compute the parameter values that maximize
    # the expectation of the complete-data log-likelihood.
    theta[it] = sum(r2 * x) / sum(r2)# EXERCISE
    tau[it] = np.mean(r2) # EXERCISE


# Print and plot the values of theta and tau in each iteration
print("theta      tau")
for theta_i, tau_i in zip(theta, tau):
    print("{0:.7f}  {1:.7f}".format(theta_i, tau_i))

plt.plot(range(n_iter), theta, label = 'theta')
plt.plot(range(n_iter), tau, label = 'tau')
plt.xlabel('Iteration')
plt.legend()
plt.show()
```

```
theta       tau
1.0000000  0.1000000
3.2393002  0.3798055
3.2787207  0.5396835
3.2208416  0.5618515
3.2012856  0.5684228
3.1949151  0.5705275
3.1928396  0.5712102
3.1921631  0.5714324
3.1919426  0.5715048
3.1918707  0.5715284
```

```
3.1918472   0.5715361
3.1918396   0.5715386
3.1918371   0.5715394
3.1918363   0.5715397
3.1918360   0.5715397
3.1918359   0.5715398
3.1918359   0.5715398
3.1918359   0.5715398
3.1918359   0.5715398
3.1918359   0.5715398
```



Problem 3. *"Probabilistic programming with Edward"*

1. What is a probabilistic programming language?

A probabilistic programming language (PPL) is a programming language designed to describe probabilistic models and then perform inference in those models. PPLs are closely related to graphical models and Bayesian networks, but are more expressive and flexible.

PPLs often extend from a basic language. The choice of underlying basic language depends on the similarity of the model to the basic language's ontology, as well as commercial considerations and personal preference.

Probabilistic programing languages unify techniques for formal description of computation with the representation and use of uncertain knowledge. Probabilistic reasoning is a foundational technology of machine learning. It is used by companies such as Google, Amazon.com and Microsoft. Probabilistic reasoning has been used for predicting stock prices, recommending movies, diagnosing computers, detecting cyber intrusions and image detection.

2. Why is probabilistic programming needed?

Probabilistic modeling is a powerful approach for analyzing empirical information. Probabilistic models are essential to fields related to its methodology, such as statistics and machine learning, as well as fields related to its application, such as computational biology, computational neuroscience, cognitive science, information theory, and natural language processing.

Probabilistic programming creates systems that help make decisions in the face of uncertainty. Probabilistic reasoning combines knowledge of a situation with the laws of

probability. Until recently, probabilistic reasoning systems have been limited in scope, and have not successfully addressed real world situations. Probabilistic programming is a new approach that makes probabilistic reasoning systems easier to build and more widely applicable. Reasoning about variables as probability distributions causes difficulties for novice programmers, but these difficulties can be addressed through use of Bayesian network visualisations and graphs of variable distributions embedded within the source code editor.

Software systems for probabilistic modeling provide new and faster ways of experimentation. This enables research advances in probabilistic modeling that could not have been completed before.

3. What is Edward?

Edward is a library for probabilistic modeling. Probabilistic modeling in Edward uses a simple language of random variables. Edward is a Python library for probabilistic modeling, inference, and criticism. It is a testbed for fast experimentation and research with probabilistic models, ranging from classical hierarchical models on small data sets to complex deep probabilistic models on large data sets.

It is named after the statistician George Edward Pelham Box. Edward is built around an iterative process for probabilistic modeling, pioneered by Box and his collaborators.

Edward is built on top of TensorFlow, a library for numerical computing using data flow graphs. TensorFlow enables Edward to speed up computation with hardware such as GPUs, to scale up computation with distributed training, and to simplify engineering effort with automatic differentiation.

4. What are the three steps in the iterative process for probabilistic modeling that Edward is built around?

Step 1: Formulate a model of the phenomena. It supports **modeling** with
- Directed graphical models
- Neural networks (via libraries such as tf.layers and Keras)
- Implicit generative models
- Bayesian nonparametrics and probabilistic programs

Step 2: Use an algorithm to infer the model's hidden structure, thus reasoning about the phenomena. It supports **inference** with
- Variational inference
- Black box variational inference
- Stochastic variational inference
- Generative adversarial networks
- Maximum a posteriori estimation
- Monte Carlo
- Gibbs sampling
- Hamiltonian Monte Carlo
- Stochastic gradient Langevin dynamics
- Compositions of inference
- Expectation-Maximization
- Pseudo-marginal and ABC methods
- Message passing algorithms

Step 3: Criticize how well the model captures the data's generative process. It supports **criticism** of the model and inference with

- Point-based evaluations
- Posterior predictive checks

As we criticize our model's fit to the data, we revise components of the model and repeat to form an iterative loop.

5. Identify parts of the example probabilistic program (on p. 3–4) that correspond to the three steps mentioned in the previous question.

Step 1 – Modeling: Simulate a toy dataset of 50 observations with a cosine relationship.

```
import numpy as np
x_train = np.linspace(-3, 3, num=50)
y_train = np.cos(x_train) + np.random.normal(0, 0.1, size=50)
x_train = x_train.astype(np.float32).reshape((50, 1))
y_train = y_train.astype(np.float32).reshape((50, 1))
```

Next, define a two-layer Bayesian neural network. Here, we define the neural network manually with tanh nonlinearities.

```
import tensorflow as tf
from edward.models import Normal
W_0 = Normal(mu=tf.zeros([1, 2]), sigma=tf.ones([1, 2]))
W_1 = Normal(mu=tf.zeros([2, 1]), sigma=tf.ones([2, 1]))
b_0 = Normal(mu=tf.zeros(2), sigma=tf.ones(2))
b_1 = Normal(mu=tf.zeros(1), sigma=tf.ones(1))
x = x_train
y = Normal(mu=tf.matmul(tf.tanh(tf.matmul(x, W_0) + b_0), W_1) + b_1,sigma=0.1)
```

Step 2 – Inferences: Next, make inferences about the model from data. We will use variational inference. Specify a normal approximation over the weights and biases.

```
qW_0 = Normal(mu=tf.Variable(tf.zeros([1, 2])),
          sigma=tf.nn.softplus(tf.Variable(tf.zeros([1, 2]))))
qW_1 = Normal(mu=tf.Variable(tf.zeros([2, 1])),
          sigma=tf.nn.softplus(tf.Variable(tf.zeros([2, 1]))))
qb_0 = Normal(mu=tf.Variable(tf.zeros(2)),
          sigma=tf.nn.softplus(tf.Variable(tf.zeros(2))))
qb_1 = Normal(mu=tf.Variable(tf.zeros(1)),
          sigma=tf.nn.softplus(tf.Variable(tf.zeros(1))))
```

Defining tf.Variable allows the variational factors' parameters to vary. They are all initialized at 0. The standard deviation parameters are constrained to be greater than zero according to a softplus transformation[2].

Now, run variational inference with the Kullback-Leibler divergence in order to infer the model's latent variables given data. We specify 1000 iterations.

```
import edward as ed
inference = ed.KLqp({W_0: qW_0, b_0: qb_0,
                     W_1: qW_1, b_1: qb_1}, data={y: y_train})
inference.run(n_iter=1000)
```

Step 3 – Criticism: Finally, criticize the model fit. Bayesian neural networks define a distribution over neural networks, so we can perform a graphical check. Draw neural networks from the inferred model and visualize how well it fits the data.

The model has captured the cosine relationship between x and y in the observed domain.