MathDNN Homework 11

Department of Computer Science and Engineering
2021-16988 Jaewan Park

Problem 1

(a) Since log is a concave function, using Jensen's inequality, we obtain the following.

$$\begin{aligned}
\mathrm{VLB}^{(K)}_{\theta,\phi}(x) &= \mathbb{E}_{Z_1,\cdots,Z_K\sim q_\phi(z|x)}\left[\log\frac{1}{K}\sum_{k=1}^{K}\frac{p_\theta(x\,|\,Z_k)p_Z(Z_k)}{q_\phi(Z_k\,|\,x)}\right] \\
&\le \log\left(\mathbb{E}_{Z_1,\cdots,Z_K\sim q_\phi(z|x)}\left[\frac{1}{K}\sum_{k=1}^{K}\frac{p_\theta(x\,|\,Z_k)p_Z(Z_k)}{q_\phi(Z_k\,|\,x)}\right]\right) \\
&= \log\left(\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}_{Z_k\sim q_\phi(z|x)}\left[\frac{p_\theta(x\,|\,Z_k)p_Z(Z_k)}{q_\phi(Z_k\,|\,x)}\right]\right) \\
&= \log\left(\frac{1}{K}\sum_{k=1}^{K}p_\theta(x)\right) = \log p_\theta(x)
\end{aligned}$$

(b) Using the given hint together with Jensen's inequality, we obtain the following.

$$\begin{aligned}
\mathrm{VLB}^{(K)}_{\theta,\phi}(x) &= \mathbb{E}_{Z_1,\cdots,Z_K\sim q_\phi(z|x)}\left[\log\frac{1}{K}\sum_{k=1}^{K}\frac{p_\theta(x\,|\,Z_k)p_Z(Z_k)}{q_\phi(Z_k\,|\,x)}\right] \\
&= \mathbb{E}_{Z_{i_1},\cdots,Z_{i_M}\sim q_\phi(z|x)}\left[\log\left(\mathbb{E}_{I=\{i_1,\cdots,i_M\}}\left[\frac{1}{M}\sum_{m=1}^{M}\frac{p_\theta(x|Z_{i_m})p_Z(Z_{i_m})}{q_\phi(Z_{i_m}|x)}\right]\right)\right] \\
&\ge \mathbb{E}_{Z_{i_1},\cdots,Z_{i_M}\sim q_\phi(z|x)}\left[\mathbb{E}_{I=\{i_1,\cdots,i_M\}}\left[\log\frac{1}{M}\sum_{m=1}^{M}\frac{p_\theta(x|Z_{i_m})p_Z(Z_{i_m})}{q_\phi(Z_{i_m}|x)}\right]\right] \\
&= \mathbb{E}_{I=\{i_1,\cdots,i_M\}}\left[\mathbb{E}_{Z_{i_1},\cdots,Z_{i_M}\sim q_\phi(z|x)}\left[\log\frac{1}{M}\sum_{m=1}^{M}\frac{p_\theta(x|Z_{i_m})p_Z(Z_{i_m})}{q_\phi(Z_{i_m}|x)}\right]\right] \\
&= \mathbb{E}_{I=\{i_1,\cdots,i_M\}}\left[\mathrm{VLB}^{(M)}_{\theta,\phi}(x)\right] = \mathrm{VLB}^{(M)}_{\theta,\phi}(x)
\end{aligned}$$

(c) We should choose q_ϕ powerful enough so that $q_\phi(Z_k\,|\,x) = p_\theta(Z_k\,|\,x)$ for all $k = 1,\cdots,K$.

$$\begin{aligned}
\mathrm{VLB}^{(K)}_{\theta,\phi}(x) &= \mathbb{E}_{Z_1,\cdots,Z_K\sim q_\phi(z|x)}\left[\log\frac{1}{K}\sum_{k=1}^{K}\frac{p_\theta(x\,|\,Z_k)p_Z(Z_k)}{q_\phi(Z_k\,|\,x)}\right] \\
&= \mathbb{E}_{Z_1,\cdots,Z_K\sim q_\phi(z|x)}\left[\log\frac{1}{K}\sum_{k=1}^{K}\frac{p_\theta(x\,|\,Z_k)p_Z(Z_k)}{p_\theta(Z_k\,|\,x)}\right] \\
&= \mathbb{E}_{Z_1,\cdots,Z_K\sim q_\phi(z|x)}\left[\log\frac{1}{K}\sum_{k=1}^{K}p_\theta(x)\right] = p_\theta(x)
\end{aligned}$$

Problem 2

(a) Since log is a concave function, using Jensen's inequality, we obtain the following.

$$\begin{aligned}\log p_\theta(X_i) &= \log\left(\mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\frac{p_\theta(X_i\,|\,Z)r_\lambda(Z)}{q_\phi(Z\,|\,X_i)}\right]\right)\\ &\geq \mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log\left(\frac{p_\theta(X_i\,|\,Z)r_\lambda(Z)}{q_\phi(Z\,|\,X_i)}\right)\right] = \mathrm{VLB}_{\theta,\phi,\lambda}(X_i)\end{aligned}$$

(b) Gradients regarding θ and λ can be easily derived as the follwing.

$$\nabla_\theta \mathrm{VLB}_{\theta,\phi,\lambda}(X_i) = \nabla_\theta \mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log\left(\frac{p_\theta(X_i\,|\,Z)r_\lambda(Z)}{q_\phi(Z\,|\,X_i)}\right)\right] = \mathbb{E}_{Z\sim q_\phi(z|X_i)}[\nabla_\theta \log p_\theta(X_i\,|\,Z)]$$

$$\nabla_\lambda \mathrm{VLB}_{\theta,\phi,\lambda}(X_i) = \nabla_\lambda \mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log\left(\frac{p_\theta(X_i\,|\,Z)r_\lambda(Z)}{q_\phi(Z\,|\,X_i)}\right)\right] = \mathbb{E}_{Z\sim q_\phi(z|X_i)}[\nabla_\lambda \log r_\lambda(Z)]$$

For gradients on ϕ, we can use the log-derivative trick.

$$\begin{aligned}\nabla_\phi \mathrm{VLB}_{\theta,\phi,\lambda}(X_i) &= \nabla_\phi \mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log\left(\frac{p_\theta(X_i\,|\,Z)r_\lambda(Z)}{q_\phi(Z\,|\,X_i)}\right)\right] = \nabla_\phi \int \log\left(\frac{p_\theta(X_i\,|\,z)r_\lambda(z)}{q_\phi(z\,|\,X_i)}\right) q_\phi(z\,|\,X_i)dz\\ &= \int \left(-\frac{\nabla_\phi q_\phi(z\,|\,X_i)}{q_\phi(z\,|\,X_i)} q_\phi(z\,|\,X_i) + \log\left(\frac{p_\theta(X_i\,|\,z)r_\lambda(z)}{q_\phi(z\,|\,X_i)}\right)\nabla_\phi q_\phi(z\,|\,X_i)\right)dz\\ &= \int \log\left(\frac{p_\theta(X_i\,|\,z)r_\lambda(z)}{q_\phi(z\,|\,X_i)}\right)\frac{\nabla_\phi q_\phi(z\,|\,X_i)}{q_\phi(z\,|\,X_i)} q_\phi(z\,|\,X_i)dz\\ &= \mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log\left(\frac{p_\theta(X_i\,|\,Z)r_\lambda(Z)}{q_\phi(Z\,|\,X_i)}\right)\nabla_\phi \log q_\phi(Z\,|\,X_i)\right]\end{aligned}$$

(c) We can rewrite VLB as the following.

$$\begin{aligned}\mathrm{VLB}_{\theta,\phi,\lambda}(X_i) &= \mathbb{E}_{Z\sim q_\phi(z|X_i)}\left[\log\left(\frac{p_\theta(X_i\,|\,Z)r_\lambda(Z)}{q_\phi(Z\,|\,X_i)}\right)\right]\\ &= \mathbb{E}_{Z\sim q_\phi(z|X_i)}[\log p_\theta(X_i\,|\,Z)] - D_{\mathrm{KL}}(q_\phi(z\,|\,X_i)\,||\,r_\lambda(z))\end{aligned}$$

Then the first term can be calculated as

$$\begin{aligned}\mathbb{E}_{Z\sim q_\phi(z|X_i)}[\log p_\theta(X_i\,|\,Z)] &= \mathbb{E}_{Z\sim\mathcal{N}(\mu_\phi(X_i),\Sigma_\phi(X_i))}\left[\log\mathcal{N}\big(f_\theta(Z),\sigma^2 I\big)\right]\\ &= \mathbb{E}_{Z\sim\mathcal{N}(\mu_\phi(X_i),\Sigma_\phi(X_i))}\Big[-\frac{1}{2}(X_i - f_\theta(Z))^\intercal\big(\sigma^2 I\big)^{-1}(X_i - f_\theta(Z))\\ &\qquad\qquad - \frac{1}{2}\log\Big((2\pi)^k\big|\sigma^2 I\big|\Big)\Big]\\ &= -\frac{1}{2\sigma^2}\mathbb{E}_{Z\sim\mathcal{N}(\mu_\phi(X_i),\Sigma_\phi(X_i))}\left[\|X_i - f_\theta(Z)\|^2\right] - \frac{k}{2}\log\big(2\pi\sigma^2\big)\\ &= -\frac{1}{2\sigma^2}\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}\left[\left\|X_i - f_\theta\left(\mu_\phi(X_i) + \sqrt{\Sigma_\phi(X_i)}\varepsilon\right)\right\|^2\right] - \frac{k}{2}\log\big(2\pi\sigma^2\big).\end{aligned}$$

Using the reparametrization trick simplifies the expectation term, and makes able the gradient of this first term be directly calculated. The second term also can be calculated as the following.

$$
\begin{aligned}
&D_{\mathrm{KL}}(q_\phi(z \mid X_i) \,||\, r_\lambda(z)) \\
&\quad = \frac{1}{2}\left(\mathrm{tr}\Big(\mathrm{diag}(\lambda_2)^{-1}\Sigma_\phi(X_i)\Big) + (\lambda_1 - \mu_\phi(X_i))^\intercal \mathrm{diag}(\lambda_2)^{-1}(\lambda_1 - \mu_\phi(X_i)) - k + \log\left(\frac{\det(\mathrm{diag}(\lambda_2))}{\det(\Sigma_\phi(X_i))}\right)\right)
\end{aligned}
$$

The gradient of the second term can also be directly calculated, so we can obtain the gradients via backpropagation.

Problem 4

(a) Let $p_A = (p_{A1}, p_{A2}, p_{A3})$ and $p_B = (p_{B1}, p_{B2}, p_{B3})$. Then

$$
\mathbb{E}_{p_A, p_B}[\text{points for } B] = p_{A1}p_{B2} + p_{A2}p_{B3} + p_{A3}p_{B1} - p_{A1}p_{B3} - p_{A2}p_{B1} - p_{A3}p_{B2}.
$$

Suppose $p_A^* = (p_{A1}^*, p_{A2}^*, p_{A3}^*)$, $p_B^* = (p_{B1}^*, p_{B2}^*, p_{B3}^*)$ is the solution for the given problem. Then we have

$$
\begin{aligned}
&p_{A1}^*p_{B2} + p_{A2}^*p_{B3} + p_{A3}^*p_{B1} - p_{A1}^*p_{B3} - p_{A2}^*p_{B1} - p_{A3}^*p_{B2} \\
&\qquad \le p_{A1}^*p_{B2}^* + p_{A2}^*p_{B3}^* + p_{A3}^*p_{B1}^* - p_{A1}^*p_{B3}^* - p_{A2}^*p_{B1}^* - p_{A3}^*p_{B2}^* \\
&\qquad\qquad \le p_{A1}p_{B2}^* + p_{A2}p_{B3}^* + p_{A3}p_{B1}^* - p_{A1}p_{B3}^* - p_{A2}p_{B1}^* - p_{A3}p_{B2}^*.
\end{aligned}
$$

for all $p_A, p_B \in \Delta^3$. If $p_A^* = p_B^* = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$, all three terms are 0, so it is a solution of the problem. Now we should show that this is the only solution for the problem. Suppose $p_A^* \neq \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$ and generally let $p_{A1}^* < p_{A2}^*$. Now substitute $p_A = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$ and $p_B = (0, 0, 1)$, then

$$
\begin{aligned}
&p_{A1}^*p_{B2} + p_{A2}^*p_{B3} + p_{A3}^*p_{B1} - p_{A1}^*p_{B3} - p_{A2}^*p_{B1} - p_{A3}^*p_{B2} = p_{A2}^* - p_{A1}^* > 0 \\
&p_{A1}p_{B2}^* + p_{A2}p_{B3}^* + p_{A3}p_{B1}^* - p_{A1}p_{B3}^* - p_{A2}p_{B1}^* - p_{A3}p_{B2}^* = 0
\end{aligned}
$$

so the inequality becomes false. Similarly, suppose $p_B^* \neq \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$ and generally let $p_{B1}^* < p_{B2}^*$. Now substitute $p_A = (0, 0, 1)$ and $p_B = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$, then

$$
\begin{aligned}
&p_{A1}^*p_{B2} + p_{A2}^*p_{B3} + p_{A3}^*p_{B1} - p_{A1}^*p_{B3} - p_{A2}^*p_{B1} - p_{A3}^*p_{B2} = 0 \\
&p_{A1}p_{B2}^* + p_{A2}p_{B3}^* + p_{A3}p_{B1}^* - p_{A1}p_{B3}^* - p_{A2}p_{B1}^* - p_{A3}p_{B2}^* = p_{B1}^* - p_{B2}^* < 0
\end{aligned}
$$

so the inequality also becomes false. Therefore always $p_A^* = p_B^* = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$, so it is the unique solution for the problem.

(b) If B chooses p_B as given, the expected points for B is always 0 regardless of A, so A can choose any strategy. However, if B chooses strategies other than $p_B = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$, choosing any strategy may not be optimal for A. Choosing $p_B = (1, 0, 0), (0, 1, 0), (0, 0, 1)$ results in $\mathbb{E}_{p_A, p_B}[\text{points for } B] > 0$ each when A chooses strategies such that $p_{A3} > p_{A2}$, $p_{A1} > p_{A3}$, $p_{A2} > p_{A1}$.

Problem 3

[1]:
```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import transforms

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
batch_size = 128
```

[2]:
```python
'''
Step 1:
'''
test_val_dataset = datasets.MNIST(root='./mnist_data/',
                                  train=False,
                                  transform=transforms.ToTensor(),
                                  download=True)

test_dataset, validation_dataset = \
        torch.utils.data.random_split(test_val_dataset, [5000, 5000])

# KMNIST dataset, only need test dataset
anomaly_dataset = datasets.KMNIST(root='./kmnist_data/',
                                  train=False,
                                  transform=transforms.ToTensor(),
                                  download=True)
```

[3]:
```python
'''
Step 2:
'''
# Define prior distribution
class Logistic(torch.distributions.Distribution):
    def __init__(self):
        super(Logistic, self).__init__()

    def log_prob(self, x):
        return -(F.softplus(x) + F.softplus(-x))

    def sample(self, size):
        z = torch.distributions.Uniform(0., 1.).sample(size).to(device)
        return torch.log(z) - torch.log(1. - z)

# Implement coupling layer
class Coupling(nn.Module):
    def __init__(self, in_out_dim, mid_dim, hidden, mask_config):
        super(Coupling, self).__init__()
        self.mask_config = mask_config

        self.in_block = \
                nn.Sequential(nn.Linear(in_out_dim//2, mid_dim), nn.ReLU())
        self.mid_block = \
```

```
                nn.ModuleList([nn.Sequential(nn.Linear(mid_dim, mid_dim), nn.ReLU())
                    for _ in range(hidden - 1)])
        self.out_block = nn.Linear(mid_dim, in_out_dim//2)

    def forward(self, x, reverse=False):
        [B, W] = list(x.size())
        x = x.reshape((B, W//2, 2))
        if self.mask_config:
            on, off = x[:, :, 0], x[:, :, 1]
        else:
            off, on = x[:, :, 0], x[:, :, 1]

        off_ = self.in_block(off)
        for i in range(len(self.mid_block)):
            off_ = self.mid_block[i](off_)
        shift = self.out_block(off_)

        if reverse:
            on = on - shift
        else:
            on = on + shift

        if self.mask_config:
            x = torch.stack((on, off), dim=2)
        else:
            x = torch.stack((off, on), dim=2)
        return x.reshape((B, W))

class Scaling(nn.Module):
    def __init__(self, dim):
        super(Scaling, self).__init__()
        self.scale = nn.Parameter(torch.zeros((1, dim)))

    def forward(self, x, reverse=False):
        log_det_J = torch.sum(self.scale)
        if reverse:
            x = x * torch.exp(-self.scale)
        else:
            x = x * torch.exp(self.scale)
        return x, log_det_J

class NICE(nn.Module):
    def __init__(self,in_out_dim, mid_dim, hidden,
                mask_config=1.0, coupling=4):
        super(NICE, self).__init__()
        self.prior = Logistic()
        self.in_out_dim = in_out_dim

        self.coupling = nn.ModuleList([
            Coupling(in_out_dim=in_out_dim,
                     mid_dim=mid_dim,
                     hidden=hidden,
                     mask_config=(mask_config+i)%2) \
```

```
            for i in range(coupling)])

        self.scaling = Scaling(in_out_dim)

    def g(self, z):
        x, _ = self.scaling(z, reverse=True)
        for i in reversed(range(len(self.coupling))):
            x = self.coupling[i](x, reverse=True)
        return x

    def f(self, x):
        for i in range(len(self.coupling)):
            x = self.coupling[i](x)
        z, log_det_J = self.scaling(x)
        return z, log_det_J

    def log_prob(self, x):
        z, log_det_J = self.f(x)
        log_ll = torch.sum(self.prior.log_prob(z), dim=1)
        return log_ll + log_det_J

    def sample(self, size):
        z = self.prior.sample((size, self.in_out_dim)).to(device)
        return self.g(z)

    def forward(self, x):
        return self.log_prob(x)
```

[4]:
```
'''
Step 3: Load the pretrained model
'''
nice = NICE(in_out_dim=784, mid_dim=1000, hidden=5).to(device)
nice.load_state_dict(torch.load('nice.pt', map_location=device))
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/torch/distributions/distribution.py:44: UserWarning: <class
'__main__.Logistic'> does not define `arg_constraints`. Please set
`arg_constraints = {}` or initialize the distribution with `validate_args=False`
to turn off validation.
  warnings.warn(f'{self.__class__} does not define `arg_constraints`. ' +
```

[4]: <All keys matched successfully>

[5]:
```
'''
Step 4: Calculate standard deviation by using validation set
'''
validation_loader = torch.utils.data.DataLoader(
        dataset=validation_dataset, batch_size=batch_size)
likelihood_list = []
for batch, (images, _) in enumerate(validation_loader):
    likelihood_list += nice(images.view(-1, 784).to(device)).tolist()

# calculate standard deviation
```

```python
import statistics
std = statistics.stdev(likelihood_list)
mean = statistics.mean(likelihood_list)
threshold = mean - 3*std
```

[6]:
```python
'''
Step 5: Anomaly detection (mnist)
'''
test_loader = torch.utils.data.DataLoader(
        dataset=test_dataset, batch_size=batch_size)
count = 0
for batch, (images, _) in enumerate(test_loader):
    likelihood = nice(images.view(-1, 784).to(device))
    count += torch.sum(likelihood < threshold).item()

print(f'{count} type I errors among {len(test_dataset)} data')
```

```
103 type I errors among 5000 data
```

[7]:
```python
'''
Step 6: Anomaly detection (kmnist)
'''
anomaly_loader = torch.utils.data.DataLoader(
        dataset=anomaly_dataset, batch_size=batch_size)
count = 0
for batch, (images, _) in enumerate(anomaly_loader):
    likelihood = nice(images.view(-1, 784).to(device))
    count += torch.sum(likelihood > threshold).item()

print(f'{count} type II error samong {len(anomaly_dataset)} data')
```

```
15 type II error samong 10000 data
```