hw8

November 29, 2021

1 Mathematical Foundations on Deep Neural Network

1.1 Homework #8

1.1.1 2017-11362

1.2 Problem 1.

[1]:
```
import torch
import torch.utils.data as data
import torch.nn as nn
from torch.distributions.normal import Normal
from torch.distributions.uniform import Uniform
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

epochs = 500
learning_rate = 5e-3
batch_size = 128
n_components = 5 # the number of kernel
target_distribution = Normal(0.0, 1.0)
```

[2]:
```
######################################
# STEP 1: Implement 1-d Flow model   #
# Model is mixture of Gaussian CDFs #
######################################

class Flow1d(nn.Module):
    def __init__(self, n_components):
        super(Flow1d, self).__init__()
        self.mus = nn.Parameter(torch.randn(n_components), requires_grad=True)
        self.log_sigmas = nn.Parameter(torch.zeros(n_components),␣
↪requires_grad=True)
        self.weight_logs = nn.Parameter(torch.zeros(n_components),␣
↪requires_grad=True)

    def forward(self, x):
        x = x.view(-1,1)
```

```python
        weights = self.weight_logs.exp().view(1,-1)
        distribution = Normal(self.mus, self.log_sigmas.exp())
        z = (weights * (distribution.cdf(x) - 0.5)).sum(dim=1)
        dz_by_dx = (distribution.log_prob(x).exp() * weights).sum(dim=1)
        return z, dz_by_dx
```

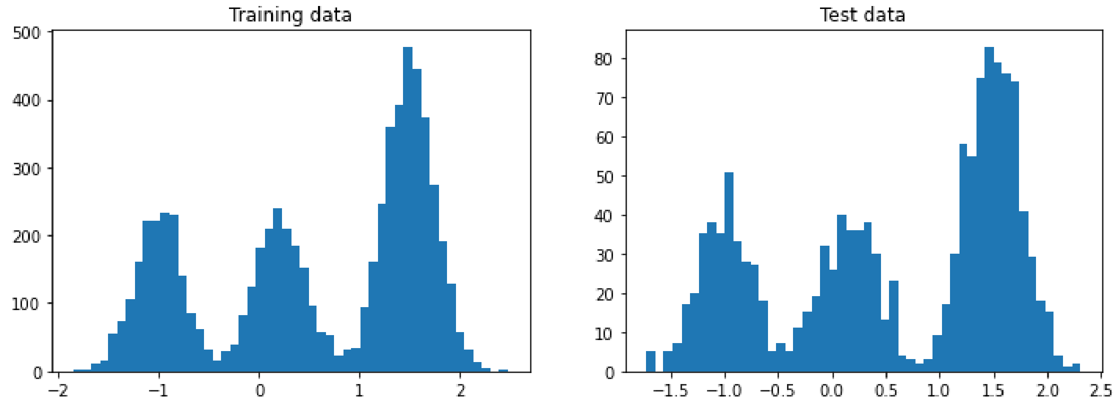
[3]:
```python
################################################
# STEP 2: Create Dataset and Create Dataloader #
################################################

def mixture_of_gaussians(num, mu_var=(-1,0.25, 0.2,0.25, 1.5,0.25)):
    n = num // 3
    m1,s1,m2,s2,m3,s3 = mu_var
    gaussian1 = np.random.normal(loc=m1, scale=s1, size=(n,))
    gaussian2 = np.random.normal(loc=m2, scale=s2, size=(n,))
    gaussian3 = np.random.normal(loc=m3, scale=s3, size=(num-n,))
    return np.concatenate([gaussian1, gaussian2, gaussian3])

class MyDataset(data.Dataset):
    def __init__(self, array):
        super().__init__()
        self.array = array

    def __len__(self):
        return len(self.array)

    def __getitem__(self, index):
        return self.array[index]
```

[4]:
```python
################################
# STEP 3: Define Loss Function #
################################

def loss_function(target_distribution, z, dz_by_dx):
    # log(p_Z(z)) = target_distribution.log_prob(z)
    # log(dz/dx) = dz_by_dx.log() (flow is defined so that dz/dx>0)
    log_likelihood = target_distribution.log_prob(z) + dz_by_dx.log()
    return -log_likelihood.mean()  #flip sign, and sum of data X_1,...X_N
```

[5]:
```python
###########################
# STEP 4: Train the model #
###########################

# create dataloader
n_train, n_test = 5000, 1000
train_data = mixture_of_gaussians(n_train)
test_data = mixture_of_gaussians(n_test)
```

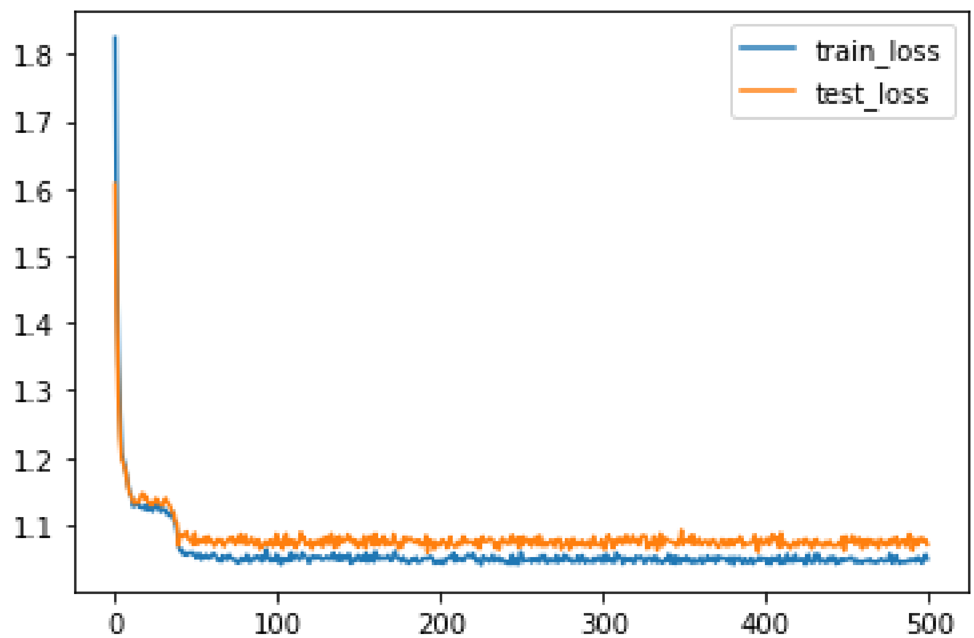
```
train_loader = data.DataLoader(MyDataset(train_data), batch_size=batch_size,␣
 ↪shuffle=True)
test_loader = data.DataLoader(MyDataset(test_data), batch_size=batch_size,␣
 ↪shuffle=True)

# create model
flow = Flow1d(n_components)
optimizer = torch.optim.Adam(flow.parameters(), lr=learning_rate)

train_losses, test_losses = [], []

for epoch in range(epochs):
    # train
    # flow.train()
    mean_loss = 0
    for i, x in enumerate(train_loader):
        z, dz_by_dx = flow(x)
        loss = loss_function(target_distribution, z, dz_by_dx)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        mean_loss += loss.item()
    train_losses.append(mean_loss/(i+1))

    # test
    flow.eval()
    mean_loss = 0
    for i, x in enumerate(test_loader):
        z, dz_by_dx = flow(x)
        loss = loss_function(target_distribution, z, dz_by_dx)

        mean_loss += loss.item()
    test_losses.append(mean_loss/(i+1))
```

[6]:
```
_, axes = plt.subplots(1,2, figsize=(12,4))
_ = axes[0].hist(train_loader.dataset.array, bins=50)
_ = axes[1].hist(test_loader.dataset.array, bins=50)
_ = axes[0].set_title('Training data')
_ = axes[1].set_title('Test data')
```



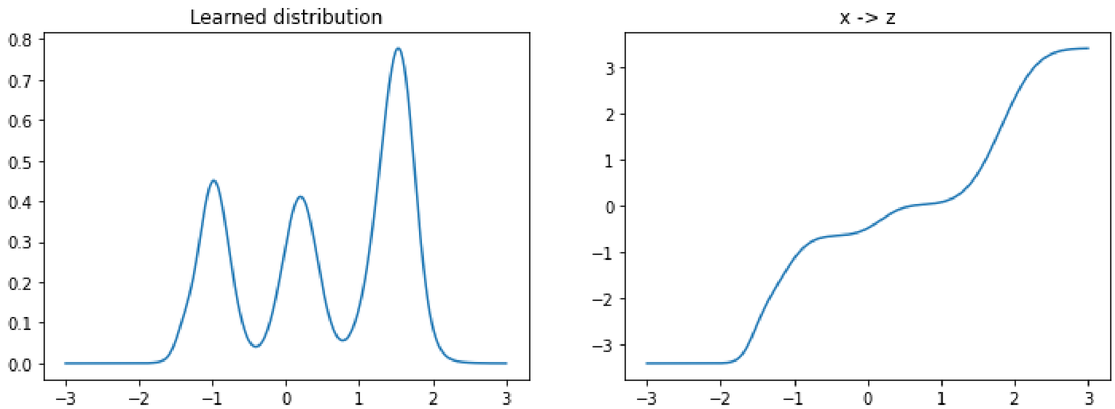

```
[7]: plt.plot(train_losses, label='train_loss')
     plt.plot(test_losses, label='test_loss')
     plt.legend()
```

```
[7]: <matplotlib.legend.Legend at 0x7f2e35276d30>
```

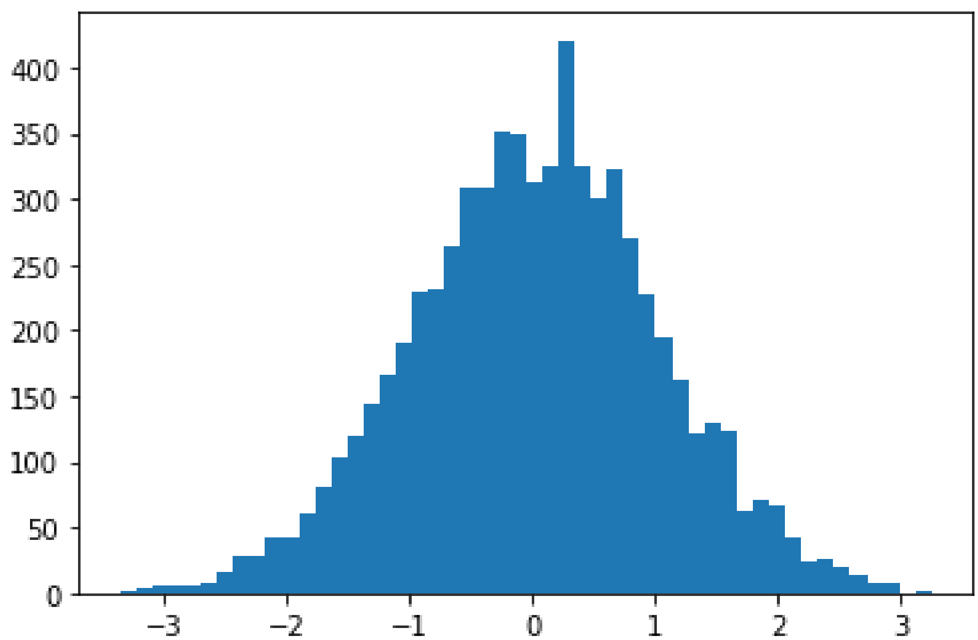



```
[8]: x = np.linspace(-3,3,1000)
     with torch.no_grad():
         z, dz_by_dx = flow(torch.FloatTensor(x))
         px = (target_distribution.log_prob(z) + dz_by_dx.log()).exp().cpu().numpy()
```

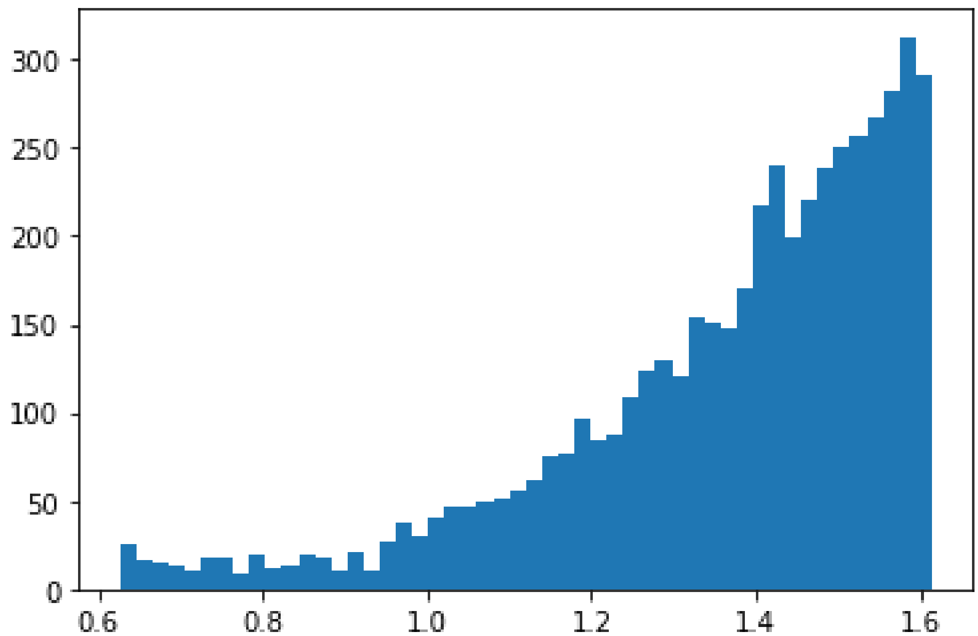
```
_, axes = plt.subplots(1,2, figsize=(12,4))
_ = axes[0].plot(x,px)
_ = axes[0].set_title('Learned distribution')

_ = axes[1].plot(x,z)
_ = axes[1].set_title('x -> z')
```



[9]:
```
with torch.no_grad():
    z, _ = flow(torch.FloatTensor(train_loader.dataset.array))

_ = plt.hist(np.array(z), bins=50)
```



[10]:
```python
# sampling
N = 5000
z = torch.rand(N)
x_low = torch.full((N,), -3.)
x_high = torch.full((N,), 3.)

#Perform bisection
with torch.no_grad():
    for _ in range(30):
        m = (x_low+x_high)/2
        f,_ = flow(m)
        x_high[f>=z] = m[f>=z]
        x_low[f<z] = m[f<z]
    x = (x_low+x_high)/2

_ = plt.hist(np.array(x), bins=50)
```



1.3 Problem 2.

$f(\theta) = g(\theta, \phi) + h(\theta, \phi)$, $\quad h(\theta,\phi) \geq 0$, $\quad \min_{\phi} h(\theta, \phi) = 0$, $\forall \theta \in \Theta$

$\max_{\theta,\phi} g(\theta, \phi) = \max_{\theta,\phi} \{f(\theta) - h(\theta,\phi)\} = \max_{\theta} [\max_{\phi} \{f(\theta) - h(\theta,\phi)\}] = \max_{\theta} \{f(\theta) - \min_{\phi} h(\theta,\phi)\} = \max_{\theta} f(\theta).$

Let $\operatorname{argmax} g(\theta, \phi) = (\theta^*, \phi^*)$. Then $\phi^* = \operatorname{argmin}_{\phi} h(\theta^*, \phi)$, that is, $h(\theta^*, \phi^*) = 0$.

$\therefore f(\theta^*) = g(\theta^*, \phi^*) + h(\theta^*, \phi^*) = g(\theta^*, \phi^*) \geq g(\theta, \phi_\theta)$ $\quad (\exists \phi_\theta \in \Phi \text{ s.t } h(\theta, \phi_\theta) = 0)$

$= g(\theta, \phi_\theta) + h(\theta, \phi_\theta) = f(\theta)$, $\forall \theta \in \Theta$.

$\therefore \operatorname{argmax}_{\theta} f(\theta) = \theta^*$.

1.4 Problem 3.

[11]:
```python
def inv_perm(perm):
    ans = [0] * len(perm)
    for (n, i) in enumerate(perm):
        ans[i - 1] = n + 1
    return ans

perm = [1,4,6,5,3,2]
print(inv_perm(perm))
for i in range(1, 7):
    print(f'p({i}) = {perm[i-1]}, p^-1(p({i})) = p^-1({perm[i-1]}) =␣
 ↪{inv_perm(perm)[perm[i-1]-1]}')
```

```
[1, 6, 5, 2, 4, 3]
p(1) = 1, p^-1(p(1)) = p^-1(1) = 1
p(2) = 4, p^-1(p(2)) = p^-1(4) = 2
p(3) = 6, p^-1(p(3)) = p^-1(6) = 3
p(4) = 5, p^-1(p(4)) = p^-1(5) = 4
p(5) = 3, p^-1(p(5)) = p^-1(3) = 5
p(6) = 2, p^-1(p(6)) = p^-1(2) = 6
```

1.5 Problem 4.

$P_\sigma = [e_{\sigma(1)} \cdots e_{\sigma(n)}]^T \in \mathbb{R}^{n\times n}$

(a) ① $P_\sigma^T = P_\sigma^{-1} \iff P_\sigma^T P_\sigma = I.$

$$P_\sigma^T P_\sigma = [e_{\sigma(1)} \cdots e_{\sigma(n)}] \cdot \begin{bmatrix} e_{\sigma(1)}^T \\ \vdots \\ e_{\sigma(n)}^T \end{bmatrix} = \sum_{i=1}^{n} e_{\sigma(i)} e_{\sigma(i)}^T$$

$$(e_{\sigma(i)} e_{\sigma(i)}^T)_{kj} = (e_{\sigma(i)})_k \cdot (e_{\sigma(i)})_j = \begin{cases} 1 & , \ k=j=\sigma(i) \\ 0 & , \ \text{o.w} \end{cases}$$

$\because$ only $(\sigma_{(i)}, \sigma_{(i)})$th entry is $1.$

$$\therefore P_\sigma^T P_\sigma = \sum_{i=1}^{n} e_{\sigma(i)} e_{\sigma(i)}^T = \sum_{i=1}^{n} e_i e_i^T = I_n.$$

② $P_\sigma^{-1} = P_{\sigma^{-1}} \iff P_\sigma \cdot P_{\sigma^{-1}} = I.$

$$(P_\sigma \cdot P_{\sigma^{-1}})_{ij} = \sum_{k=1}^{n} (P_\sigma)_{ik} \cdot (P_{\sigma^{-1}})_{kj} = \sum_{k=1}^{n} (e_{\sigma(i)})_k \, (e_{\sigma^{-1}(k)})_j$$

$$(e_{\sigma(i)})_k \, (e_{\sigma^{-1}(k)})_j = \begin{cases} (e_{\sigma^{-1}(\sigma(i))})_j & , \ k=\sigma(i) \\ 0 & , \ \text{o.w} \end{cases} = \begin{cases} (e_i)_j & , \ k=\sigma(i) \\ 0 & , \ \text{o.w} \end{cases} = \begin{cases} 1 & , \ i=j \text{ and } k=\sigma(i) \\ 0 & , \ \text{o.w} \end{cases}$$

$$\therefore (P_\sigma \cdot P_{\sigma^{-1}})_{ij} = \begin{cases} 1 & , \ i=j \\ 0 & , \ \text{o.w} \end{cases} = I_n.$$

(b) $$|\det P_\sigma| = \sqrt{(\det P_\sigma)^2} = \sqrt{\det P_\sigma \cdot \det P_\sigma^T} = \sqrt{\det P_\sigma \cdot P_\sigma^T} = \sqrt{\det I_n} = 1.$$

1.6 Problem 5.

Suppose $\Omega = \{\Omega_1, \cdots, \Omega_k\}$, $0<k<n.$

Let $\sigma : \{1,\cdots,n\} \to \{1,\cdots,n\}$ be a permutation s.t $\sigma(i) = \Omega_i$, $\forall_{1\le i\le k}.$ $(P_\sigma \cdot x = [x_\Omega, x_{\Omega^c}],\ P_\sigma \cdot z = [z_\Omega, z_{\Omega^c}])$

$$z_\Omega = x_\Omega \Rightarrow \frac{\partial z_\Omega}{\partial x_\Omega} = I_k, \quad \frac{\partial z_\Omega}{\partial z_{\Omega^c}} = 0.$$

$$z_{\Omega^c} = \exp(s_\theta(x_\Omega)) \odot x_{\Omega^c} + t_\theta(x_\Omega) \Rightarrow \frac{\partial z_{\Omega^c}}{\partial x_{\Omega^c}} = \mathrm{diag}(e^{s_\theta(x_\Omega)})$$

$$\therefore \frac{\partial z}{\partial x} = \frac{\partial z}{\partial [z_\Omega, z_{\Omega^c}]} \frac{\partial [z_\Omega, z_{\Omega^c}]}{\partial [x_\Omega, x_{\Omega^c}]} \frac{\partial [x_\Omega, x_{\Omega^c}]}{\partial x} = \left(\frac{\partial P_\sigma z}{\partial z}\right)^{-1} \cdot \begin{bmatrix} \frac{\partial z_\Omega}{\partial x_\Omega} & \frac{\partial z_\Omega}{\partial x_{\Omega^c}} \\ \frac{\partial z_{\Omega^c}}{\partial x_\Omega} & \frac{\partial z_{\Omega^c}}{\partial z_\Omega} \end{bmatrix} \cdot \frac{\partial P_\sigma x}{\partial x} = P_{\sigma^{-1}} \begin{bmatrix} I_k & 0 \\ * & \mathrm{diag}(e^{s_\theta(x_\Omega)}) \end{bmatrix} P_\sigma$$

$$\left|\frac{\partial z}{\partial x}\right| = \overbrace{\det P_{\sigma^{-1}} \cdot \det P_\sigma}^{1} \cdot \prod e^{s_\theta(x_\Omega)_i}$$

$$\therefore \log\left|\frac{\partial z}{\partial x}\right| = \sum_i e^{s_\theta(x_\Omega)_i} = 1_{(n-|\Omega|)}^T e^{s_\theta(x_\Omega)}$$ $(\because s_\theta : \mathbb{R}^k \to \mathbb{R}^{n-k})$

1.7 Problem 6.

[12]:
```python
N = 3000
K = 600
p, q = 18/37, 0.55
def B_samp(n, p):
    return (torch.rand(n) < p).numpy()
```

```python
def f_over_g(x, p, q):
    x_sum = np.sum(x)
    return np.exp(x_sum * np.log(p/q) + (len(x) - x_sum) * np.log((1-p)/(1-q)))

def win(x):
    cumsum = 100
    for i in range(len(x)):
        cumsum += 2 * x[i] - 1
        if cumsum <= 0:
            return 0, i + 1
        elif cumsum >= 200:
            return 1, i + 1
    return 0, i + 1

total = 0
for _ in range(N):
    x = B_samp(K, q)
    prob, ind = win(x)
    imp_samp = prob * f_over_g(x, p, q)
    total += imp_samp

print(total / N)
```

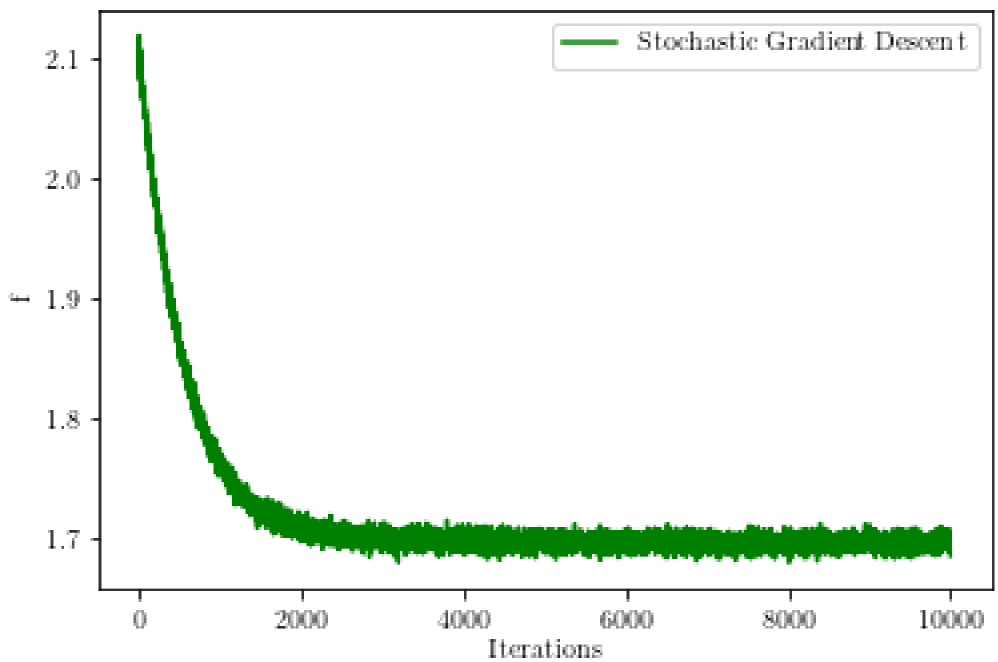
```
2.519068918962955e-06
```

1.8 Problem 7.

(a) the log-derivative trick

[13]:
```python
alpha = 5e-4
theta = np.zeros(2) # (mu, tau), exp(tau) = sigma
def f(theta):
    x = np.exp(theta[1]) * np.random.randn(10000) + theta[0]
    E = np.sum(x * np.sin(x)) / 10000
    return E + 0.5 * (theta[0] - 1) ** 2 + np.exp(theta[1]) - theta[1]

def grad(theta):
    mu, sigma = theta[0], np.exp(theta[1])
    x = sigma * np.random.randn(10000) + mu
    grad_mu = np.sum((x - mu) / sigma ** 2 * x * np.sin(x)) / 10000 + mu - 1
    grad_tau = np.sum(((x - mu) ** 2 / sigma ** 2 - 1) * x * np.sin(x)) / 10000␣
 ↪+ sigma - 1
    return np.array([grad_mu, grad_tau])

f_val = []
K = 10000
for _ in range(K):
```

```python
    theta = theta - alpha * grad(theta)
    f_val.append(f(theta))

plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.plot(list(range(K)),f_val, color = "green", label = "Stochastic Gradient␣
↪Descent")
plt.xlabel('Iterations')
plt.ylabel('f')
plt.legend()
plt.show()
print(f'mu: {theta[0]}, sigma: {np.exp(theta[1])}')
```

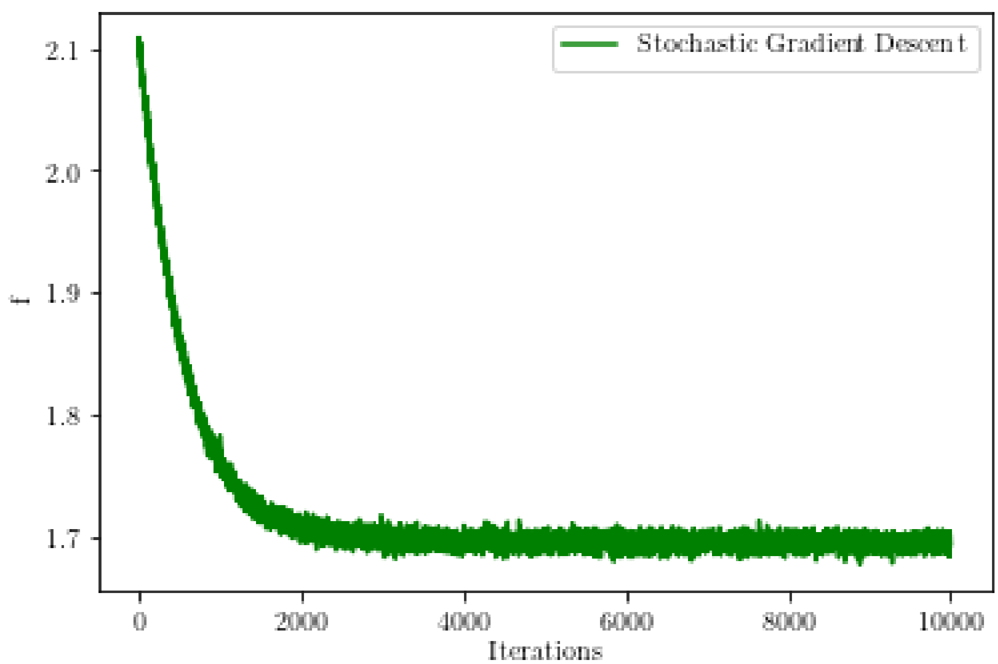



```
mu: 0.44403578860227777, sigma: 0.6089689065449186
```

(b) the reparameterization trick

```python
[14]: alpha = 5e-4
theta = np.zeros(2) # (mu, tau), exp(tau) = sigma
def f(theta):
    x = np.exp(theta[1]) * np.random.randn(10000) + theta[0]
    E = np.sum(x * np.sin(x)) / 10000
    return E + 0.5 * (theta[0] - 1) ** 2 + np.exp(theta[1]) - theta[1]

def grad(theta):
```

```
    mu, sigma = theta[0], np.exp(theta[1])
    eps = np.random.randn(10000)
    x = sigma * eps + mu
    grad_mu = np.sum(np.sin(x) + x * np.cos(x)) / 10000 + mu - 1
    grad_tau = np.sum(sigma * eps * (np.sin(x) + x * np.cos(x))) / 10000 +␣
 ↪sigma - 1
    return np.array([grad_mu, grad_tau])

f_val = []
K = 10000
for _ in range(K):
    theta = theta - alpha * grad(theta)
    f_val.append(f(theta))

plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.plot(list(range(K)),f_val, color = "green", label = "Stochastic Gradient␣
 ↪Descent")
plt.xlabel('Iterations')
plt.ylabel('f')
plt.legend()
plt.show()
print(f'mu: {theta[0]}, sigma: {np.exp(theta[1])}')
```




```
mu: 0.44390201993186706, sigma: 0.6087760410307997
```

1.9 Problem 8.

$$\nabla_\phi \mathbb{E}_{z\sim q_\phi(z)}\left[\log\left(\frac{h(z)}{q_\phi(z)}\right)\right] = \nabla_\phi \int \log\left(\frac{h(z)}{q_\phi(z)}\right) q_\phi(z)\, dz = \int \nabla_\phi \left[\log\left(\frac{h(z)}{q_\phi(z)}\right) q_\phi(z)\right] dz = \int \nabla_\phi \left[\log(h(z))\, q_\phi(z) - \log(q_\phi(z))\, q_\phi(z)\right] dz$$

$$= \int \log(h(z))\, \nabla_\phi q_\phi(z) - \frac{\nabla_\phi q_\phi(z)}{q_\phi(z)} q_\phi(z) - \log(q_\phi(z))\, \nabla_\phi q_\phi(z)\, dz$$

$$= \int \log\left(\frac{h(z)}{q_\phi(z)}\right) \nabla_\phi q_\phi(z)\, dz - \underbrace{\int \nabla_\phi q_\phi(z)\, dz}_{=0}$$

$$= \int \log\left(\frac{h(z)}{q_\phi(z)}\right) \cdot \underbrace{\frac{\nabla_\phi q_\phi(z)}{q_\phi(z)}}_{= \nabla_\phi \log q_\phi(z).} \cdot q_\phi(z)\, dz = \mathbb{E}_{z\sim q_\phi(z)}\left[\left(\nabla_\phi \log q_\phi(z)\right) \log\left(\frac{h(z)}{q_\phi(z)}\right)\right]$$