STA414: Statistical Methods for Machine Learning II

Homework 3

Tianyu (Shin) Ren

1002379188

University of Toronto

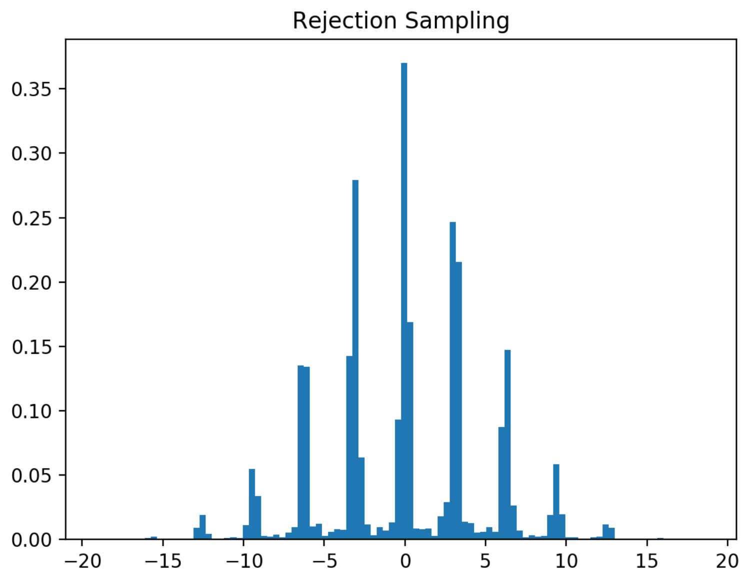
April 2019

1. Nature's rejection sampler

(a) **Estimate $\int p(x, g = 0|\theta = 0)dx$ by summing over 10,000 values of x, equally spaced from -20 to 20**

The estimate of $p(g = 0|\theta = 0)$: 0.8

(b) **Implement a rejection sampler to sample from $p(x|g = 1, \theta = 0)$**

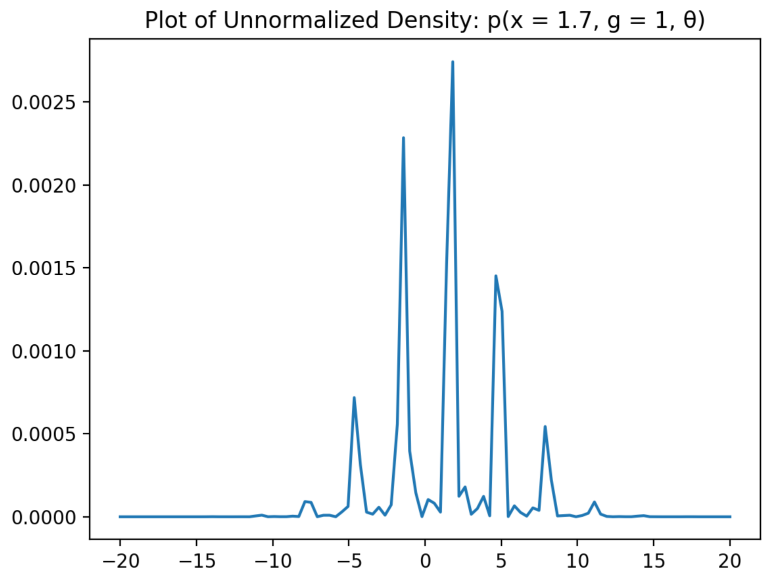


The fraction of accepted samples: 0.2

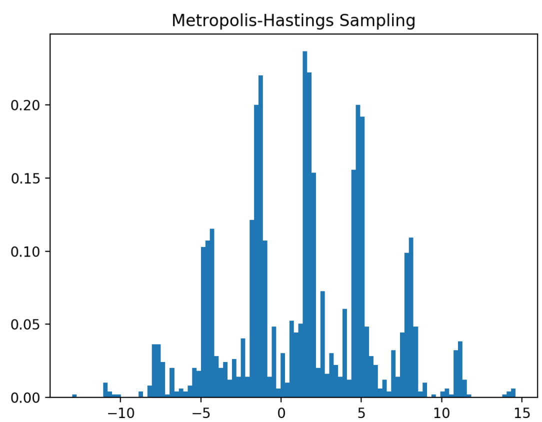
(c) **Implement a self-normalized importance sampler to estimate $p(g = 0|\theta = 0)$. Use $p(x|\theta = 0)$ as a proposal distribution.**

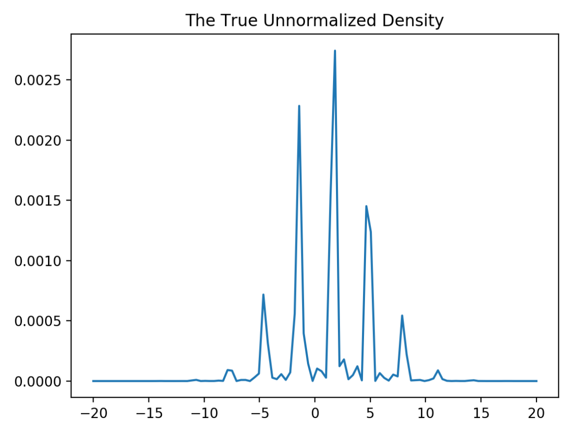
The estimate of the fraction of photons that are absorbed: 0.79

(d) **Plot the unnormalized density $p(x = 1.7, g = 1, \theta)$, as a function of θ**



(e) **Write a Metropolis-Hastings sampler to sample from $p(\theta|x, a = 1)$. For the proposal distribution, use a Gaussian centered around the current sample**





(f) **Use samples from your MH chain to estimate the posterior probability $p(-3 < \theta < 3|x = 1.7, g = 1)$**

Estimate of the posterior probability: 0.47

2. Gradient estimators

(a) **Prove** $\mathbb{E}_{p(x|\theta)}\left[\nabla_\theta \log p(x|\theta)\right] = 0$

$$\begin{aligned}
\mathbb{E}_{p(x|\theta)}\left[\nabla_\theta \log p(x|\theta)\right] &= \int \nabla_\theta \log p(x|\theta) p(x|\theta) dx \\
&= \int \frac{\nabla_\theta p(x|\theta)}{p(x|\theta)} p(x|\theta) dx \\
&= \int \nabla_\theta p(x|\theta) dx \\
&= \nabla_\theta \int p(x|\theta) dx \\
&= \nabla_\theta 1 \\
&= 0
\end{aligned}$$

(b) **Show that** $\mathbb{E}_{p(b|\theta)}\left[f(b)\frac{\partial}{\partial\theta}\log p(b|\theta)\right] = \frac{\partial}{\partial\theta}\mathbb{E}_{p(b|\theta)}\left[f(b)\right]$

$$\begin{aligned}
\mathbb{E}_{p(b|\theta)}\left[f(b)\frac{\partial}{\partial\theta}\log p(b|\theta)\right] &= \int f(b)\frac{\partial}{\partial\theta}\log p(b|\theta)\cdot p(b|\theta) db \\
&= \int f(b)\frac{\frac{\partial}{\partial\theta}p(b|\theta)}{p(b|\theta)} p(b|\theta) db \\
&= \int f(b)\frac{\partial}{\partial\theta}p(b|\theta) db \\
&= \frac{\partial}{\partial\theta}\int f(b) p(b|\theta) db \\
&= \frac{\partial}{\partial\theta}\mathbb{E}_{p(b|\theta)}\left[f(b)\right]
\end{aligned}$$

(c) **Show that** $\mathbb{E}_{p(b|\theta)}\left[[f(b)-c]\frac{\partial}{\partial\theta}\log p(b|\theta)\right] = \frac{\partial}{\partial\theta}\mathbb{E}_{p(b|\theta)}\left[f(b)\right]$**, for any fixed** c

$$\begin{aligned}
\mathbb{E}_{p(b|\theta)}\left[[f(b)-c]\frac{\partial}{\partial\theta}\log p(b|\theta)\right] &= \int [f(b)-c]\frac{\partial}{\partial\theta}\log p(b|\theta)\cdot p(b|\theta) db \\
&= \int f(b)\frac{\partial}{\partial\theta}p(b|\theta) db - c\int \frac{\partial}{\partial\theta}p(b|\theta) db \\
&= \frac{\partial}{\partial\theta}\mathbb{E}_{p(b|\theta)}\left[f(b)\right] - c\frac{\partial}{\partial\theta}\int p(b|\theta) db \\
&= \frac{\partial}{\partial\theta}\mathbb{E}_{p(b|\theta)}\left[f(b)\right] - c\frac{\partial}{\partial\theta}1 \\
&= \frac{\partial}{\partial\theta}\mathbb{E}_{p(b|\theta)}\left[f(b)\right]
\end{aligned}$$

(d) **Give an example where** $\mathbb{E}_{p(b|\theta)}\left[[f(b)-c(b)]\frac{\partial}{\partial\theta}\log p(b|\theta)\right] \neq \frac{\partial}{\partial\theta}\mathbb{E}_{p(b|\theta)}\left[f(b)\right]$ **for some function** $c(b)$**, and show that it is biased.**

$$\text{Let } c(b) = f(b) \implies \mathbb{E}_{p(b|\theta)}\left[[f(b)-c(b)]\frac{\partial}{\partial\theta}\log p(b|\theta)\right] = 0$$

$$\text{But } \frac{\partial}{\partial\theta}\mathbb{E}_{p(b|\theta)}\left[f(b)\right] \neq 0 \text{ in general. So, it is biased.}$$

3. Comparing variances of gradient estimators

(a) **Compute $\mathbb{V}\left[\hat{L}_{MC}\right]$ as a function of D**

$$\mathbb{V}\left[\hat{L}_{MC}\right] = \mathbb{V}\left[\sum_{d=1}^{D} x_d\right] = \sum_{d=1}^{D}\mathbb{V}\left[x_d\right] = D\cdot\mathbb{V}\left[x_d\right] = D$$

(b) **Derive a closed form for $\hat{g}^{\mathrm{SF}}[f] = [f(x) - c(\theta)]\,\frac{\partial}{\partial\theta}\log p(x|\theta)$**

$$\begin{aligned}
\hat{g}^{\mathrm{SF}}[f] &= \left[\sum_{d=1}^{D} x_d - \sum_{d=1}^{D}\theta_d\right]\frac{\partial}{\partial\theta}[\log(\frac{1}{\sqrt[D]{2\pi}}) - \frac{1}{2}(x-\theta)^T(x-\theta)]\\
&= \left[\sum_{d=1}^{D}(x_d-\theta_d)\right]\frac{\partial}{\partial\theta}[\log(\frac{1}{\sqrt[D]{2\pi}}) - \frac{1}{2}(x^Tx - 2\theta^Tx + \theta^T\theta)]\\
&= \left[\sum_{d=1}^{D}(x_d-\theta_d)\right](x-\theta)\\
&= (\mathbb{1}^T\epsilon)\epsilon
\end{aligned}$$

$$\text{where } \epsilon = x - \theta = \begin{pmatrix} x_1-\theta_1\\ \vdots \\ x_d-\theta_d\end{pmatrix}, \quad \mathbb{1} = \begin{pmatrix}1\\ \vdots\\ 1\end{pmatrix}, \quad \epsilon\sim\mathcal{N}(0,I)$$

(c) **Derive the scalar value $\mathbb{V}\left[\hat{g}_1^{\mathrm{SF}}\right]$ as a function of D**
By Hint: $\mathbb{E}(\epsilon^3) = 0$, $\mathbb{E}(\epsilon^4) = 3$, $\mathbb{E}(\epsilon^2) = \mathbb{V}(\epsilon) + \mathbb{E}(\epsilon) = 1$, $\mathbb{E}(\epsilon_i\epsilon_j) = COV(\epsilon_i,\epsilon_j) + \mathbb{E}(\epsilon_i)\mathbb{E}(\epsilon_j) = 0$

$$\begin{aligned}
\mathbb{V}\left[\hat{g}_1^{\mathrm{SF}}\right] &= \mathbb{V}((\mathbb{1}^T\epsilon)\epsilon_1)\\
&= \mathbb{E}([(\mathbb{1}^T\epsilon)\epsilon_1]^2) - \mathbb{E}((\mathbb{1}^T\epsilon)\epsilon_1)^2\\
&= \mathbb{E}(\epsilon_1^2(\sum_d\epsilon_d^2 + 2\sum_i\sum_j\epsilon_i\epsilon_j)) - \mathbb{E}((\mathbb{1}^T\epsilon)\epsilon_1)^2\\
&= \mathbb{E}(\epsilon_1^2\sum_d\epsilon_d^2) + 2\sum_i\sum_j\mathbb{E}(\epsilon_i\epsilon_j) - \mathbb{E}((\mathbb{1}^T\epsilon)\epsilon_1)^2\\
&= \mathbb{E}(\epsilon_1^4 + \epsilon_1^2\epsilon_2^2 + \cdots + \epsilon_1^2\epsilon_D^2) - \mathbb{E}(\epsilon_1^2 + \epsilon_1\epsilon_2 + \cdots + \epsilon_1\epsilon_D)^2\\
&= \mathbb{E}(\epsilon_1^4) + \mathbb{E}(\epsilon_1^2)\mathbb{E}(\epsilon_2^2 + \cdots + \epsilon_D^2) - [\mathbb{E}(\epsilon_1^2) + \mathbb{E}(\epsilon_1)\mathbb{E}(\epsilon_2 + \cdots + \epsilon_D)]^2\\
&= 3 + 1\cdot(D-1) - 1^2\\
&= D + 1
\end{aligned}$$

(d) **Derive this gradient estimator for $\nabla_\theta L(\theta)$, and give $\mathbb{V}\left[\hat{g}_1^{\text{REPARAM}}\right]$ as a function of D**

$$
\begin{aligned}
\nabla_\theta L(\theta) &= \nabla_\theta \mathbb{E}_{p(b|\theta)}[f(x)] \\
&= \nabla_\theta \mathbb{E}_{p(\epsilon)}[\sum_{d=1}^{D}(\theta_d + \epsilon_d)] \\
&= \mathbb{E}_{p(\epsilon)}[\nabla_\theta \sum_{d=1}^{D}(\theta_d + \epsilon_d)] \\
&= \mathbb{E}_{p(\epsilon)} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \\
&= (1, \cdots, 1)^T
\end{aligned}
$$

$$
\mathbb{V}\left[\hat{g}_1^{\text{REPARAM}}\right] = \mathbb{V}[\frac{\partial f}{\partial x}\frac{\partial x}{\partial \theta}] = \mathbb{V}[\frac{\partial f}{\partial x}(1)] = \mathbb{V}[(1, \cdots, 1)^T] = 0
$$

Appendix

Python Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
np.random.seed(414)


# numerical integration
def p_x_given_theta(x, theta):
  return norm.pdf(x, loc=theta, scale=4)

def f(x, theta):
  return np.sin(5 * (x - theta)) ** 2 / (25 * np.sin(x - theta) ** 2)

def p_g_given_x_theta(x, theta, g=1):
  return f(x, theta) ** g * (1 - f(x, theta)) ** (1 - g)

def numerical_integration(g, theta):
  x = np.linspace(-20,20,10000)
  prod = p_g_given_x_theta(x, theta, g) * p_x_given_theta(x, theta)
  area = np.sum(prod) * 40 / len(x)
  # f1 = plt.figure()
  # ax1 = f1.add_subplot(111)
  # plt.plot(x, prod)
  return area

# rejection sampling
def rejection_sampling(iteration=1000):
  samples = []
  i, j = 0, 0
  M = 1
  while i < iteration:
    u = np.random.uniform(0, 1)
    x = np.random.normal(0, 4) / 0.8
    if u < p_g_given_x_theta(x, theta=0) / M:
      samples.append(x)
      i += 1
    j += 1
  print("fraction of accepted samples: " + str(round(i/j, 2)))
  return samples

# importance sampling
def importance_sampling(iteration=1000):
  i, summation = 0, 0
  while i < iteration:
    x = np.random.normal(0, 4)
    summation += p_g_given_x_theta(x, theta=0, g=0)
    i += 1
  return summation / i

# Q1(d)
def p_x_given_theta(x, theta):
  return 1 / (4 * np.sqrt(2 * np.pi)) * np.exp(-(x - theta) ** 2 / 32)

```

```
def cauchy(theta):
  return 1 / (10 * np.pi * (1 + (theta / 10) ** 2))

def p_x_g_theta(theta):
  """
  the unnormalized density p(x = 1.7, g = 1, theta), as a function of theta
  """
  return f(1.7, theta) * p_x_given_theta(1.7, theta) * cauchy(theta)

# Metropolis-Hastings sampling
def metropolis_hastings(iteration=10000):
  accepted, rejected = [], []
  i, x = 0, 0
  while i < iteration:
    u = np.random.uniform(0, 1)
    x_new = np.random.normal(x, 100)
    func = lambda a, b : p_g_given_x_theta(1.7, a) * p_x_given_theta(1.7, a) *
    cauchy(a) * norm.pdf(b, loc=a, scale=100)
    fraction = func(x_new, x) / func(x, x_new)
    if u < fraction:
      x = x_new
      accepted.append(x)
    else:
      rejected.append(x)
    i += 1
  return accepted, rejected


if __name__ == '__main__':
  # Q1(a) estimate the fraction of photons that get absorbed on average
  print("estimate of p(g = 0|theta = 0):" + str(round(numerical_integration(g=0,
    theta=0), 2)))

  # Q1(b) plot rejection sampling
  x = rejection_sampling(iteration=10000)
  f2 = plt.figure()
  ax2 = f2.add_subplot(111)
  count, bins, ignored = plt.hist(x, 100, density=True)
  plt.title("Rejection Sampling")

  # Q1(c) estimatep(g = 0|theta = 0)
  print("the estimate (importance sampling): " + str(round(importance_sampling(), 2)
    ))

  # Q1(d) plot unnormalized density p(x = 1.7, g = 1, theta)
  theta = np.linspace(-20,20,100) # 100 linearly spaced numbers
  y = p_x_g_theta(theta)
  f3 = plt.figure()
  ax3 = f3.add_subplot(111)
  plt.plot(theta, y)
  plt.title("Plot of Unnormalized Density: p(x = 1.7, g = 1, theta)")

  # Q1(e) plot Metropolis-Hastings sampling
  theta_accepted = metropolis_hastings(iteration=100000)[0]
  f4 = plt.figure()
  ax4 = f4.add_subplot(111)
  count, bins, ignored = plt.hist(theta_accepted, 100, density=True)
  plt.title("Metropolis-Hastings Sampling")
```

```
# plot the true unnormalized density
f5 = plt.figure()
ax5 = f5.add_subplot(111)
theta = np.linspace(-20,20,100)
y = p_g_given_x_theta(1.7, theta) * p_x_given_theta(1.7, theta) * cauchy(theta)
plt.plot(theta, y)
plt.title("The True Unnormalized Density")

# Q1(f) estimate the posterior probability
within = 0
for theta in theta_accepted:
  if -3 < theta < 3:
    within +=1
estimate_posterior = within / len(theta_accepted)
print("estimate of the posterior probability: " + str(round(estimate_posterior, 2)
  ))
plt.show()
```