MathDNN Homework 2

Department of Computer Science and Engineering
2021-16988 Jaewan Park

Problem 4

Let $\varphi(x) = -\log(x)$, then $\varphi''(x) = 1/x^2$ is nonnegative for all $x \in \mathbb{R}^+$. Therefore $-\log(x)$ is a convex function. Since $\varphi(x)$ is defined over $\mathbb{R}^+$ which is a convex set (it is obvious that $\forall x_1, x_2 \in \mathbb{R}^+$, $\eta x_1 + (1 - \eta)x_2\ (> 0) \in \mathbb{R}^+$), we can apply Jensen's inequality to $-\log(x)$. Therefore we obtain

$$\begin{aligned}
D_{KL}(p||q) &= \sum_{i=1}^{N} p_i \log\left(\frac{p_i}{q_i}\right) = \sum_{i=1}^{N} p_i \left(-\log\left(\frac{q_i}{p_i}\right)\right) = \sum_{i=1}^{N} p_i \varphi(\frac{q_i}{p_i}) \\
&= \mathbb{E}\left[\varphi\left(\frac{q_i}{p_i}\right)\right] \\
&\geq \varphi\left(\mathbb{E}\left[\frac{q_i}{p_i}\right]\right) \\
&= -\log\left(\sum_{i=1}^{N} p_i \frac{q_i}{p_i}\right) = -\log\left(\sum_{i=1}^{N} q_i\right) = -\log 1 \\
&= 0
\end{aligned}$$

when we let q_i/p_i a random variable with probability mass p_i. In cases where $p_i = 0$ or $q_i = 0$, the value of $p_i \log(p_i/q_i)$ is considered either 0 or ∞, so the sum maintains nonnegative.

Problem 5

Let $\varphi(x) = -\log(x)$, then $\varphi''(x) = 1/x^2$ is positive for all $x \in \mathbb{R}^+$. Therefore $-\log(x)$ is a strictly convex function. As described in **Problem 4**, $\mathbb{R}^+$ is a convex set, and if we let $X = q_i/p_i$ a random variable X is non constant. Therefore we can apply the strict Jensen's inequality to $-\log(x)$. Therefore we obtain

$$\begin{aligned}
D_{KL}(p||q) &= \sum_{i=1}^{N} p_i \log\left(\frac{p_i}{q_i}\right) = \sum_{i=1}^{N} p_i \left(-\log\left(\frac{q_i}{p_i}\right)\right) = \sum_{i=1}^{N} p_i \varphi(\frac{q_i}{p_i}) \\
&= \mathbb{E}\left[\varphi\left(\frac{q_i}{p_i}\right)\right] \\
&> \varphi\left(\mathbb{E}\left[\frac{q_i}{p_i}\right]\right) \\
&= -\log\left(\sum_{i=1}^{N} p_i \frac{q_i}{p_i}\right) = -\log\left(\sum_{i=1}^{N} q_i\right) = -\log 1 \\
&= 0.
\end{aligned}$$

For cases where $p_i = 0$ or $q_i = 0$ the inequality stil remains. If $p_i > 0$ and $q_i = 0$, $p_i \log\left(\frac{p_i}{0}\right) = \infty$, so the sum still maintains positive. If $p_i = 0$ and $q_i = 0$,$p_i \log\left(\frac{p_i}{0}\right) = 0$, but since $p \neq q$, $p_i = q_i = 0$ cannot be true for all i. Thus the sum maintains positive. If $p_i = 0$ and $q_i > 0$, $p_i \log\left(\frac{p_i}{0}\right) = 0$, but since $\sum q_i = 1 > 0$, $q_i = 0$ cannot be true for all i. Thus the sum maintains positive.

Compared to **Problem 4**, the difference is that $p \neq q$, and this results in $D_{KL}(p||q) = 0$ not being possible. Since $\forall i\ p_i \log\left(\frac{p_i}{q_i}\right) \geq 0$, for $D_{KL}(p||q)$ to be 0, $p_i \log\left(\frac{p_i}{q_i}\right) = 0$ should be satisfied at all i. In that $\sum p_i = \sum q_i = 1$, this can only be satisfied when $p = q$. Thus supposing $p \neq q$ eliminates the equality condition from **Problem 4**, resulting in a strict inequality to satisfy.

Problem 6

(a)

$$\begin{aligned}
\frac{\partial f_\theta(x)}{\partial u_i} &= \frac{\partial}{\partial u_i} \sum_{j=1}^{p} u_j \sigma(a_j x + b_j) \\
&= \sum_{j=1}^{p} \frac{\partial}{\partial u_i} u_j \sigma(a_j x + b_j) = \frac{\partial}{\partial u_i} u_i \sigma(a_i x + b_i) \\
&= \sigma(a_i x + b_i)
\end{aligned}$$

Therefore

$$\nabla_u f_\theta(x) = \left(\frac{\partial f_\theta(x)}{\partial u_1}, \cdots, \frac{\partial f_\theta(x)}{\partial u_p}\right) = (\sigma(a_1 x + b_1), \cdots, \sigma(a_p x + b_p)) = \sigma(ax + b).$$

(b)

$$\begin{aligned}
\frac{\partial f_\theta(x)}{\partial b_i} &= \frac{\partial}{\partial b_i} \sum_{j=1}^{p} u_j \sigma(a_j x + b_j) \\
&= \sum_{j=1}^{p} \frac{\partial}{\partial b_i} u_j \sigma(a_j x + b_j) = \frac{\partial}{\partial b_i} u_i \sigma(a_i x + b_i) \\
&= u_i \sigma'(a_i x + b_i)
\end{aligned}$$

Therefore

$$\nabla_b f_\theta(x) = \left(\frac{\partial f_\theta(x)}{\partial b_1}, \cdots, \frac{\partial f_\theta(x)}{\partial b_p}\right) = (u_1 \sigma'(a_1 x + b_1), \cdots, u_p \sigma'(a_p x + b_p)) = \operatorname{diag}(\sigma'(ax + b))u.$$

(c)

$$\begin{aligned}\frac{\partial f_\theta(x)}{\partial a_i} &= \frac{\partial}{\partial a_i}\sum_{j=1}^{p} u_j\sigma(a_jx+b_j) \\ &= \sum_{j=1}^{p}\frac{\partial}{\partial a_i}u_j\sigma(a_jx+b_j) = \frac{\partial}{\partial a_i}u_i\sigma(a_ix+b_i) \\ &= u_ix\sigma'(a_ix+b_i)\end{aligned}$$

Therefore

$$\nabla_a f_\theta(x) = \left(\frac{\partial f_\theta(x)}{\partial a_1}, \cdots, \frac{\partial f_\theta(x)}{\partial a_p}\right) = (u_1x\sigma'(a_1x+b_1), \cdots, u_px\sigma'(a_px+b_p)) = \mathrm{diag}(\sigma'(ax+b))ux.$$

Problem 1

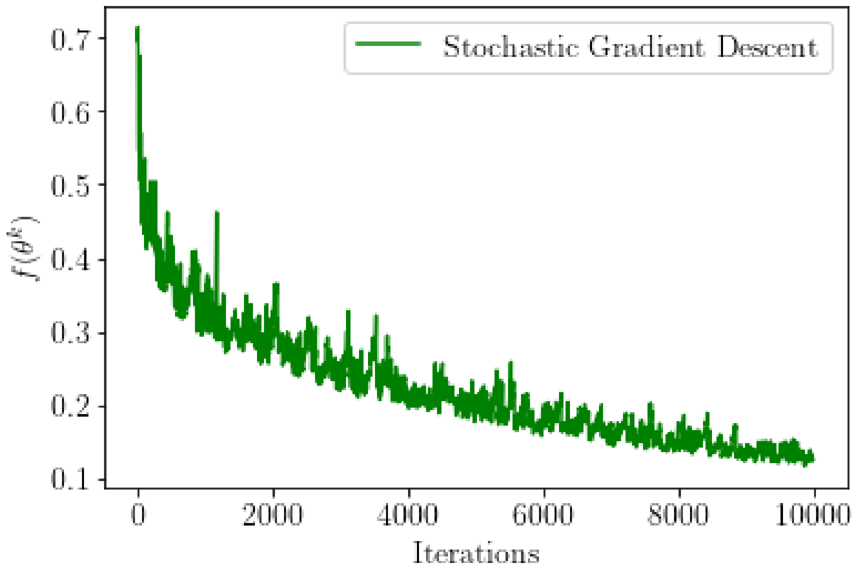
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import random as rd
%matplotlib inline
np.seterr(invalid='ignore', over='ignore')
```

[1]:
```
{'divide': 'warn', 'over': 'warn', 'under': 'ignore', 'invalid': 'warn'}
```

[2]:
```
N, p = 30, 20
np.random.seed(0)
X = np.random.randn(N, p)
Y = 2 * np.random.randint(2, size = N) - 1
```

[3]:
```
theta = np.zeros(p)
alpha = 0.1

K = 10000
f_val = []
for _ in range(K):
    ind = np.random.randint(N)
    theta -= alpha * (-Y[ind] * X[ind, :]) * np.exp(-Y[ind] * X[ind, :]@theta) / (1 +␣
↪np.exp(-Y[ind] * X[ind, :]@theta))
    f_val.append(1 / N * sum([np.log(1 + np.exp(-Y[i] * X[i, :]@theta)) for i in␣
↪range(N)]))

print("Minimizer :", theta)

plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.rc('font', size = 14)
plt.plot(range(K), f_val, color = "green", label = "Stochastic Gradient Descent")
plt.xlabel('Iterations')
plt.ylabel(r'$f(\theta^k)$')
plt.legend()
plt.show()
```

```
Minimizer : [-0.48651935  1.23818473  0.41953234  4.96438239 -1.58543155
-0.91604375
 -4.6043269  -2.67397853  1.16083962  2.06781531  4.87451857 -7.32035519
 -0.31449899 -2.10115613  4.67614609  5.37479007 -4.47990975  0.67006012
 -0.76432097 -7.14086801]
```



The minimizer value is of the above.

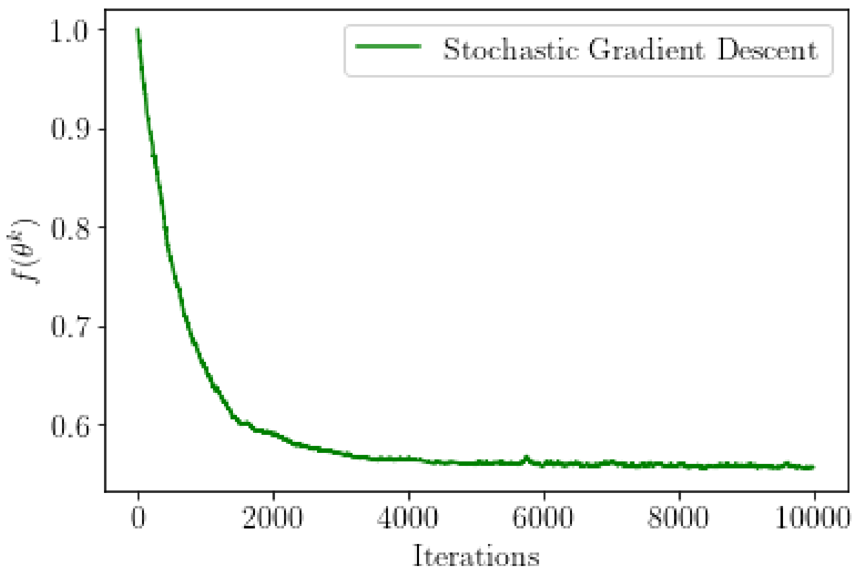
Problem 2

[4]:
```
N, p = 30, 20
np.random.seed(0)
X = np.random.randn(N, p)
Y = 2 * np.random.randint(2, size = N) - 1
```

[5]:
```
theta = np.zeros(p)
alpha = 0.001
lb = 0.1

K = 10000
f_val = []
non_diff_count = 0
for _ in range(K):
    if Y[ind] * X[ind, :]@theta == 1 : non_diff_count += 1
    ind = np.random.randint(N)
    theta -= alpha * ((-Y[ind] * X[ind, :]) + 2 * lb * theta) if Y[ind] * X[ind, :
↪]@theta < 1 else alpha * 2 * lb * theta
    f_val.append(1 / N * sum([max(0, 1 - Y[i] * X[i, :]@theta) + lb * theta.T@theta for␣
↪i in range(N)]))

print("Minimizer :", theta)
print("Non-Dfferentiable Point Encounters :", non_diff_count)

plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.rc('font', size = 14)
plt.plot(range(K), f_val, color = "green", label = "Stochastic Gradient Descent")
```

```
plt.xlabel('Iterations')
plt.ylabel(r'$f(\theta^k)$')
plt.legend()
plt.show()
```

```
Minimizer : [ 0.04245915  0.02304598 -0.37121936  0.17323276 -0.05328655
-0.16701866
 -0.28764165 -0.24306239  0.33406482 -0.11305208  0.07719667 -0.1188748
  0.1277413  -0.23376922  0.24730886  0.26902977 -0.37221256 -0.09283627
 -0.08036682 -0.45411963]
Non-Dfferentiable Point Encounters : 0
```



The minimizer value is of the above, and the process didn't encounter a point of non-differentiability.

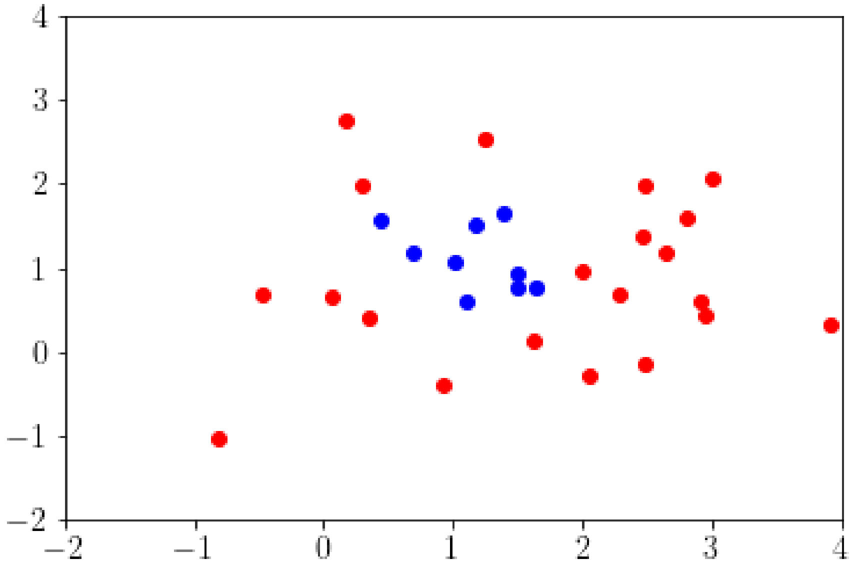
Problem 3

[6]:
```
N = 30
np.random.seed(0)
X = np.random.randn(2, N)
y = np.sign(X[0, :]**2 + X[1, :]**2 - 0.7)
theta = 0.5
c, s = np.cos(theta), np.sin(theta)
X = np.array([[c, -s], [s, c]])@X
X = X + np.array([[1], [1]])
```

Simply plotting the labeled points by their labels results in the following.

[7]:
```
plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.rc('font', size = 14)
plt.xlim(-2, 4)
```

```python
plt.ylim(-2,4)
for i in range(N):
    plt.scatter(X[0, i], X[1, i], color = "red" if y[i] == 1 else "blue")
plt.show()
```



The red points are ones labeled $y = 1$, and the blue points are those labeled $y = -1$. The plot simply shows that the data is not linearly separable in $\mathbb{R}^2$.

The given transformation can be applied as the following.

[8]:
```python
def phi(u, v):
    return np.array([1, u, u**2, v, v**2])
```

[9]:
```python
X_transformed = np.array([phi(X[0, i], X[1, i]) for i in range(N)]).T
```

Now apply logistic regression via SGD. We are targeting the minimization

$$\underset{a\in\mathbb{R}^5, b\in\mathbb{R}}{\text{minimize}} \frac{1}{N}\sum_{i=1}^{N} \log\left(1+\exp\left(-Y_i\left(a^\intercal X_i + b\right)\right)\right)$$

and we can simplify this as

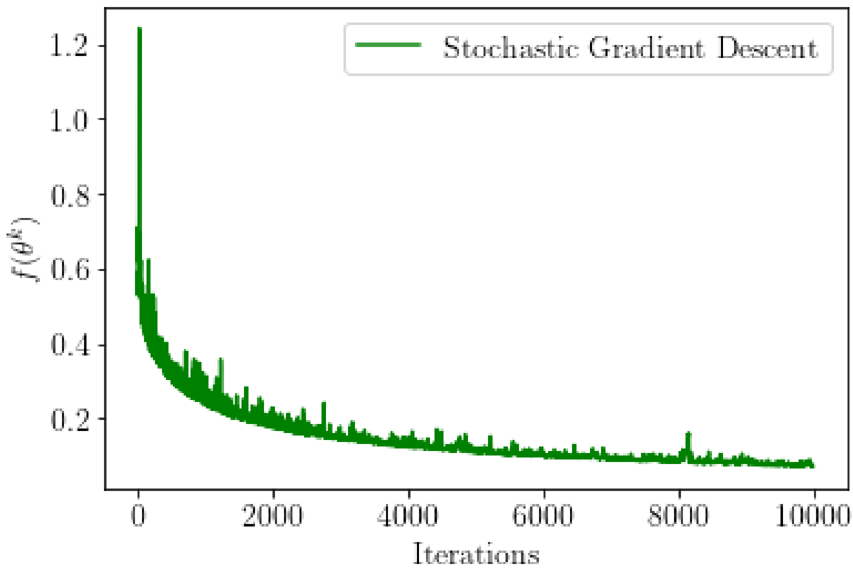
$$\underset{\theta\in\mathbb{R}^5}{\text{minimize}} \frac{1}{N}\sum_{i=1}^{N} \log\left(1+\exp\left(-Y_i X_i^\intercal \theta\right)\right).$$

As a result we are executing SGD in $\mathbb{R}^5$.

[10]:
```python
p = 5
theta = np.zeros(p)
alpha = 0.1

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

```
    ind = np.random.randint(N)
    theta -= alpha * (-y[ind] * X_transformed[:, ind]) * np.exp(-y[ind] *␣
↪X_transformed[:, ind]@theta) / (1 + np.exp(-y[ind] * X_transformed[:, ind]@theta))
    f_val.append(1 / N * sum([np.log(1 + np.exp(-y[i] * X_transformed[:, i]@theta)) for␣
↪i in range(N)]))

print("Minimizer :", theta)

plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.rc('font', size = 14)
plt.plot(range(K), f_val, color = "green", label = "Stochastic Gradient Descent")
plt.xlabel('Iterations')
plt.ylabel(r'$f(\theta^k)$')
plt.legend()
plt.show()
```

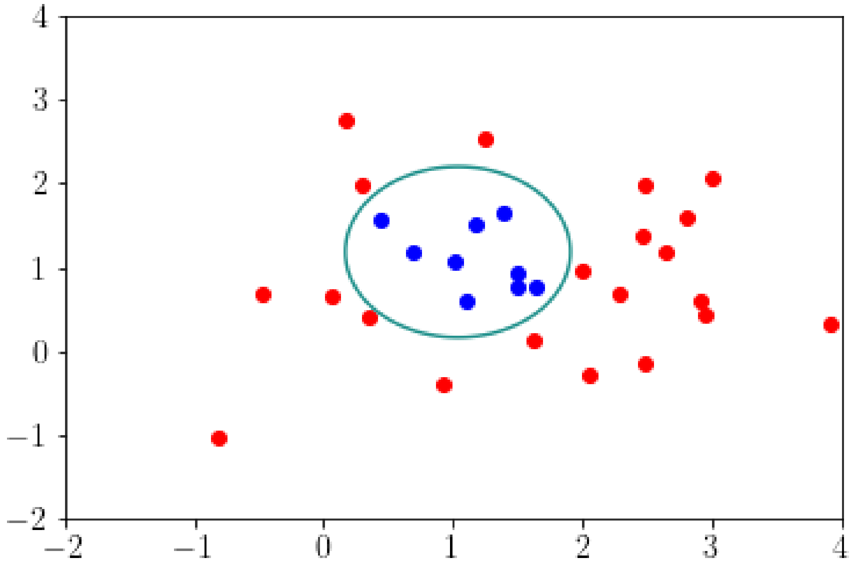
```
Minimizer : [  6.7173994  -10.32495424   4.9762752   -8.64245319   3.64384716]
```



The minimizer value is of the above, and we can visualize the decision boundary in $\mathbb{R}^2$.

[11]:
```
xx = np.linspace(-2, 4, 1024)
yy = np.linspace(-2, 4, 1024)
xx, yy = np.meshgrid(xx, yy)
Z = theta[0] + (theta[1] * xx + theta[2] * xx**2) + (theta[3] * yy + theta[4] * yy**2)
plt.contour(xx, yy, Z, 0)

plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.rc('font', size = 14)
for i in range(N):
    plt.scatter(X[0, i], X[1, i], color = "red" if y[i] == 1 else "blue")
```

```
plt.show()
```



Problem 7

[12]:
```
def f_true(x) :
    return (x-2)*np.cos(x*4)

def sigmoid(x) :
    return 1 / (1 + np.exp(-x))

def sigmoid_prime(x) :
    return sigmoid(x) * (1 - sigmoid(x))
```

[13]:
```
K = 10000
alpha = 0.007
N, p = 30, 50
np.random.seed(0)
a0 = np.random.normal(loc = 0.0, scale = 4.0, size = p)
b0 = np.random.normal(loc = 0.0, scale = 4.0, size = p)
u0 = np.random.normal(loc = 0, scale = 0.05, size = p)
theta = np.concatenate((a0,b0,u0))

X = np.random.normal(loc = 0.0, scale = 1.0, size = N)
Y = f_true(X)
```

[14]:
```
def f_th(theta, x) :
    return np.sum(theta[2*p : 3*p] * sigmoid(theta[0 : p] * np.reshape(x,(-1,1)) +␣
↪theta[p : 2*p]), axis=1)

def diff_f_th(theta, x) :
```

```
    partial_f_u = sigmoid(theta[0 : p] * x + theta[p : 2*p])
    partial_f_b = np.diag(sigmoid_prime(theta[0 : p] * x + theta[p : 2*p]))@theta[2*p :
↪3*p]
    partial_f_a = np.diag(sigmoid_prime(theta[0 : p] * x + theta[p : 2*p]))@theta[2*p :
↪3*p] * x
    return np.concatenate((partial_f_a, partial_f_b, partial_f_u), axis=None)
```

[15]:
```
xx = np.linspace(-2,2,1024)
plt.plot(X,f_true(X),'rx',label='Data points')
plt.plot(xx,f_true(xx),'r',label='True Fn')

for k in range(K) :
    ind = np.random.randint(N)
    theta -= alpha * (f_th(theta, X[ind]) - Y[ind]) * diff_f_th(theta, X[ind])
    if (k+1)%2000 == 0 :
        plt.plot(xx, f_th(theta, xx), label=f'Learned Fn after {k+1} iterations')

plt.legend()
plt.show()
```