Introduction to Deep Learning M2177.0043
Seoul National University

Homework #**2**

Donghak Lee

INSTRUCTIONS

- Anything that is received after the deadline will be considered to be late and we do not receive late homeworks. We do however ignore your lowest homework grade.
- Answers to every theory questions need to be submitted electronically on ETL. Only PDF generated from LaTex is accepted.
- Make sure you prepare the answers to each question separately. This helps us dispatch the problems to different graders.
- Collaboration on solving the homework is allowed. Discussions are encouraged but you should think about the problems on your own.
- If you do collaborate with someone or use a book or website, you are expected to write up your solution independently. That is, close the book and all of your notes before starting to write up your solution.

1 Learning a binary classifier with gradient descent

1.1 Derive the subgradient of the loss function

Small proof 1. $g_1, g_2 \in \delta f(x)$ and a, b $(a + b = 1, a \geq 0, b \geq 0)$.

$$a * (f(y) \geq f(x) + g_1^T(y - x))$$
$$b * (f(y) \geq f(x) + g_2^T(y - x))$$
$$\Rightarrow f(y) \geq f(x) + (ag_1 + bg_2)^T(y - x)$$

$\therefore$ Subdifferential is convex set.
Proof If $f_1, f_2 : \mathbb{R}^n \to \mathbb{R}$, convex and differentiable,

$$Max(f_1(y), f_2(y)) \geq f_1(y) \geq f_1(x) + \nabla f_1(x)^T(y - x)$$
$$Max(f_1(y), f_2(y)) \geq f_2(y) \geq f_2(x) + \nabla f_2(x)^T(y - x)$$

By **small proof 1**, $g = a\nabla f_1(x) + b\nabla f_2(x)$ when $(f_1(x) = f_2(x))$

$$\therefore \delta Max(f_1(x), f_2(x)) = \begin{cases} \nabla f_1(x) & (f_1(x) > f_2(x)) \\ \nabla f_2(x) & (f_1(x) < f_2(x)) \\ a\nabla f_1(x) + b\nabla f_2(x) & (f_1(x) = f_2(x)) \end{cases}$$

By **Proof**, subgradient of $max(0, 1 - y_i W^T x_i)$ is

$$g_i = \begin{cases} 0 & (1 < y_i W^T x_i) \\ -\frac{y_i x_i}{n} & (1 \geq y_i W^T x_i) \end{cases}$$

(WLOG, let a = 1, b = 0 to make simple)
Because $\frac{\lambda}{2}||W||^2$ is convex and differentiable, its subgradient is λW.
Small proof 2. If subgradient of $f_i(x) = g_i$, subgradient of $\Sigma f_i(x) = \Sigma g_i$

$$(\because \Sigma f_i(y) \geq \Sigma(f_i(x) + g_i^T(y - x)) = \Sigma f_i(x) + (\Sigma g_i)^T(y - x))$$

By **small proof 2**, subgradient of the loss function is

$$\sum_{i=1}^{n} g_i + \lambda W$$

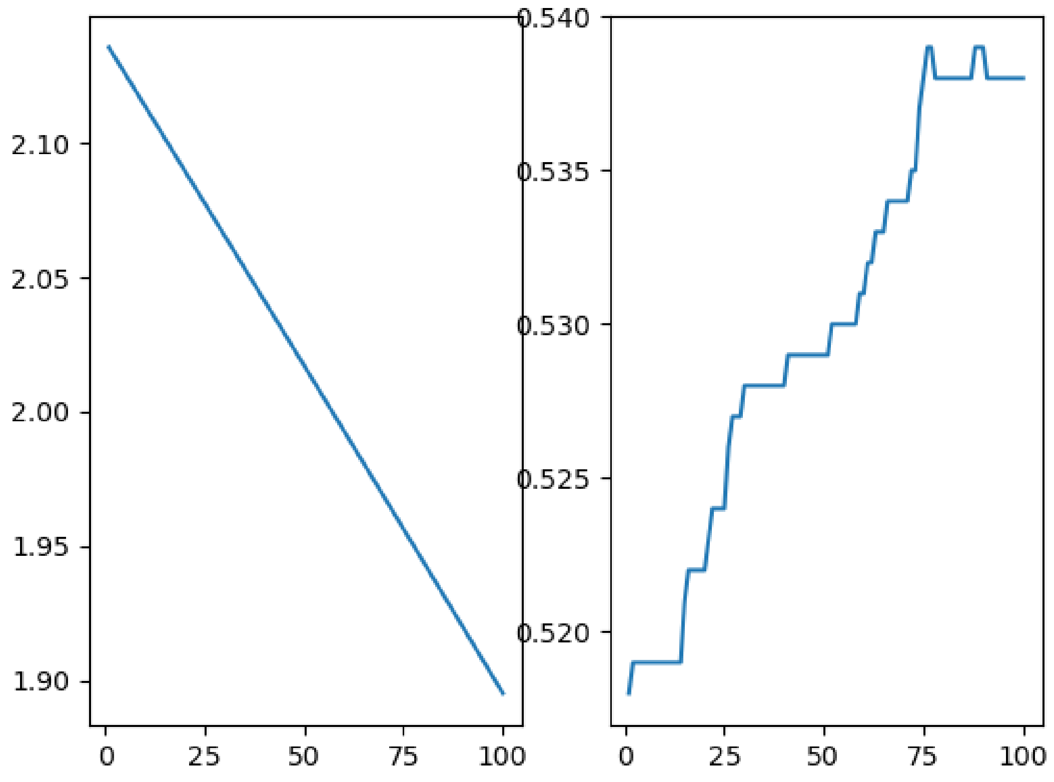
Introduction to Deep Learning M2177.0043
Seoul National University

Homework #**2**
Donghak Lee

1.2

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

np.random.seed(1337)

n = 1000
d = 100

X = np.vstack([np.random.normal(0.1, 1, (n//2, d)),
               np.random.normal(-0.1, 1, (n//2, d))])
y = np.hstack([np.ones(n//2), -1*np.ones(n//2)])
w0 = np.random.normal(0, 1, d)
w = w0

def subGrad(w):
    SG = np.zeros(d)
    SG += 0.1 * w
    for i in range(n):
        Xi = X[i, :]
        yi = y[i]
        yiwTXi = yi * np.matmul(w.T, Xi)
        if 1 > yiwTXi:
            SG += -(yi/n)*Xi
    return SG

def lossVal(w):
    ret = 0.0
    ret += (0.1/2)*np.matmul(w.T, w)
    for i in range(n):
        Xi = X[i, :]
        yi = y[i]
        yiwTXi = yi * np.matmul(w.T, Xi)
        ret += np.max([0.0, 1 - yiwTXi])/n
    return ret

def accuracy(w):
    match = 0
    for i in range(n):
        Xi = X[i, :]
        yi = y[i]
        wTXi = np.matmul(w.T, Xi)
        if np.sign(wTXi) == yi:
            match += 1
    return match/n

iter = []
```

```
f_value = []
c_accur = []

for i in range(100):
    w -= subGrad(w)*0.01
    iter.append(i+1)
    f_value.append(np.log(lossVal(w)))
    c_accur.append(accuracy(w))

plt.subplot(1, 2, 1)
plt.plot(iter, f_value)
plt.subplot(1, 2, 2)
plt.plot(iter, c_accur)
plt.show()
```

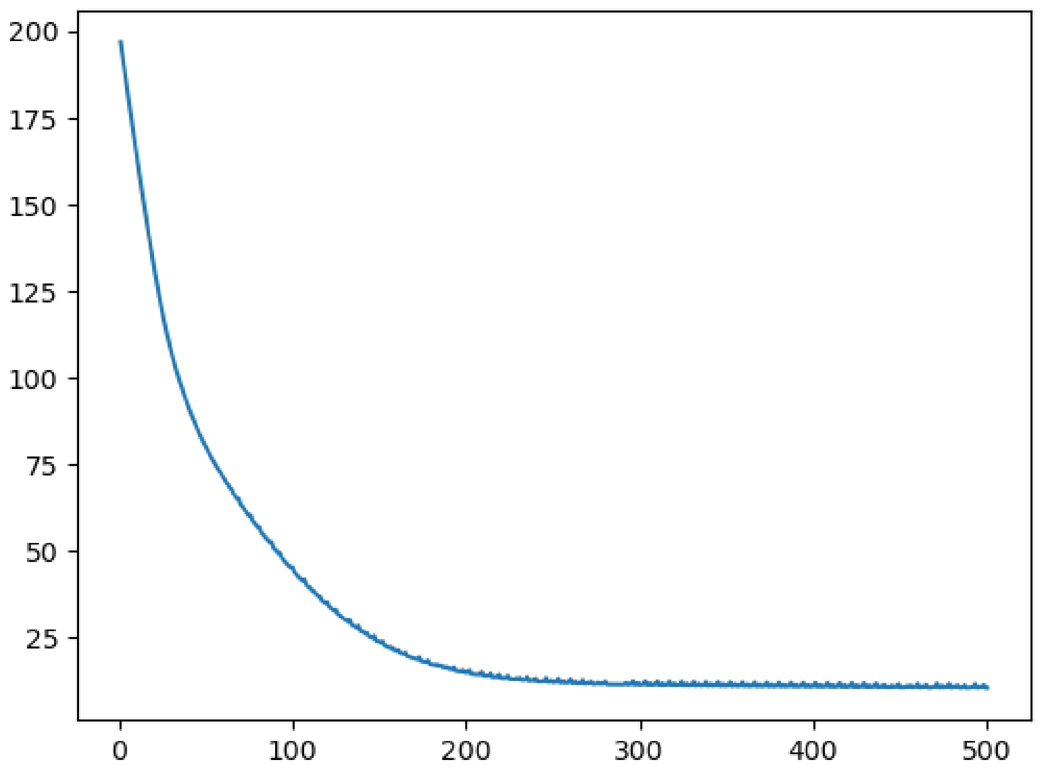


2 Matrix estimation with positive semidefinite constraint

```
import autograd.numpy as np
```

Homework #**2**
Donghak Lee

```
from autograd import grad
import matplotlib.pyplot as plt
np.random.seed(1337)

n = 5
A = np.random.normal(0, 1, (n, n))
S = A.dot(A.T)

def projection(X):
    eValue, eVector = np.linalg.eig(X)
    index = eValue.argsort()[::-1]
    eValue = eValue[index]
    eVector = eVector[:,index]
    for i in range(n):
        eValue[i] = np.max([0.0, eValue[i]])
    return eVector.dot(np.diag(eValue)).dot(eVector.T)

def target(X):
    ST = S.T
    tr = np.trace(ST.dot(X))
    det_log = np.log(np.linalg.det(X) + 1e-10)
    abs = np.sum(np.absolute(X))
    return tr - det_log + 0.1 * abs

iter = []
f_value = []

temp = np.random.normal(0, 1, (n, n))
X = temp.dot(temp.T)

for i in range(500):
    X -= grad(target)(X)*0.01
    X = projection(X)
    iter.append(i+1)
    f_value.append(target(X))

plt.plot(iter, f_value)
plt.show()
```


200
175
150
125
100
75
50
25
0
100
200
300
400
500