Introduction to Deep Learning M2177.0043
Seoul National University

Homework #(**HW2**)
Seo Junwon

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 the subgradient of the loss function.

Theorem 1. *If $F = \Sigma f_i$, the subgradient of F, "G" is sum of the subgradients of $f_i(:: g_i) : \iff G = \Sigma g_i$*

Proof. $F(y) = \Sigma f_i(y) \geq \Sigma\{f(x)_i + g_i^T(y-x)\}, \forall y$
$\{f(x)_i + g_i^T(y-x)\} = F(x) + G^T(y-x)$
$\therefore F(y) \geq F(x) + G^T(y-x) \forall x$
Therefore, $G = \Sigma g_i$ □

$f_i = \frac{1}{n} \times max(0, 1 - y_i w^t x_i)$
The subgradient of $max(f_1, f_2)$ when both of them are convex differentiable is

$$g = \begin{cases} \bigtriangledown f_1 & \text{for} f_1 > f_2 \\ \bigtriangledown f_2 & \text{for} f_2 > f_1 \\ \text{any point on the line segment between } \bigtriangledown f_1 \text{and } \bigtriangledown f_2 & \text{if} f_2 = f_1 \end{cases}$$

In case of all the "g_i" s, f_1 is 0, f_2 is $1 - y_i w^t x_i$
Both of them are convex and differentiable.
So, the subgradient of $f_i = \frac{1}{n} \times max(0, 1 - y_i w^t x_i)$ is as follows.

$$g_i = \begin{cases} 0 & \text{for } 1 \leq y_i w^t x_i \\ \frac{-y_i x_i}{n} & \text{for } 1 > y_i w^t x_i \end{cases}$$

By Theorem 1, the subgradient of the loss function is as follows

$$G = \Sigma g_i + \lambda w$$

1.2 solve the optimization problem

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

Introduction to Deep Learning M2177.0043
Seoul National University

Homework #(**HW2**)
Seo Junwon

```
np.random.seed(1337)
lam = 0.1
n = 1000
d = 100
n_step = 100
step_size = 0.01

# Loss function
def loss(w, x, y, lam) :
    height, width = x.shape
    s = 0
    for i in range(height) :
        s += max(0, 1-y[i] * np.dot(w, x[i]))
    s /= height
    s += lam / 2 * np.linalg.norm(w)
    return s


def accuracy(w, x, y) :
    height= y.shape[0]
    res = np.sign(np.matmul(x, w))
    correct = sum(np.equal(res, y))
    return correct / height

def subgradient(w, x, y, lam) :
    height= y.shape[0]
    d = w.shape[0]

    vec = np.zeros_like(w)
    for i in range(height) :
        xi = x[i]
        yi = y[i]
        if yi * (np.dot(xi,w)) < 1 :
            vec += -yi * xi #sigma(g_i)
    vec /= height # divide by n

    g = lam * w  + vec # W + sigma(g_i)
    return g


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

step = []
```

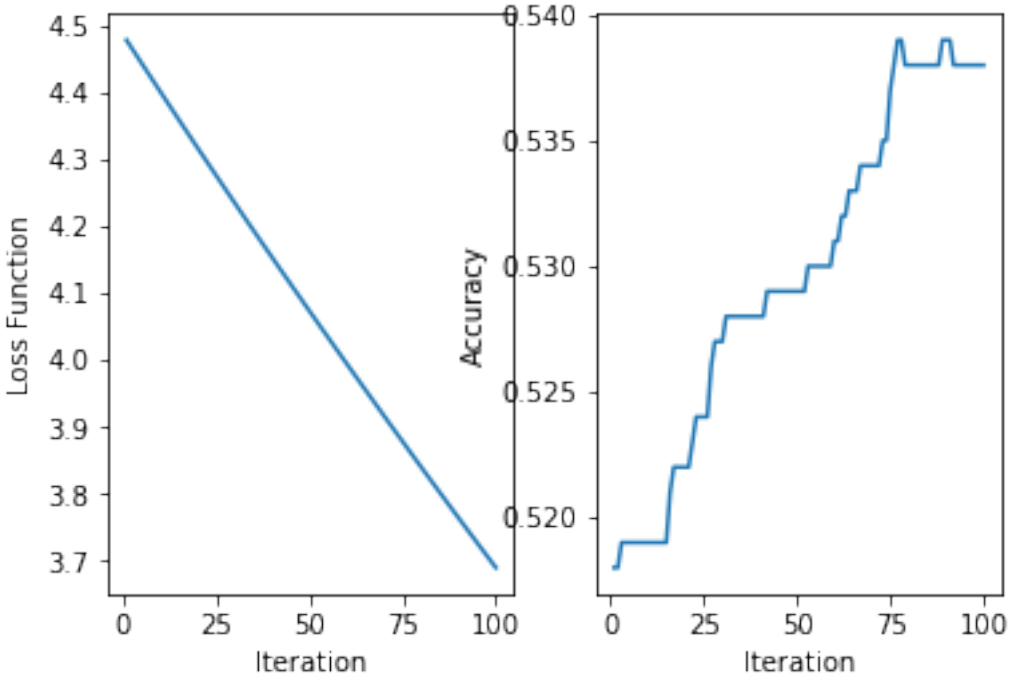
```
losses = []
accuracies = []

for i in range(n_step) :
    fun_val = loss(w = w, x = X, y = y, lam = lam)
    accur = accuracy(w = w, x = X, y = y)
    step.append(i+1)
    losses.append(fun_val)
    accuracies.append(accur)
    g = subgradient(w, X, y, lam)
    w = w - g * step_size # update

plt.subplot(1, 2, 1)
plt.xlabel("Iteration")
plt.ylabel("Loss Function")
plt.plot(step, losses)

plt.subplot(1, 2, 2)
plt.xlabel("Iteration")
plt.ylabel("Accuracy")
plt.plot(step, accuracies)

plt.show()
```



---

2 Matrix estimation with positive semidefinite constraint

under the constaint of PSD, I used projection derived by eigenvalue decomposition.

```
import autograd.numpy as np
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)

A0 = np.random.normal(0, 1, (n,n))
X0 = A0.dot(A0.T)

num_step = 500
step_size = 0.01
alpha = 0.1

def obj_func(X) :
    ip = np.trace(S.T.dot(X)) # inner product -> sum of traces
    det = np.log(np.linalg.det(X)) # determinant
    norm = alpha * np.sum(np.absolute(X)) # 1-norm of the matrix
    return ip - det + norm

# projection onto PSD cone
# by 1. eigenvalue decomposition of matrix X (as X is PSD)
# assume (lambda)n < 0
# X=max{i,0}vivTi
# https://math.stackexchange.com/questions/2776803/matrix-projection-onto-positive-semi-de
def projection(X) :

    e_val, e_vec = np.linalg.eig(X)
    index = e_val.argsort()
    e_vec = e_vec[:,index] # column vector
    e_val = e_val[index]

    for i in range(n) :
        e_val[i] = max(0.0, e_val[i]) #X=ni=1max{i,0}vivTi
    return e_vec.dot(np.diag(e_val)).dot(e_vec.T)

# create a function to compute the gradient
grad_fun = grad(obj_func)

iteration = []
f_val = []
X = X0
for i in range(num_step) :
```

```
    f = obj_func(X)
    X -= grad_fun(X) * step_size
    X = projection(X)
    iteration.append(i+1)
    f_val.append(f)

plt.title("Q2")
plt.plot(iteration, f_val)
plt.xlabel("Iteration")
plt.xlabel("func_val")
```