HW6

Problem 1 (Convergence of gradient descent and stochastic gradient method)**.** Which of the following statements on gradient descent (GD) and the stochastic gradient method (SGM) applied to minimizing a convex and differentiable function f are true, without making any assumption on f other than that f has a minimum?

(a) If the stepsize of GD is chosen constant and sufficiently small, GD converges to a minimizer of f.

(b) If the stepsize of SGM is chosen constant and sufficiently small, SGM converges to a minimizer of f.

(c) There is a fixed choice of stepsizes α_k such that the average of the iterates of SGM, weighted by the stepsizes, converges to a minimizer of f.

k, weighted by 2k

a) True, since we are chosen the stepsize sufficiently small, eventhough it will be very slow, but it will be minimum eventually

b) False, according to the inequality
$E[f(\bar{x}_k)] - f(x^*) \le \frac{1}{2\gamma k}\|x^0 - x^*\|_2^2 + \frac{1}{2}B^2\gamma$
thus, with constant stepsize, the bound suggest that SGM doesn't converge to the optimum. instead it converges to a ball of size $B^2\gamma/2$

C) True.

Problem 2 (Stochastic subgradient methods, 6pts)**.** In this problem you will implement the subgradient and stochastic subgradient methods for minimizing the convex but nondifferentiable function

$$J(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \langle \mathbf{w}, \mathbf{x}_i \rangle + b) + \frac{\lambda}{2}||\mathbf{w}||^2$$

where $L(y, t) = \max\{0, 1 - yt\}$ is the hinge loss. This corresponds to the optimal soft margin hyperplane.

1. Determine $J_i(\mathbf{w}, b)$ such that

$$J(\mathbf{w}, b) = \sum_{i=1}^{n} J_i(\mathbf{w}, b).$$

$J_i(w,b) = \frac{1}{n}\left[L(y_i, \langle w, x_i\rangle + b) + \frac{\lambda}{2}\|w\|^2\right]$

2. Determine a subgradient $\mathbf{u}_i$ of each J_i with respect to the variable $\boldsymbol{\theta} = [b\ \mathbf{w}^T]^T$. A subgradient of J is then $\sum_{i=1}^{n} \mathbf{u}_i$.

   Note that for a convex function $f\colon \mathbb{R}^d \to \mathbb{R}$ that is convex, but not necessarily differentiable, a sub-gradient at θ is a vector $\mathbf{g} \in \mathbb{R}^d$ such that

$$f(\theta') \ge f(\theta) + \langle \theta' - \theta, \mathbf{g} \rangle \qquad \text{for all } \mathbf{x}' \in \mathbb{R}^d.$$

If f is differentiable at θ then the sub-gradient at θ is unique and equal to the gradient at θ. The sub-gradient is a natural generalization of the gradient of a convex function. For example, the function $f(\theta) = \max\{\langle \theta, \mathbf{x} \rangle, 0\}$ is not differentiable at $\theta = 0$. However, a sub-gradient at $\theta = 0$ is given by 0. At all points where $\langle \theta, \mathbf{x} \rangle$ is positive, f is differentiable and thus the sub-gradient is equal to $\mathbf{x}$; at all points where $\langle \theta, \mathbf{x} \rangle$ is negative, f is also differentiable and the sub-gradient is equal to 0.

$J_i' = \frac{1}{n}\left[L' + \lambda \|w\|\right]$

$L = \max\{0, 1 - yt\}$

subgradient

① $J_i(w,b) = \frac{1}{n}\left\{L(y_i, \langle w, x_i\rangle + b) + \frac{\lambda}{2}\|w\|^2\right\}$

$J_i(w,b) = \frac{1}{n}\left\{\max(0, 1 - y_i(\langle w x_i\rangle + b)) + \frac{1}{2}\lambda\|w^2\|\right\}$

if $1 - y_i(\langle w, x\rangle + b) > 0$

$y_i(\langle w, x\rangle + b) < 1$

$u_i' = \frac{1}{n}[-y_i \tilde{x}_i + \lambda \hat{w})$

② if $1 - y_i(\langle w, x\rangle + b) > 0$

$u_i = \frac{1}{n} \cdot \lambda \hat{w}$

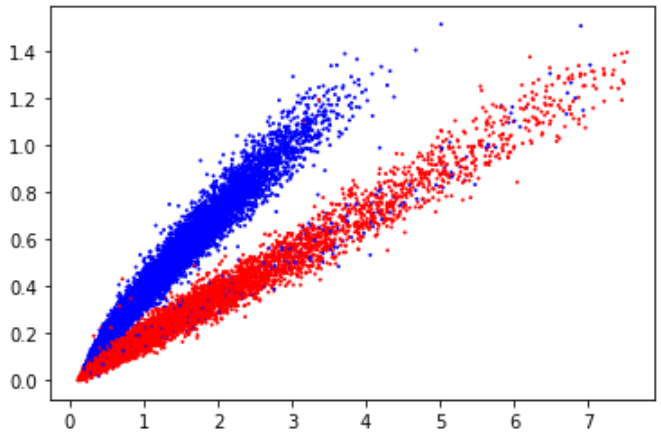
in summary $u_i = \begin{cases} \frac{1}{n}\left[-y_i \hat{x} + \lambda \hat{w}\right], & y_i \hat{x}^T \theta \le 1 \\ \frac{1}{n}\lambda \hat{w} & , \text{otherwise} \end{cases}$

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib inline
```

In [ ]:
```
# data
path_X = 'nuclear_x.csv'
path_y = 'nuclear_y.csv'
path=[path_X,path_y]

for i in range(2):
    with open(path[i], encoding='utf-8') as f:
        if i ==0:
            X = np.loadtxt(path[i],delimiter=',') #20000,2
        else:
            y = np.loadtxt(path[i], delimiter=',').reshape(-1,1)
```

In [ ]:
```
# visualization
fig,ax = plt.subplots()
ax.scatter(X[:,0],X[:,1],c=['red' if label == -1 else 'blue' for label in y],s=1)
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x1cf1993f910>
```

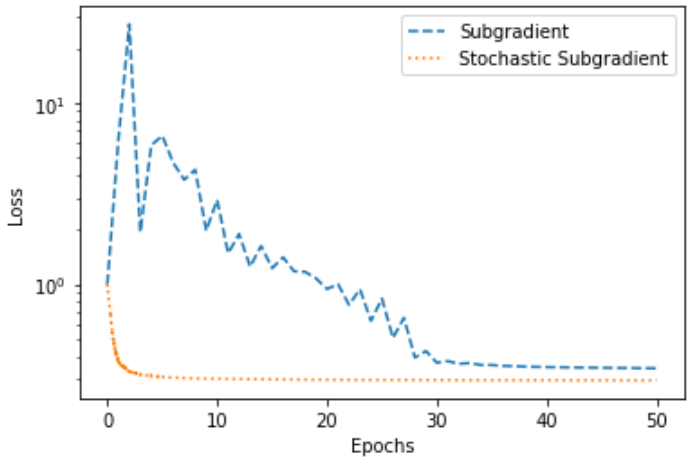


3&4

In [ ]:
```
def subgrad(X_aug,y,theta,lam=0.001):
    n1 = len(X_aug)
    n = 20000
    w = np.concatenate((np.zeros((1,1)),theta[1:,:].T),axis=1).T # 3,1 [b;w1;w2]
    mask = y*np.dot(X_aug,theta) <= 1 #20000,1
    grad = -((mask*y*X_aug).sum(0)).reshape(-1,1) + n1*lam*w#20000,3
    return grad/n

def subgrad_descent(X,y,batch_size,epochs):
    np.random.seed(42)
    X = np.concatenate((np.ones((X.shape[0],1)),X),axis=1)

    theta = np.zeros((X.shape[1],1))
    a = theta.copy()
    thetas=[a]

    for j in range(1,epochs+1):
        step_size = 100/j
        indices = list(np.arange(X.shape[0]))
        np.random.shuffle(indices)

        for k in range(0,len(X),batch_size):
            batch_indices = indices[k:k+batch_size] # 0-1 1-2 2-3
```

```
            X_batch = X[batch_indices]
            y_batch = y[batch_indices]
            grad = subgrad(X_batch,y_batch,a)
            a = a - step_size*grad
            thetas.append(a)
    return thetas

def loss(X,y,thetas,lam=0.001):
    ls = []
    thetas = np.array(thetas)
    for w,b in zip(thetas[:,1:,:],thetas[:,0,:]):
        t = X@w+b
        L = np.maximum(0,1-y*t)
        ls.append(L.mean()+(lam/2)*np.linalg.norm(w)**2)
    return ls

fig,ax = plt.subplots()
# subgradient
thetas = subgrad_descent(X,y,batch_size=20000,epochs=50)
Js = loss(X,y,thetas)
ax.plot(np.arange(len(Js)), Js, '--', label='Subgradient')

thetas_sto = subgrad_descent(X,y,batch_size=1,epochs=50)
Js_sto = loss(X,y,thetas_sto[::100])
ax.plot(np.arange(len(Js_sto)) / X.shape[0]*100, Js_sto, ':', label='Stochastic Subgradient')

plt.yscale('log')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x1cf19a06be0>
```



In [ ]:
```
print(f'Subgradient Method:\nw0={thetas[-1][1]}, w1={thetas[-1][2]}, b={thetas[-1][0]}, J={Js[-1]
print(f'Stochastic Subgradient Method:\nw0={thetas_sto[-1][1]}, w1={thetas_sto[-1][2]}, b={thetas
```

```
Subgradient Method:
w0=[-3.67021664], w1=[18.12076997], b=[-1.04167911], J=0.3441124087943893

Stochastic Subgradient Method:
w0=[-2.22976727], w1=[11.38929958], b=[-0.94546602], J=0.295309709368565
```

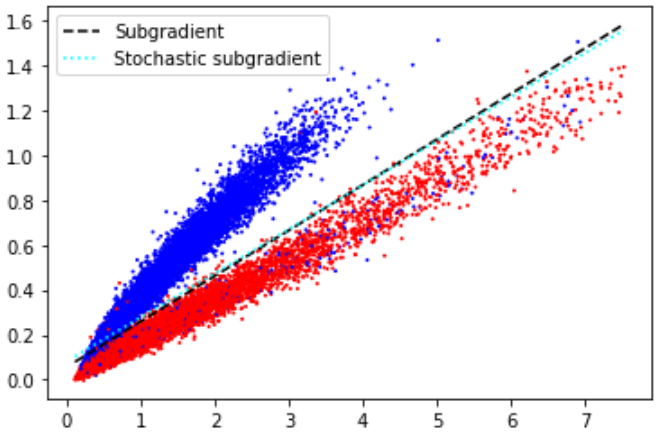
In [ ]:
```
def plot_decesion_boundary(x_min,x_max,w,b,linestyle,label,color):
    plt.plot([x_min,x_max],[(-b-x_min*w[0])/w[1],(-b-x_max*w[0])/w[1]],linestyle,label=label,c=co
```

Plot decision boundary of two methods

In [ ]:
```
x_min=X[:,0].min()
x_max=X[:,0].max()
```

```
plt.scatter(X[:,0],X[:,1],c=['red' if label == -1 else 'blue' for label in y],s=1)

plot_decesion_boundary(x_min,x_max,thetas[-1][1:],thetas[-1][0],'--','Subgradient',color='black')
plot_decesion_boundary(x_min,x_max,thetas_sto[-1][1:],thetas_sto[-1][0],':','Stochastic subgradie
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x1cf19d5fa00>
```



5. Comment on the (emperical) rate of convergence of the stochastic subgradient method relative to the subgradient mehtod.

The convergence rate of Stochasitic subgradient method is much more faster than subgradient method, as shown in the fig above. Because stochastic method consider only one example at each update.

In [ ]: