CS 260B-80 | Assignment #1 | Jason Chapman

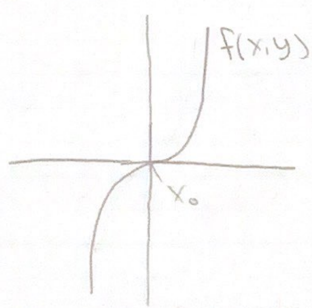
(1.)

a.) $f(x,y) = x^3 y$, $x_0 = 0$

$\nabla f = \langle 3x^2 y, x^3 \rangle$

$\nabla f(x_0) = 0$

Geographically, we know that despite $\nabla f(x_0) = 0$ for this function, x_0 is not a local minimum or a local maximum



b.) From smoothness upperbound: f is β-smooth, $f: \mathbb{R}^d \to \mathbb{R}$

$$\forall a,h \quad f(a+h) \le f(a) + \langle \nabla f(a), h \rangle + \frac{\beta}{2} \| \nabla f(a) \|_2^2$$

$$w_{k+1} = w_k - \eta \nabla f(w_k) \quad // \text{ GD}$$

$$\Rightarrow f(w_{k+1}) \le f(w_k) + \langle \nabla f(w_k), -\eta \nabla f(w_k) \rangle + \frac{\beta}{2} \| \eta \nabla f(w_k) \|_2^2$$

$$\Rightarrow f(w_{k+1}) \le f(w_k) - \eta \| \nabla f(w_k) \|_2^2 + \frac{\eta^2 \beta}{2} \| \nabla f(w_k) \|_2^2$$

Let $\eta = \frac{1}{\beta}$

$$f(w_{k+1}) \le f(w_k) - \frac{1}{2\beta} \| \nabla f(w_k) \|_2^2$$

In k GD steps, we have

$$f_* - f(w_0) \le f(w_k) - f(w_0) \le \frac{1}{2\beta} \sum_{i=0}^{k-1} \| \nabla f(w_i) \|_2^2$$

when f_* is global optimum

$$\Rightarrow \frac{1}{k} \sum_{i=0}^{k-1} \| \nabla f(w_i) \|_2^2 \le \frac{2\beta (f(w_0) - f_*)}{k}$$

So, for w_i we have $\| \nabla f(w_i) \|_2^2 \le \frac{2\beta (f(w_0) - f_*)}{k}$

And, to find our point $\| \nabla f(w) \|_2^2 \le \varepsilon$, we need

at least $\frac{2\beta (f(x_0) - f_*)}{\varepsilon^2}$ iterations

(2) In 1b, we showed that for a β-smooth function f,

$$f(x_{k+1}) \le f(x_k) - \frac{1}{2\beta} \| \nabla f(x_k) \|_2^2$$

For α-convex function that is smooth & $\alpha > 0$, we know

$$f(x^*) \ge f(x_k) + \langle \nabla f(x_k), x^* - x_k \rangle + \frac{\alpha}{2} \| x^* - x_k \|_2^2$$

$$f(x_k) - f(x^*) \le \langle \nabla f(x_k), x_k - x^* \rangle - \frac{\alpha}{2} \| x^* - x_k \|_2^2$$

$$= -\frac{1}{2} \| \sqrt{\alpha}(x_k - x^*) - \frac{1}{\sqrt{\alpha}} \nabla f(x_k) \|_2^2 + \frac{1}{2\alpha} \| \nabla f(x_k) \|_2^2$$

$$\le \frac{1}{2\alpha} \| \nabla f(x_k) \|_2^2$$

Combine inequalities:

$$f(x_{k+1}) - f(x^*) \le f(x_k) - f(x^*) - \frac{1}{2\beta} \| \nabla f(x_k) \|_2^2$$

$$\le f(x_k) - f(x^*) - \frac{\alpha}{\beta} \left(f(x_k) - f(x^*) \right)$$

$$= \left(1 - \frac{\alpha}{\beta}\right) \left(f(x_k) - f(x^*) \right)$$

Iterate this until

$$f(x_k) - f(x^*) \le \left(1 - \frac{\alpha}{\beta}\right)^k \left(f(x_0) - f(x^*) \right)$$

$$\Rightarrow \quad \| x_k - x^* \|_2^2 \le \left(1 - \frac{\alpha}{\beta}\right)^k \| x_0 - x^* \|_2^2$$

(5) Let x^* be any point that satisfies $\nabla f(x^*) = 0$ & therefore a local minimum

Proof: From convexity

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle \qquad \forall x, y$$

In particular,

$$f(y) \geq f(x^*) + \langle \nabla f(x^*), y - x \rangle \qquad \forall y$$

$// \nabla f(x^*) = 0,$

$$\Rightarrow \quad f(y) \geq f(x^*) \qquad \forall y$$

So, for any y, $f(y) \geq f(x^*)$, proving that for any point x^* that satisfies $\nabla f(x^*) = 0$, x^* is a <u>global</u> minimum

④ logit loss function: $\ell(a,b) = -a\log b - (1-a)\log(1-b)$

$h_w(x) = \sigma(w \cdot x)$

$\sigma(t) = \frac{1}{1+e^{-t}}$

$L(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \sigma(\langle w, x_i\rangle))$

a.)

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}\left[-y_i \log(\sigma(\langle w,x_i\rangle)) - (1-y_i)\log(1-\sigma(\langle w,x_i\rangle))\right]$$

$$\nabla L(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla_w\left[-y_i\log(\sigma(\langle w,x_i\rangle)) - (1-y_i)\log(1-\sigma(\langle w,x_i\rangle))\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n} -\left[\frac{y_i}{\sigma(\langle w,x_i\rangle)} \cdot \frac{\partial}{\partial w}\sigma(\langle w,x_i\rangle) + \frac{1-y_i}{(1-\sigma(\langle w,x_i\rangle))} \cdot \frac{\partial}{\partial w}(1-\sigma(\langle w,x_i\rangle))\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n} -\left[\frac{y_i}{\sigma(\langle w,x_i\rangle)} - \frac{1-y_i}{1-\sigma(\langle w,x_i\rangle)}\right]\frac{\partial}{\partial w}\sigma(\langle w,x_i\rangle)$$

// $\frac{\partial}{\partial w}\sigma(\langle w,x_i\rangle) = -(1+e^{-wx_i})^{-2} \cdot -x_i = \frac{x_i}{(1+e^{-wx_i})^2} = \sigma(\langle w,x_i\rangle)(1-\sigma(\langle w,x_i\rangle))x_i$

$$= \frac{1}{n}\sum_{i=1}^{n} -\left[\frac{y_i - \cancel{y_i\sigma(\langle w,x_i\rangle)} - \sigma(\langle w,x_i\rangle) + \cancel{y_i\sigma(\langle w,x_i\rangle)}}{\cancel{\sigma(\langle w,x_i\rangle)}\cancel{(1-\sigma(\langle w,x_i\rangle))}}\right]\cancel{\sigma(\langle w,x_i\rangle)}\cancel{(1-\sigma(\langle w,x_i\rangle))}x_i$$

$$= \frac{1}{n}\sum_{i=1}^{n} -\left[y_i - \sigma(\langle w,x_i\rangle)\right]x_i$$

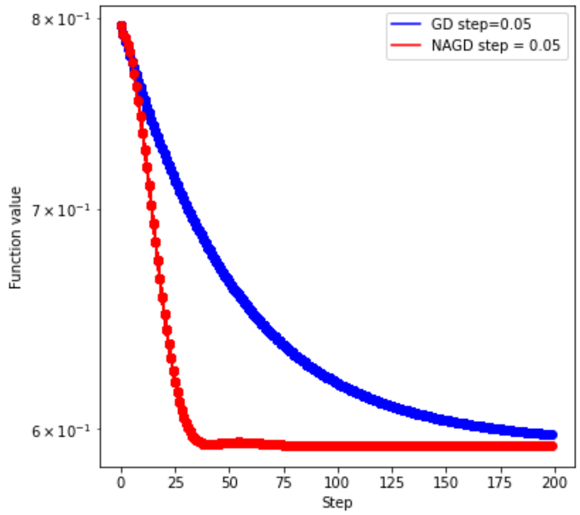
$$\Rightarrow \boxed{\nabla L(w) = \frac{1}{n}\sum_{i=1}^{n}\left\{\left[\sigma(\langle w,x_i\rangle) - y_i\right]x_i\right\}}$$

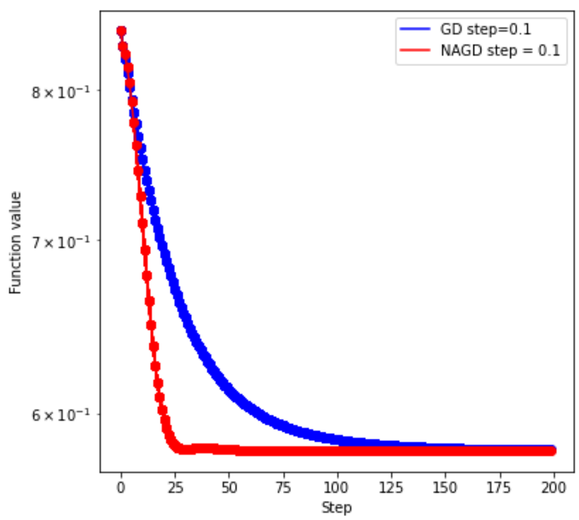
b.)

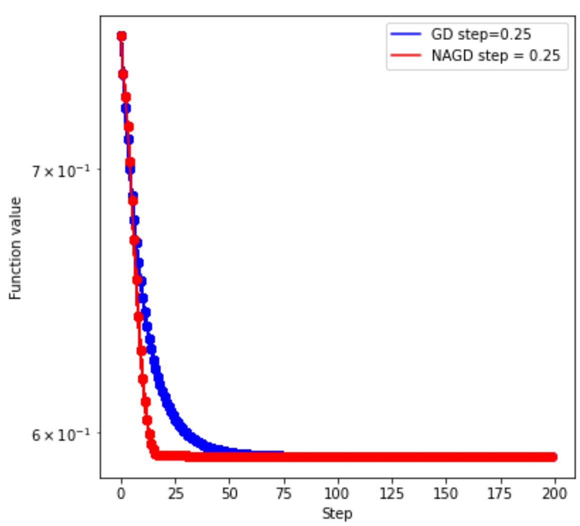
i.)

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
import time

def gradient_descent(xinit,steps,gradient):
    xs = [xinit]
    x = xinit
    for step in steps:
        x = x - step*gradient(x)
        xs.append(x)
    return np.array(xs)

def nagd(winit,gradient,eta,nsteps):
    ws = [winit]
    u = v = w = winit
    for i in range(nsteps):
        etai = (i+1)*eta/2
        alphai = 2/(i+3)
        w = v - eta*gradient(v)
        u = u - etai*gradient(v)
        v = alphai*u + (1-alphai)*w
        ws.append(w)
    return np.array(ws)

def plotting(Ys,labels=['1','2','3','4','5','6'],ylabel='Function value'):
    colors = ['blue','red','green','black','cyan','purple','pink']
    fig, ax = plt.subplots(figsize=(6,6))
    T = len(Ys[0])
    plt.yscale('log')
    for t in range(T):
        for j in range(len(Ys)):
            plt.plot(range(t),Ys[j][:t],color=colors[j],marker='o')
    handles = []
    for i in range(len(Ys)):
        handles.append(Line2D([0], [0], color=colors[i], label=labels[i]))
    plt.legend(handles = handles, loc = 'upper right')
    plt.xlabel('Step')
    plt.ylabel(ylabel)

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

def y_labeling(p):
    randint = np.random.uniform(0,1)
    if randint < p:
        return 1
    else:
        return 0


def sig_LR(X,Y,w):
    n,d = X.shape
    return -(1/n)*(Y.dot(np.log(sigmoid(X.dot(w)))) + (1-Y).dot(np.log(1-sigmoid(X.dot(w)))))

def sig_LR_gradient(X,Y,w):
    n,d = X.shape
    return (1/n)*(X.T.dot(sigmoid(X.dot(w)) - Y))

n, d = 1000, 20
X = np.random.normal(0, 1, (n, d))
wstar = np.random.normal(0, 1, d)
wstar = wstar/np.linalg.norm(wstar) # unit-norm
Y = []
for i in range(n): Y.append(y_labeling(sigmoid(X[i].dot(wstar))))
Y = np.array(Y)
w0 = np.random.normal(0, 1, d)
w0 = w0/np.linalg.norm(w0)

objective = lambda w: sig_LR(X, Y, w)
gradient = lambda w: sig_LR_gradient(X, Y, w)

wgd = gradient_descent(w0, [0.1]*300, gradient)
wnagd = nagd(w0,gradient,0.1,300)

obj_gd = [objective(w) for w in wgd]
obj_nagd = [objective(w) for w in wnagd]
obj_wstar = [objective(wstar) for w in wgd]

plotting([obj_gd,obj_nagd],['GD step=0.1','NAGD step = 0.01'])

distance_gd = [np.linalg.norm(w - wstar) for w in wgd]
distance_nagd = [np.linalg.norm(w - wstar) for w in wnagd]

plotting([distance_gd,distance_nagd],['GD step=0.1','NAGD step = 0.01'],'Distance to Optimum')
```

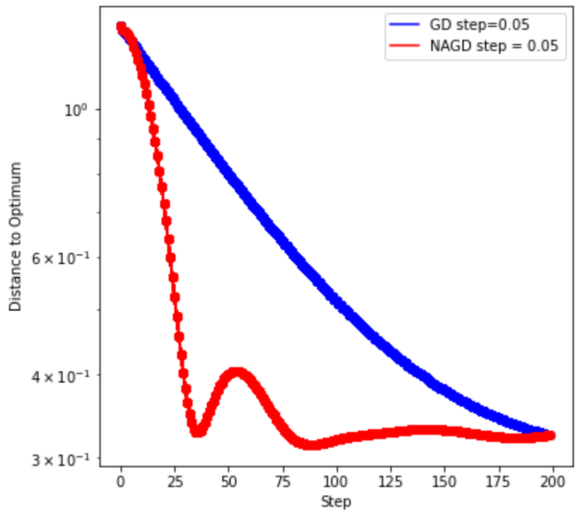
ii.)

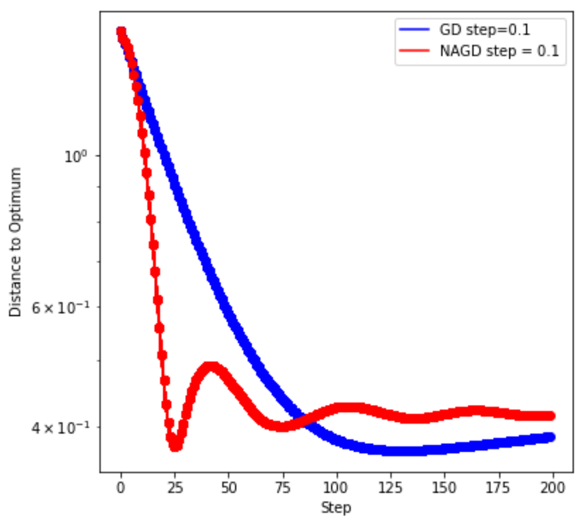

GD step=0.05
NAGD step = 0.05
Function value
Step

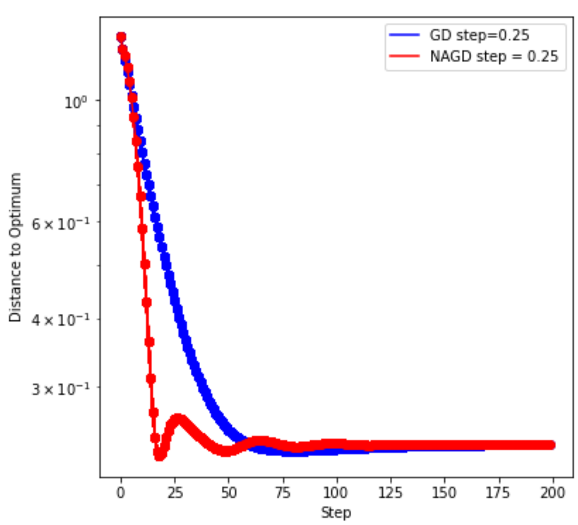

GD step=0.1
NAGD step = 0.1
Function value
Step


GD step=0.25
NAGD step = 0.25
Function value
Step

iii.)


GD step=0.05
NAGD step = 0.05
Distance to Optimum
10^0
6×10^{-1}
4×10^{-1}
3×10^{-1}
0
25
50
75
100
125
150
175
200
Step


GD step=0.1
NAGD step = 0.1
Distance to Optimum
10^0
6×10^{-1}
4×10^{-1}
0
25
50
75
100
125
150
175
200
Step


GD step=0.25
NAGD step = 0.25
Distance to Optimum
10^0
6×10^{-1}
4×10^{-1}
3×10^{-1}
0
25
50
75
100
125
150
175
200
Step

5.)

```
## Number 5 ##

def lsr_cost(X,Y,w):
    n,d = X.shape
    return (1/n)* np.linalg.norm(X.dot(w) - Y)**2

def lsr_gradient(X,Y,w):
    n,d = X.shape
    return (2/n)*X.T.dot(X.dot(w) - Y)

n,d = 1000, 1000
X = np.random.normal(0, 1, (n, d))
wstar = np.random.normal(0, 1, d)
noise = np.random.normal(0, 0.1, n)
Y = X.dot(wstar) + noise
objective = lambda w: lsr_cost(X, Y, w)
gradient = lambda w: lsr_gradient(X, Y, w)
w0 = np.random.normal(0, 1, d)

t1 = time.time()
ws = gradient_descent(w0, [0.05]*100, gradient)
gd = [objective(w) for w in ws]
time_gd = time.time() - t1

t2 = time.time()
wnagd = nagd(w0,gradient,0.05,100)
nagd = [objective(w) for w in wnagd]
time_nagd = time.time() - t2

t3 = time.time()
w_formula = np.linalg.lstsq(X,Y,rcond=None)[0]
#w_formula1 = np.linalg.inv(X.T.dot(X)).dot(Y.dot(X))
formula = [objective(w_formula) for w in ws]
time_formula = time.time() - t3
```

To analyze the time of running GD, NAGD, and the LSR formula, I ran the above code with varying n and d. At small values of n and d, there was not a significant advantage to using GD or NAGD as compared to the least squares regression function. However, as n and d were increased it was apparent that GD and NAGD were advantages over the least squares regression function. Large values of d compared to n showed the greater advantage of using GD and NAGD over the least squares regression function. This is expected when we look at the computational cost of each of these. GD and NAGD are O(nd), while the least squares regression function is O(nd^2 + d^3).