HW 5: $E_{in} > 0.008$; $\Delta = 0.1$; $d = 8$;

Plug into given equation hence;

$$0.008 \leq \left(1 - \frac{8+1}{N}\right)(0.1)^2$$

$$1 - \frac{(0.008)}{(0.1)^2} \leq \frac{9}{N}$$

$$0.2 \leq \frac{9}{N};$$

$$N > \frac{9}{0.2} > 45$$

[C] is correct because it is the smallest number of examples required for an $E_{in} > 0.008$ as shown by the math.

(2) We are looking at $\text{sign}(1 \cdot \tilde{w}_0 + x_1^2 \cdot \tilde{w}_1 + x_2^2 \tilde{w}_2)$.

According to the figure as x_2 gets bigger then the function should most likely return positive and as x_1 gets bigger it should return negative. Hence for $\tilde{w}_1 < 0$ and $\tilde{w}_2 > 0$ we will get a positive sign for big $|x_2|$ & a negative sign for big $|x_1|$. Hence the correct answer is [d].

(3) We know that $d_{VC} \leq \tilde{d} + 1$ from lecture 9 and here we have $\tilde{d} = 14$, by counting the resulting vector, hence $d_{VC} \leq 14 + 1$ hence $d_{VC} \leq 15$ thus the correct answer is [C].

(4) Find partial derivative with respect to u using chain rule.

$$E(u,v)_u = 2\left(ue^v - 2ve^{-u}\right)\left(e^v - \left(-2ve^{-u}\right)\right)$$

hence: $E(u,v)_u = 2\left(ue^v - 2ve^{-u}\right)\left(e^v + 2ve^{-u}\right)$

hence the correct answer is [e].

(5) Scratch work

$$2\left(ue^v - 2ve^{-u}\right)\left(e^v + 2ve^{-u}\right)$$

$$2\left(ue^v - 2ve^{-u}\right)\left(ue^v - 2e^{-u}\right)$$

See Below Code

(10) In PLA, the weight vector is updated whenever a point is misclassified thus we need to find an error function that adjusts the weights only for misclassified points where determining a, b, d as they always adjust the weight vector. Now looking at c & d, we know that for PLA the weight vector is updated by: $w(t) + yx_n$ and we know for SGD, the weight vector is updated by: $w(t) - \eta \nabla E_{in}$ hence eliminating c as it will always adjust the weight positively due to it being squared. Hence the correct answer is e. This makes sense because e, as seen in the equations for how the weights are updated in each, in e, if a point is misclassified hence we have opposit signs resulting in negative then e will return $y_n w^T x_n$ because $\min(0, -y_n w^T x_n)$ is $-y_n w^T x_n$ which will be converted to positive by the negative sign in front of min in e hence updating the weight vector however, when a point is correctly classified hence they are the same sign then e will return zero as it is never less than any positive hence not updating the weight vector thus correctly implementing the PLA. Thus by this reasoning [e] is the correct answer.

(8 and 9)

cs156aHW5.py × ml2.py

cs156aHW5.py > ggradient

```python
def get_slope(x1,x2,y1, y2):
    slope = ((y2 -y1)/(x2-x1))
    intercept = y1 - (slope * x1)
    return(slope, intercept)
def classify(point, line):
    if ((line[0] * point[1]) + (line[1])) >point[2]:
        return -1
    else:
        return 1
def ggradient(x, y, w):
    return -((numpy.dot(y,x))/(1 + numpy.e **(numpy.dot(numpy.dot(y, numpy.array(w).T), x))))
def algo(N):
    p1 = [1, random.uniform(-1, 1), random.uniform(-1, 1)]
    p2 = [1, random.uniform(-1, 1), random.uniform(-1, 1)]
    line = get_slope(p1[1], p2[1], p1[2], p2[2])
    data = []
    score = []
    eta = 0.01
    for i in range(N):
        curr = [1, random.uniform(-1, 1), random.uniform(-1, 1)]
        data.append(curr)
        score.append(classify(curr, line))

    w = [0, 0, 0]
    ww = [1, 1, 1]
    p = 0
    while (numpy.linalg.norm(numpy.subtract(ww,w)) > 0.01):
        ww = w
        for i in range(N):
            w1 = ggradient(data[i], score[i], w)
            w = numpy.subtract(w, (numpy.dot(eta, w1)))
        p += 1

    def Eout(ow):
        nscores = []
        tpoints = []
        test_points = 1000
        eout = 0
        for i in range(test_points):
            curr = [1, random.uniform(-1, 1), random.uniform(-1, 1)]
            tpoints.append(curr)
            nscores.append(classify(curr, line))

        for i in range(test_points):
            eout += numpy.log(1 + numpy.exp(-nscores[i] * numpy.dot(numpy.array(ow).T, tpoints[i])))
        return eout/test_points
    return (Eout(w), p)


def q8_and_q9():
    err = 0
    p = 0
    for i in range(100):
        res = algo(100)
        err += res[0]
        p += res[1]
    return (err/100, p/100)

fin = q8_and_q9()
print("epochs: ", fin[1], "Eout: ", fin[0])
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
epochs:  337.72 Eout:  0.10433496949297881
thierno@thiernos-MacBook-Pro CS  156a % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/cs156aHW5.py"
epochs:  338.62 Eout:  0.10069020707378556
thierno@thiernos-MacBook-Pro CS  156a %
```

(8) By the code above the correct answer is [d] as it is the closest to E_{out}=0.1007 gotten above.

(9) By the code above the correct answer is [d] as it is the closest to epoch=339 gotten above.

5&6

Get Started | cs156aHW5.py

Users > thierno > Desktop > Caltech > Smore > Fall 22 > CS 156a > cs156aHW5.py > q5_and_6

```python
import numpy

def func(u, v):
    val = ((u * numpy.e**v) - (2 * v * numpy.e**(-u))) ** 2
    return val
def gradient(u, v):
    Eu = 2 * ((u * numpy.e**v) - (2 * v * numpy.e**(-u))) * ((numpy.e**v) + (2 * v * numpy.e**(-u)))
    Ev = 2 * ((u * numpy.e**v) - (2 * v * numpy.e**(-u))) * ((u * numpy.e**v) - (2 * numpy.e**(-u)))
    return (Eu, Ev)

def q5_and_6(u, v):
    iteration = 0
    bar = 10 ** (-14)
    eta = 0.1
    curr = func(u, v)
    while curr >= bar:
        grad = gradient(u, v)
        u -= grad[0]*eta
        v -= grad[1]*eta
        curr = func(u, v)
        iteration += 1
    return (iteration, u, v)

def q5():
    print("Iterations: ", q5_and_6(1,1)[0])
def q6():
    print("u:",  q5_and_6(1, 1)[1])
    print("v:",  q5_and_6(1, 1)[2])
q5()
q6()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER: VARIABLES — Python - thierno

```
thierno@Thiernos-MacBook-Pro ~ % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/cs156aHW5.py"
Iterations:  10
u: 0.044736290397782 07
v: 0.023958714099141746
thierno@Thiernos-MacBook-Pro ~ %
```

(5) By the above code, the correct answer is [d] as it is the exact value gotten above.

(6) By the above code, the correct answer is [e] as it has the closest values of $u = 0.04474$ & $v = 0.02396$ gotten above.

(1)

Get Started | cs156aHW5.py

Users > thierno > Desktop > Caltech > Smore > Fall 22 > CS 156a > cs156aHW5.py > ...

```python
from tkinter import E
import numpy

def func(u, v):
    val = ((u * numpy.e**v) - (2 * v * numpy.e**(-u))) ** 2
    return val

def gradient(u, v):
    Eu = 2 * ((u * numpy.e**v) - (2 * v * numpy.e**(-u))) * ((numpy.e**v) + (2 * v * numpy.e**(-u)))
    Ev = 2 * ((u * numpy.e**v) - (2 * v * numpy.e**(-u))) * ((u * numpy.e**v) - (2 * numpy.e**(-u)))
    return (Eu, Ev)

def q5_and_6(u, v):...

def q5():...

def q6():
    print("u:",  q5_and_6(1, 1)[1])
    print("v:",  q5_and_6(1, 1)[2])

def coord_descent(u, v, n):
    eta = 0.1
    for i in range(n):
        grad = gradient(u, v)
        u -= grad[0]*eta
        grad = gradient(u, v)
        v -= grad[1]*eta
    return func(u, v)

print(coord_descent(1, 1, 15))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER: VARIABLES — Python - thierno

```
thierno@Thiernos-MacBook-Pro ~ % /usr/bin/python3 "/Users/thierno/Desktop/Caltech/Smore/Fall 22/CS  156a/cs156aHW5.py"
0.13981379199615324
thierno@Thiernos-MacBook-Pro ~ %
```

Live Share — Ln 28, Col 1 — Spaces: 4 — UTF-8 — LF — Python — 3.10.6 64-bit

(1) By the code above, the correct answer is [a] as it is the closest to the error 0.1398 gotten above.