Linear Regression Error

1. (c) 100

$\mathbb{E}_D[E_{in}(w_{lin})] = \sigma^2(1 - \frac{d+1}{N})$. For $\sigma = 0.1 and d = 8$, we have $\mathbb{E}_D[E_{in}(w_{lin})] = (0.1)^2(1 - \frac{9}{N})$. Of the choices for N, **[c] 100** is the smallest N that results in an expected E_{in} greater than 0.008.

2. (d) $w_1 < 0, w_2 > 0$

sign($w^T \cdot x$) = sign($w_0 \cdot 1 + w_1 \cdot x_1 + w_2 \cdot x_2$). When w_1 is negative and w_2 is positive, the expression can express the hyperbolic boundary. For example, when $|w_2|$ is very large and $|w_1|$ is not large, we get positive values and the opposite results in negative values. This is the hyperbolic boundary displayed.

3. (c) 15

We know $d_{vc} \leq d + 1$. Since the given input space has 14 parameters, **[c] 15** is the smallest answer choice not smaller than $d_{vc} \leq 15$.

Gradient Descent

4. (e) $2(e^v + 2ve^{-u})(ue^v - 2ve^{-u})$

$E(u, v) = (ue^v - 2ve^{-u})^2$
$\frac{\partial E}{\partial u} = 2(ue^v - 2ve^{-u})(e^v + 2ve^{-u})$

5. (d) 10 (see code)

6. (e) (0.045, 0.024) (see code)

```
import math

def partial_u(u,v):
    return 2 * (u * math.exp(v) - 2 * v * math.exp(-u)) * (math.exp(v) + 2 * v * math.exp(-u))

def partial_v(u,v):
    return 2 * (u * math.exp(v) - 2 * v * math.exp(-u)) * (u * math.exp(v) - 2 * math.exp(-u))

def E(u,v):
    return (u * math.exp(v) - 2 * v * math.exp(-u)) ** 2

def gradient(u, v, lr, threshold):
  iterations = 0
  error = E(u, v)
  while(error > threshold):
    du = partial_u(u, v)
    dv = partial_v(u, v)
    u = u - du * lr
    v = v - dv * lr
    error = E(u, v)
    iterations += 1
  return (iterations, u, v)

res = gradient(1, 1, 0.1, 10**(-14))
print(f'Iterations: {res[0]}\nu: {res[1]}\nv: {res[2]}')
```

```
Iterations: 10
u: 0.04473629039778207
v: 0.023958714099141746
```

7. (a) 10^{-1}

```
def gradient2(u, v, lr, max_iterations):
  iterations = 0
  while(iterations < max_iterations):
    du = partial_u(u, v)
    u = u - du * lr
    dv = partial_v(u, v)
    v = v - dv * lr
    iterations += 1
  error2 = E(u, v)
  return (error2)

error2 = gradient2(1, 1, 0.1, 15)
print(f'Error after 15 iterations: {error2}')
```

```
Error after 15 iterations: 0.13981379199615315
```

Logistic Regression

8. (d) 0.100

9. (a) 350

```
import random as rnd
import numpy as np

def gen_line():
    [x1,x2,y1,y2] = [rnd.uniform(-1.0, 1.0), rnd.uniform(-1.0, 1.0), rnd.uniform(-1.0, 1.0), rnd.uniform(-1.0, 1.0)]
    w = np.array([x2*y1-y2*x1, y2-y1, x1-x2])
    return w, [x1,x2,y1,y2]

def gen_pts(n, d, w=None):
    if w is None:
        w, li = gen_line()

    d_ = np.random.uniform(-1.0, 1.0,(d,n))
    x_ = np.append(np.ones(n), d_).reshape((d+1,n))
    y = np.sign(np.dot(w.T,x_))
    d_ = np.append(x_, y).reshape((d+2,n))

    return w, d_

def compute_gradient(w, x_n, y_n):
    return -y_n*x_n/(1+np.exp(y_n*np.dot(w.T,x_n)))

def update(w, d_, eta, rand_perm):
    for n in rand_perm:
        x_n = np.array([d_[0][n], d_[1][n], d_[2][n]])
        y_n = np.array([d_[3][n]])

        v_t = -compute_gradient(w, x_n, y_n)
        w = w + eta * v_t

    return w

def calc_error(w, d_):
    return np.sum(np.log((1+np.exp(-d_[3]*np.dot(w.T,d_[0:3])))))/ len(d_[0])

eta = 0.01

E_out_list = []
epoch_list = []

for i in range(100):
    w, d_ = gen_pts(100,2)
    _, d_test = gen_pts(5000, 2, w=w)
    w_init = np.array([0.0, 0.0, 0.0])
    w_prior = w_init
```

```
        w_prior = w_init
        rand_perm = np.random.permutation(len(d_[0]))
        w_ = update(w_prior, d_, eta, rand_perm)
        epoch = 1

        while np.linalg.norm(w_prior - w_) >= 0.01:
            w_prior = w_
            rand_perm = np.random.permutation(len(d_[0]))
            w_ = update(w_prior, d_, eta, rand_perm)
            epoch += 1

        E_out_list.append(calc_error(w_, d_test))
        epoch_list.append(epoch)

print('E_out: ', np.sum(E_out_list)/len(E_out_list))
print('epoch: ', np.sum(epoch_list)/len(epoch_list))
```

```
E_out:  0.10241205879527879
epoch:  346.96
```

PLA as SGD

10. (e) -min($0, y_n$**w^Tx_n**)

SGD and PLA both reduce the error based on an individual point at a time. To simulate PLA, we want the gradient of $e_n(w)$ to be 0 when classified correctly and $-yw^Tx$ when classified incorrectly. Choice **[e]** shows this.