1. $E_D[E_{in}(w_{in})] = \sigma^2\left(1 - \frac{d+1}{N}\right)$

for $\sigma = 0.1, d = 8$ we want

$$\sigma^2\left(1 - \frac{d+1}{N}\right) > 0.008$$

$$0.1^2\left(1 - \frac{9}{N}\right) > 0.008$$

$$1 - \frac{9}{N} > \frac{0.008}{0.1^2}$$

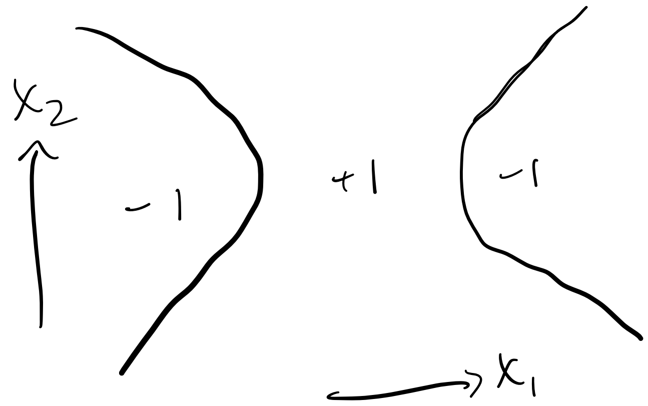
$$1 - \frac{0.008}{0.1^2} > \frac{9}{N}$$

$$\frac{N}{9} > 5$$

$$N > 45$$

$\Rightarrow$ smallest answer choice $> 45 \Rightarrow 100 \Rightarrow \boxed{[c]}$

2. $\Phi(1, x_1, x_2) = (1, x_1^2, x_2^2)$ weights: $[w_0, w_1, w_2]$



$\Phi(1, x_1, x_2) = w_0 + w_1 x_1^2 + w_2 x_2^2$

$y = w_0 + w_1 x_1^2 + w_2 x_2^2$

↳ we want two hyperbolas that are outwards-facing

we know that a hyperbola has the form

$ax^2 + by^2 = 1$ $(x_1 = x,\ x_2 = y,\ x_0 = 1$ in this case$)$

where $a > 0$ and $b < 0$

options a and b eliminate one of the variables by setting one weight to 0, which is just a parabola.

c: $\tilde{w}_1 > 0,\ \tilde{w}_2 > 0 \rightarrow$ this results in an elliptical shape

d: $\tilde{w}_1 < 0,\ \tilde{w}_2 > 0 \rightarrow$ this results in a vertically-oriented hyperbola

e: $\tilde{w}_1 > 0,\ \tilde{w}_2 < 0 \rightarrow$ this is a horizontally-oriented parabola $\Rightarrow$ $\boxed{[e]}$

3. smallest value that is at least the vc-dimension of a linear model in the transformed space

From slide 4 in lecture 9:

$d_{vc} \leq \tilde{d}+1$ for non-linear transformed space with $\tilde{d}$ points

$\Phi_4 : x$ has 14 terms $\Rightarrow \tilde{d} = 15 \Rightarrow d_{vc} \leq 16$

so, the vc-dimension of this transformed space is at most 16 $\Rightarrow$ 20 is the smallest $\boxed{[d]}$

4. $E(u,v) = (ue^{v} - 2ve^{-u})^2$

use power rule + chain rule

$$\frac{dE}{du} = 2(ue^{v}-2ve^{-u})(e^{v}+2ve^{-u}) \Rightarrow \boxed{[e]}$$

$$\frac{dE}{dv} = 2(ue^{v}-2ve^{-u}) \cdot (ue^{v}-2e^{-u})$$

5.

```python
import random
import math

def error(u, v):
  return (u*np.exp(v) - 2*v*np.exp(-u))**2

def grad_u(u, v):
  return 2*(u*np.exp(v) - 2*v*np.exp(-u))*(np.exp(v) + 2*v*np.exp(-u))

def grad_v(u, v):
  return 2*(u*np.exp(v) - 2*v*np.exp(-u))*(u*np.exp(v) - 2*np.exp(-u))

def gd(u, v):
  epochs = 0
  error_val = error(u, v)
  while error_val > 10**(-14):
    epochs += 1
    u_new = u - 0.1*grad_u(u, v)
    v_new = v - 0.1*grad_v(u, v)
    u = u_new
    v = v_new
    error_val = error(u, v)

  return u, v, epochs

u, v, epochs = gd(1.0, 1.0)
print(u)
print(v)
print(epochs)
```

```
0.04473629039778207
0.023958714099141746
10
```

Based on my code above, it takes about 10 iterations for the error to go below 10^{-14} => [d]

6.

Problem 6:

```python
[16] final = np.array([u, v])
     option1 = np.array([1.0, 1.0])
     option2 = np.array([0.713, 0.045])
     option3 = np.array([0.016, 0.112])
     option4 = np.array([-0.083, 0.029])
     option5 = np.array([0.045, 0.024])
     print(np.linalg.norm(final - option1))
     print(np.linalg.norm(final - option2))
     print(np.linalg.norm(final - option3))
     print(np.linalg.norm(final - option4))
     print(np.linalg.norm(final - option5))
```

```
1.365717886924672
0.6685948857743971
0.0926123232021653
0.12783573228217807
0.0002669218610597792
```

Based on my code above, the closest option to my final answers for u and v is (0.045, 0.024) => [e]

7.

Problem 7:

```
def cd(u, v):
  epochs = 0
  error_val = error(u, v)
  while epochs < 30:
    if epochs % 2 == 0:
      u_new = u - 0.1*grad_u(u, v)
      u = u_new
    else:
      v_new = v - 0.1*grad_v(u, v)
      v = v_new
    error_val = error(u, v)
    epochs += 1
  return error_val

print(cd(1.0, 1.0))
```

```
0.13981379199615315
```

According to my code above, the error after 30 iterations is 0.14, closest to 0.1 => $\boxed{[a]}$

8. based on my code below, $E_{out} = 0.1 \Rightarrow \boxed{[d]}$

9. based on my code below, average epochs ≈ 344, $\Rightarrow \boxed{[a]}$

```python
import numpy as np
import random
import math

def classify_point(point, m, b):
    x, y = point
    if y > m * x + b:
      return 1
    return -1

def rand_func():
  x1 = random.uniform(-1.0, 1.0)
  x2 = random.uniform(-1.0, 1.0)
  y1 = random.uniform(-1.0, 1.0)
  y2 = random.uniform(-1.0, 1.0)
  m = (y2 - y1) / (x2 - x1)
  b = y1 - m * x1
  return m, b

def create_points(m, b, num):
  X = []
  Y = []
  X = np.random.uniform(-1.0, 1.0, (num, 2))
  Y = np.array([classify_point(point, m, b) for point in X])
  return X, Y
```

```python
def lg_sgd(X, Y):
  epochs = 0
  weights = np.array([0.0, 0.0, 0.0])
  difference = 1000
  while difference >= 0.01:
    epochs += 1
    e_in = np.array([0.0, 0.0, 0.0])
    indices = list(range(X.shape[0]))
    random.shuffle(indices)
    for i in indices:
      x = X[i]
      y = Y[i]
      dot_product = np.dot(weights, x)
      result = (y * x) / (1 + np.exp(y * dot_product))
      e_in -= result
    new_weights = weights - 0.01*e_in
    difference = np.linalg.norm(weights - new_weights)
    weights = new_weights

  return weights, epochs

def calc_error(weights, x_test, y_test):
  error_total = 0
  for i in range(len(x_test)):
    dot = -y_test[i]*np.dot(weights, x_test[i])
    error_total += math.log(1 + math.exp(dot))
  return error_total / len(x_test)
```

```python
runs = 100
total_epochs = 0
total_error = 0
for _ in range(runs):
  m, b = rand_func()
  X, Y = create_points(m, b, 100)
  X_with_bias = np.hstack((np.ones((X.shape[0], 1)), X))
  weights, epochs = lg_sgd(X_with_bias, Y)
  total_epochs += epochs
  x_test, y_test = create_points(m, b, 1000)
  X_test_bias = np.hstack((np.ones((x_test.shape[0], 1)), x_test))
  total_error += calc_error(weights, X_test_bias, y_test)

print("average e_out: ", total_error/100)
print("average epochs: ", total_epochs/100)
```

```
average e_out:  0.10384674061775277
average epochs:  343.91
```

10. **PLA** is for binary classification

the error function is for misclassified points, so we want $e_n(w) = 0$ when $y_n \cdot (w \cdot x_n) > 0$ because $w \cdot x_n$ is the predicted classification (± 1) and if the actual classification (y_n) has the same sign as $w \cdot x_n$, then their product will be positive.

[a] $e^{-y_n w^T x_n}$ → if $y_n (w^T x_n) > 0$, then we have a very small number that is not 0, so this error function would contribute 'error' even when y_n and $w^T x_n$ agree.

[b] $-y_n w^T x_n$ → The actual result of the disagreement calculation will be +1 or -1, so it will also count error when y_n and $w^T x_n$ agree.

[c] $(y_n - w^T x_n)^2$ → This will always be positive unless $y_n = w^T x_n$, but we want disagreement to result in -1 for the PLA.

[d] $\ln(1 + e^{-y_n w^T x_n})$ → This function also outputs a probability (between 0 and 1)

rather than a binary classification that is required for the PLA.

[e] - $\min(0, y_n w^T x_n)$ → This error function would allow PLA to be implemented as SGD because it returns 0 when y_n and $w^T x_n$ agree, but -1 otherwise.

final answer: [e]