Kunnithi Treecharoensomboon ID 63070503461

CPE383 Machine Learning: Quiz 4

1. The following points (xi, yi) are discrete samples from a function f(x) = ax3 + bx2 + cx + d.

   1.a Show the update rule equation used to find the current a, b, c, and d after each iteration. Make sure you show the mathematics on how this is derived.

cost function

$$J(a,b,c,d) = 1/2m\,(\Sigma(f(x_i) - y_i)^2$$

update coeffiennt a, b, c, d α is learning rate

$$a \leftarrow a - \alpha \times \frac{\partial J}{\partial a}$$

$$b \leftarrow b - \alpha \times \frac{\partial J}{\partial b}$$

$$c \leftarrow c - \alpha \times \frac{\partial J}{\partial c}$$

$$d \leftarrow d - \alpha \times \frac{\partial J}{\partial d}$$

compute partial derivative use chain rules

$$\frac{\partial J}{\partial a} = 1/m \times \Sigma(f(x_i) - y_i)\ \frac{x_i\,\partial f}{\partial a}$$

$$\frac{\partial J}{\partial b} = 1/m \times \Sigma(f(x_i) - y_i)\ \frac{x_i\,\partial f}{\partial b}$$

$$\frac{\partial J}{\partial c} = 1/m \times \Sigma(f(x_i) - y_i)\ \frac{x_i\,\partial f}{\partial c}$$

$$\frac{\partial J}{\partial d} = 1/m \times \Sigma(f(x_i) - y_i)\ \frac{x_i\,\partial f}{\partial d}$$

compute partial derivative use power

$$\frac{x_i\,\partial f}{\partial a} = 3a\,x_i^2$$

$$\frac{x_i\,\partial f}{\partial b} = 2b\,x_i$$

$$\frac{x_i\,\partial f}{\partial c} = x_i$$

$$\frac{x_i\,\partial f}{\partial d} = 1$$

Ans update rule equation :

$$a \leftarrow a - \alpha/m\ (\Sigma(f(x_i) - y_i) \times 3x_i^2$$

$$b \leftarrow b - \alpha/m\ (\Sigma(f(x_i) - y_i) \times 2x_i$$

$$c \leftarrow c - \alpha/m\ (\Sigma(f(x_i) - y_i) \times x_i$$

$$d \leftarrow d - \alpha/m\ (\Sigma(f(x_i) - y_i) \times 1$$

1.b Write a program to find the best fit a, b, c, and d using gradient descent. You must write the gradient descent loop yourself and not use any gradient descent libraries. Attach the source code as well. Hint: You should get a, b, c, and d close to 0.5, 5.3, -2.7, and 3.5, respectively.

```
import numpy as np

X = np.array([-6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6])
Y = np.array([103, 87, 67, 46, 26, 11, 4, 7, 23, 57, 110, 185, 286])

m = len(X)

alpha = 0.00001

a = 0
b = 0
c = 0
d = 0

num_iter = 500000

for i in range(num_iter):
    y_pred = a*X**3 + b*X**2 + c*X + d

    J = (1/(2*m)) * np.sum((y_pred - Y)**2)

    da = (1/m) * np.sum((y_pred - Y) * X**3)
    db = (1/m) * np.sum((y_pred - Y) * X**2)
    dc = (1/m) * np.sum((y_pred - Y) * X)
    dd = (1/m) * np.sum(y_pred - Y)

    a = a - alpha * da
    b = b - alpha * db
    c = c - alpha * dc
    d = d - alpha * dd

print("Final values of a, b, c, and d:")
print("a =", a)
print("b =", b)
print("c =", c)
print("d =", d)
```

```
Final values of a, b, c, and d:
a = 0.49650152287848437
b = 5.3145183904367705
c = -2.6158258552827576
d = 3.27488351646328
```

2. Redo Problem 1b, but use the numerical method to calculate all your partial derivatives, where h is a very small number.

```python
import numpy as np

def E(a, b, c, d, x, y):
    sum = 0.
    for i in range(x.size):
        sum += (y[i] - (a * x[i]**3 + b * x[i]**2 + c * x[i] + d))**2
        return sum

def numerical_gradient(a, b, c, d, x, y, h):
    grad_a = (E(a + h, b, c, d, x, y) - E(a - h, b, c, d, x, y)) / (2 * h)
    grad_b = (E(a, b + h, c, d, x, y) - E(a, b - h, c, d, x, y)) / (2 * h)
    grad_c = (E(a, b, c + h, d, x, y) - E(a, b, c - h, d, x, y)) / (2 * h)
    grad_d = (E(a, b, c, d + h, x, y) - E(a, b, c, d - h, x, y)) / (2 * h)
    return grad_a, grad_b, grad_c, grad_d

x = np.array([-6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6])
y = np.array([103, 87, 67, 46, 26, 11, 4, 7, 23, 57, 110, 185, 286])

a, b, c, d = np.random.random(4) - 0.5

step_size = 0.00001
threshold = 0.000001
max_iterations = 1000000
h = 0.000001

for i in range(max_iterations):
    grad = numerical_gradient(a, b, c, d, x, y, h)
    a -= step_size * grad[0]
    b -= step_size * grad[1]
    c -= step_size * grad[2]
    d -= step_size * grad[3]
    E_now = E(a, b, c, d, x, y)
```

```python
print("Best fit:")
print("a =", a)
print("b =", b)
print("c =", c)
print("d =", d)
```

```
Best fit:
a = -0.38500906349213526
b = 0.5884027207305087
c = 0.1565816589616667
d = -0.4052749373464046
```

3. Solve Problem 1b using Pseudo-Inverse Linear Regression to find (a, b, c, d). You can use numpy or other tools to invert matrices.

```
import numpy as np

def find_best_fit(x, y):
    ones = np.ones(len(x))
    X = np.vstack([x**3, x**2, x, ones]).T
    X_pinv = np.linalg.pinv(X)
    a, b, c, d = np.dot(X_pinv, y)
    return a, b, c, d

# Example usage:
x = np.array([-6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6])
y = np.array([103, 87, 67, 46, 26, 11, 4, 7, 23, 57, 110, 185, 286])

a, b, c, d = find_best_fit(x, y)
print("Best fit:")
print("a =", a)
print("b =", b)
print("c =", c)
print("d =", d)
```

```
Best fit:
a = 0.49650349650349607
b = 5.299200799200799
c = -2.61588411588411
d = 3.657342657342658
```

4. Solve Problem 1b using the Gauss-Newton method to find (a, b, c, d). You can use numpy or other tools to invert matrices in each iteration.

```
import numpy as np

def f(x, a, b, c, d):
    return a * x**3 + b * x**2 + c * x + d

def find_best_fit(x, y, a=1, b=1, c=1, d=1, max_iter=100, tol=1e-6):
    for i in range(max_iter):
        J = np.column_stack([x**3, x**2, x, np.ones_like(x)])
        r = y - f(x, a, b, c, d)
        delta = np.linalg.inv(J.T @ J) @ J.T @ r
        a, b, c, d = a + delta[0], b + delta[1], c + delta[2], d + delta[3]
        if np.abs(delta).max() < tol:
            break
    return a, b, c, d

# Example usage:
x = np.array([-6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6])
y = np.array([103, 87, 67, 46, 26, 11, 4, 7, 23, 57, 110, 185, 286])

a, b, c, d = find_best_fit(x, y)

print("Best fit:")
print("a =", a)
print("b =", b)
print("c =", c)
print("d =", d)
```

```
Best fit:
a = 0.49650349650349646
b = 5.2992007992008
c = -2.6158841158841155
d = 3.6573426573426575
```