นาย สัณหณัฐ พรมจรรย์ 63070501069

Quiz 4

1. The following points (x_i, y_i) are discrete samples from a function $f(x) = ax^3 + bx^2 + cx + d$.

   a. *5 points.* 0.5 hrs. 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.

$$f(x) = ax^3 + bx^2 + cx + d$$

Update rule: $E^{t+1} - E^{t} = -\left(\frac{\partial E}{\partial a}\right) - \left(\frac{\partial E}{\partial b}\right)^2 - \left(\frac{\partial E}{\partial c}\right)^2 - \left(\frac{\partial E}{\partial d}\right)^2$

จาก (x_i, y_i) เป็น simple point จะได้ผลรวมกำลังสองผลต่าง

$$E = (ax_i^3 + bx_i^2 + cx_i + d - y_i)^2$$

ทำการ minimize E โดยใช้อนุพันธ์ย่อยของ E เทียบแต่ละค่าสัมประสิทธิ์

$$\frac{\partial E}{\partial a} = 2(x_i^3)(ax_i^3 + bx_i^2 + cx_i + d - y_i) = 2\Sigma(x_i^3)(ax_i^3 + bx_i^2 + cx_i + d - y_i)$$

$$\frac{\partial E}{\partial b} = 2(x_i^2)(ax_i^3 + bx_i^2 + cx_i + d - y_i) = 2\Sigma(x_i^2)(ax_i^3 + bx_i^2 + cx_i + d - y_i)$$

$$\frac{\partial E}{\partial c} = 2(x_i)(ax_i^3 + bx_i^2 + cx_i + d - y_i) = 2\Sigma(x_i)(ax_i^3 + bx_i^2 + cx_i + d - y_i)$$

$$\frac{\partial E}{\partial d} = 2(ax_i^3 + bx_i^2 + cx_i + d - y_i) = 2\Sigma(ax_i^3 + bx_i^2 + cx_i + d - y_i)$$

จาก Update rule จะได้

$$a^{t+1} = a^t - \alpha\Sigma(ax_i^6 + bx_i^5 + cx_i^4 + dx_i^3 - x_i^3 y_i) \quad \#$$

$$b^{t+1} = b^t - \alpha\Sigma(ax_i^5 + bx_i^4 + cx_i^3 + dx_i^2 - x_i^2 y_i) \quad \#$$

$$c^{t+1} = c^t - \alpha\Sigma(ax_i^4 + bx_i^3 + cx_i^2 + dx_i - x_i y_i) \quad \#$$

$$d^{t+1} = d^t - \alpha\Sigma(ax_i^3 + bx_i^2 + cx_i + d - y_i) \quad \#$$

α คือ step size - จะมีขนาดเล็กพอ

b. *15 points*. 2 hrs. 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.

x_i	y_i
-6	103
-5	87
-4	67
-3	46
-2	26
-1	11
0	4
1	7
2	23
3	57
4	110
5	185
6	286

[11]
```
# Clause 1.b
import numpy as np

# Define the sample data
data = [(-6, 103), (-5, 87), (-4, 67), (-3, 46), (-2, 26), (-1, 11),
        (0, 4), (1, 7), (2, 23), (3, 57), (4, 110), (5, 185), (6, 286)]

# Define the initial values of a, b, c, and d
a, b, c, d = 1, 1, 1, 1

# Define the learning rate
alpha = 0.000001

# Define the number of iterations
num_iterations = 1000000

# Define the gradient descent loop
for i in range(num_iterations):
    grad_a = grad_b = grad_c = grad_d = 0
    for x, y in data:
        error = (a * x ** 3 + b * x ** 2 + c * x + d) - y
        grad_a += error * x ** 3
        grad_b += error * x ** 2
        grad_c += error * x
        grad_d += error
    a -= alpha * grad_a
    b -= alpha * grad_b
    c -= alpha * grad_c
    d -= alpha * grad_d

# Print the final values of a, b, c, and d
print("a =%5.2f" % a, ", b =%5.2f"% b, ", c =%5.2f" %c, ", d =%5.2f" %d)
```

```
a = 0.50 , b = 5.30 , c =-2.62 , d = 3.66
```

2. *10 points.* 1 hrs. Redo Problem 1b, but use the numerical method to calculate all your partial derivatives, where *h* is a very small number.

$$\frac{\partial}{\partial x_i} f(x_1, \ldots, x_i, \ldots, x_n) = \frac{f(x_1, \ldots, x_i + h, \ldots, x_n) - f(x_1, \ldots, x_i - h, \ldots, x_n)}{2h}$$

[19]

```
# Clause 2
import numpy as np

# Define the sample data
data = [(-6, 103), (-5, 87), (-4, 67), (-3, 46), (-2, 26), (-1, 11),
        (0, 4), (1, 7), (2, 23), (3, 57), (4, 110), (5, 185), (6, 286)]

# Define the initial values of a, b, c, and d
a, b, c, d = 1, 1, 1, 1

# Define the learning rate
alpha = 0.000001

# Define the number of iterations
num_iterations = 1000000

# Define the small number h for numerical differentiation
h = 2.83e-8

# Define the gradient descent loop
for i in range(num_iterations):
    grad_a = grad_b = grad_c = grad_d = 0
    for x, y in data:
        error = (a * x ** 3 + b * x ** 2 + c * x + d) - y
        grad_a += error * (1/h) * ((a+h) * x ** 3 + b * x ** 2 + c * x + d - (a-h) * x ** 3 - b * x ** 2 - c * x - d)
        grad_b += error * (1/h) * (a * x ** 3 + (b+h) * x ** 2 + c * x + d - a * x ** 3 - (b-h) * x ** 2 - c * x - d)
        grad_c += error * (1/h) * (a * x ** 3 + b * x ** 2 + (c+h) * x + d - a * x ** 3 - b * x ** 2 - (c-h) * x - d)
        grad_d += error * (1/h) * (a * x ** 3 + b * x ** 2 + c * x + (d+h) - a * x ** 3 - b * x ** 2 - c * x - (d-h))
    a -= alpha * grad_a
    b -= alpha * grad_b
    c -= alpha * grad_c
    d -= alpha * grad_d

# Print the final values of a, b, c, and d
print("a =%5.2f" % a, ", b =%5.2f"% b, ", c =%5.2f" %c, ", d =%5.2f" %d)
```

```
a = 0.50 , b = 5.30 , c =-2.62 , d = 3.66
```

3. *10 points*. 1 hrs. Solve Problem 1b using Pseudo-Inverse Linear Regression to find (a, b, c, d). You can use numpy or other tools to invert matrices.

[15]

```
# Clause 3
import numpy as np

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

# Create the design matrix X
X = np.column_stack((data[:, 0] ** 3, data[:, 0] ** 2, data[:, 0], np.ones(len(data))))

# Create the target vector y
y = data[:, 1]

# Calculate the pseudo-inverse of X
X_pinv = np.linalg.pinv(X)

# Calculate the coefficients a, b, c, and d
a, b, c, d = np.dot(X_pinv, y)

# Print the values of a, b, c, and d
print("a =%5.2f" % a, ", b =%5.2f"% b, ", c =%5.2f" %c, ", d =%5.2f" %d)
```

```
a = 0.50 , b = 5.30 , c =-2.62 , d = 3.66
```

4. *10 points*. 1 hrs. 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.

$$X^{t+1} = X^t - \alpha J^{\#1} r(X^t) = X^t - \alpha (J^T J)^{-1} J^T r(X^t);\ \alpha = 1 \text{ works for linear case.}$$

[16]

```
# Clause 4
import numpy as np

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

# Define the initial values of a, b, c, and d
a, b, c, d = 1, 1, 1, 1

# Define the Gauss-Newton loop
for i in range(10):
    # Compute the Jacobian matrix
    J = np.array([[x**3, x**2, x, 1] for x, y in data])

    # Compute the residual vector
    r = np.array([y - a*x**3 - b*x**2 - c*x - d for x, y in data])

    # Compute the Gauss-Newton step
    step = np.linalg.inv(J.T.dot(J)).dot(J.T).dot(r)

    # Update the parameters
    a += step[0]
    b += step[1]
    c += step[2]
    d += step[3]

# Print the final values of a, b, c, and d
print("a =%5.2f" % a, ", b =%5.2f"% b, ", c =%5.2f" %c, ", d =%5.2f" %d)
```

```
a = 0.50 , b = 5.30 , c =-2.62 , d = 3.66
```

Link Clause 1.b, 2, 3, 4: https://colab.research.google.com/drive/1n-pdl_aTb1ClhOXRzqbys7T0ZzmbwdF_?usp=sharing