SINCE 1947
SEO KYEONG
UNIVERSITY

[지능 시스템 3차 과제]

과목명	지능 시스템
담당 교수님	조용완 교수님

전공	컴퓨터공학과
학번	2018305068
이름	정동기

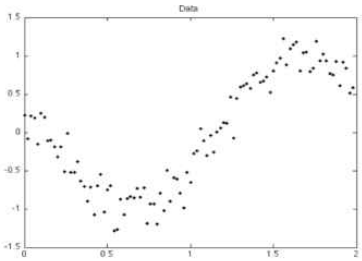
제출일	2022.4.23

과제 3: 경사하강법을 이용한 근사함수

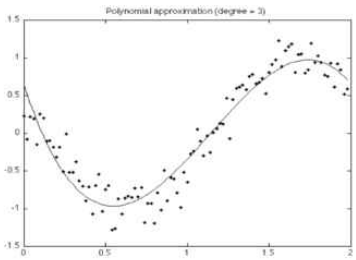
다음 그림과 같이 주어진 데이터에 대해 오차를 최소화하는 근사함수(3차 다항함수, 파라미터 (a, b, c, d))를 경사하강법으로 구하는 예제를 프로그램을 작성하여 실습하고 결과를 정리하여(학습의 중간 과정 포함, 예를 들어, 10 epoch학습 결과, 100 epoch학습 결과 등등) 보고서 형태로 제출한다.

데이터 쌍 (x_i, y_i)은 300개를 다음과 같이 생성하여 사용한다.

$[0, 3]$ 구간의 x_i를 랜덤하게 발생시켜 $y = x^3 - 4.5x^2 + 6x + 2$의 함수에 대입하여 y를 구하고 여기에 $[-0.5, 0.5]$ 사이의 값을 랜덤하게 생성하여 더한 값을 y_i로 한다. (4월 24일까지 Google Classroom 제출)



(x_i, y_i) for $i = 1, 2, \cdots, N$



$y = ax^3 + bx^2 + cx + d$

위 문제는 무수한 점들을 하나의 함수로 근사화하는 문제이다. 그렇다면 함수로 어떻게 표현해야 할까. 데이터를 가장 비슷하게 표현하기 위해 우린 삼차함수의 계수를 구해야 한다. 계수에 따라 함수의 모양이 바뀌기 때문이다. 이러한 계수를 구하기 위해선 기울기를 이용한다. 기울기가 0에 가까울수록 최소값이 된다.

$$J = \frac{1}{2N}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

계수를 구하기 위해 사용하는 식이다. 한 x에서의 y값과 추정식에서 나온 y값의 차를 제곱한 값들의 합의 평균이다. 이 값을 목적함수라고 한다. 이후 저 식을 풀어보면 차수가 하나 더 높은 식이 나온다.

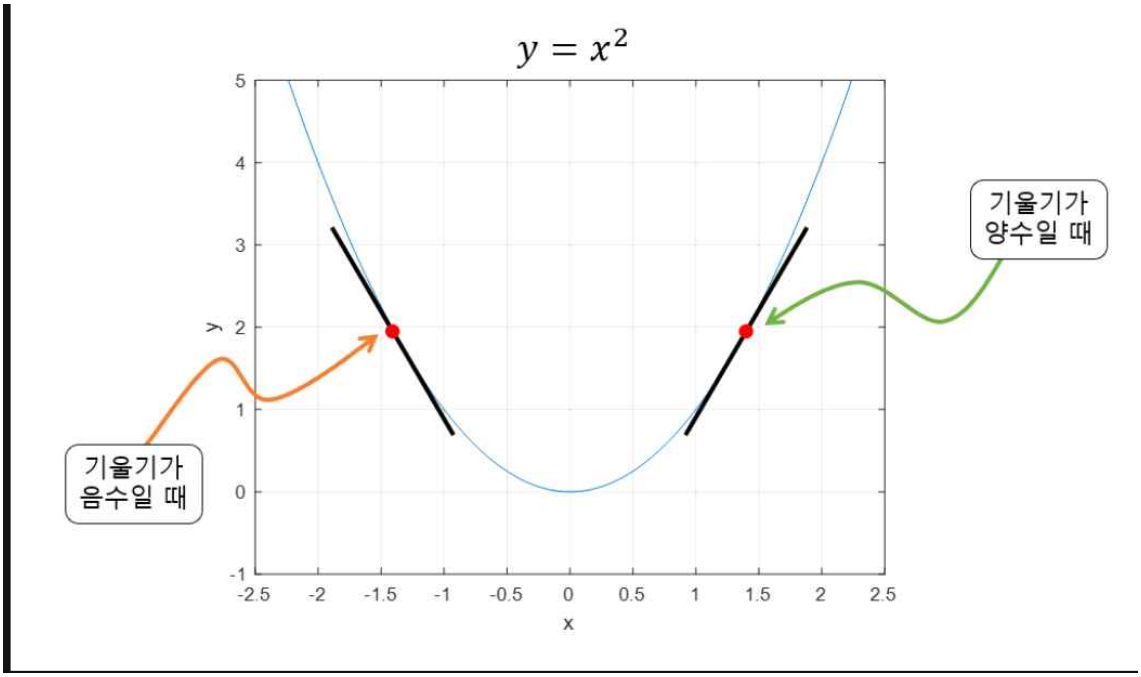
$$\frac{\partial J}{\partial m} = \frac{1}{N}\sum_{i=1}^{N}\left(-x_i(y_i - \hat{y}_i)\right)$$

$$= \frac{1}{N}\sum_{i=1}^{N}\left(-x_i(y_i - (mx_i + b))\right)$$

그렇게 나온 식을 편미분 한다. 미분값은 기울기이므로 가장 작은 곳으로 가면 최솟값을 구할 수 있다. 그렇다면 처음 구한 값에서 가장 작은 기울기 값으로 가기 위해 어떻게 해야 할까.

$$[m\ b]_{k+1}=[m\ b]_k-\alpha\left[\frac{\partial J}{\partial m}\ \ \frac{\partial J}{\partial b}\right]$$

위의 식을 이용하는 것이다. 이 식을 이용하기에 앞서 기울기가 양수냐 음수냐의 따라 움직이는 방향이 바뀌게 된다, 기울기가 양수라면 음수쪽으로 가야되고 기울기가 음수라면 양수 쪽으로가야된다. 그렇기 때문에 위의 식에 -(마이너스) 값이 붙게 되었고 그저 방향만 정해준다고 움직일 수 없으니 움직일 수 있는 최소거리 a를 곱해 준 것이다.



이렇게 최솟값으로 가게 되면 최적의 근사함수가 나오게 된다.

이제 코드를 짜보겠다.
코드를 간단히 설명하자면

1. 경사하강법을 이용할 수 있는 코드를 짠다.
2. 데이터쌍 xi,yi를 만든다.
3. 경사하강법을 이용하여 구한 게수를 이용하여 근사함수를 만든다,

이다. 1번에 경사하강법을 이용할 수 있는 코드를 짜는 것은 6장 ppt에 나와있는 코드를 참고하여 3차 근사함수를 구할 수 있게 변형하였다. 2번에 있는 데이터쌍을 구하기위해 random함수를 이용하였고 여기서 random함수의 범위가 [0,1)이 되어 오른쪽이 열린구간이 되었지만 둘 다 닫힌구간을 구하는 방법을 몰라 random함수를 사용하였다. 마지막 3번은 matplotlib를 이용하여 찍어 보였다.

코드

```python
import matplotlib.pyplot as plt
import random
import numpy as np

# 1. 경사하강법, 에폭의 크기와 러닝 메이트의 크기를 늘려 좀 더 빠르고 많이 돌 수 있도록 하였다.
def linear_regression (X, y, a_current=0, b_current=0, c_current=0, d_current=0, epochs=100000, learning_rate=0.005):
    N = float(len(y))
    for i in range(epochs):
        y_current = (a_current * (X**3)) + (b_current * (X**2)) + (c_current * X) + d_current
        cost = sum([data**2 for data in (y-y_current)]) / (2*N)
        a_gradient = -(1 / N) * sum((X**3) * (y - y_current))
        b_gradient = -(1 / N) * sum((X**2) * (y - y_current))
        c_gradient = -(1 / N) * sum(X * (y - y_current))
        d_gradient = -(1 / N) * sum(y - y_current)
        a_current = a_current - (learning_rate * a_gradient)
        b_current = b_current - (learning_rate * b_gradient)
        c_current = c_current - (learning_rate * c_gradient)
        d_current = d_current - (learning_rate * d_gradient)
        if i % 10000 == 0:
            print(i, "번째 : a:", a_gradient, "b:", b_gradient, "c:", c_gradient, "d:", d_gradient)
    return a_current, b_current, c_current, d_current, cost

xi = []
yi = []

#데이터 쌍 (xi,yi) 생성
for i in range(300):
    x = random.uniform(0, 3)
    xi.append(x)
    y = (x**3 - 4.5*(x**2) + 6*x + 2) + (random.uniform(-0.5, 0.5))
    yi.append(y)

#list가 목록이여서 거듭제곱이 안되므로 array로 바꾸어줌
xi_array = np.array(xi)
yi_array = np.array(yi)
a, b, c, d, cost = linear_regression(xi_array, yi_array)
x = np.linspace(0, 3, 300)
plt.xlabel("xi_array")
plt.ylabel("yi_array")
plt.plot(xi, yi, '.')
plt.plot(x, a*x**3 + b*x**2 + c*x + d, 'r-')
plt.show()
```

결과

에폭을 십만 번 돌렸기 때문에 만 번째마다 학습 결과값을 찍어주었다.

```
0 번째 : a: -31.67500925811505 b: -13.783320830266515 c: -6.671336790619038 d: -4.175005822324369
10000 번째 : a: -0.0032505239318594507 b: 0.015134587703922904 c: -0.0191053887000819 d: 0.005005896744574851
20000 번째 : a: -0.0025139666680221652 b: 0.011817187903068536 c: -0.015168773390611198 d: 0.004147569102211877
30000 번째 : a: -0.0019868227237243496 b: 0.00933938915786236 c: -0.01198844832045579 d: 0.003278131907355181
40000 번째 : a: -0.0015702521376435672 b: 0.007381230239565291 c: -0.009474870106205279 d: 0.002590816994271513
50000 번째 : a: -0.0012410225705948164 b: 0.005833632132403626 c: -0.007488305459835694 d: 0.0020476089727158105
60000 번째 : a: -0.0009808214769140104 b: 0.004610513796790106 c: -0.005918257245840217 d: 0.0016182935784903145
70000 번째 : a: -0.0007751758850779915 b: 0.003643842633194208 4 c: -0.004677395842862719 d: 0.0012789913216243055
80000 번째 : a: -0.0006126473236567661 b: 0.002879850212075383 c: -0.0036967017420907757 d: 0.0010108294456170816
90000 번째 : a: -0.0004841955876203982 b: 0.002276041552516428 c: -0.0029216265266156995 d: 0.0007988921823376168
```

아래는 plt를 이용하여 찍어 본 그래프 모양이다.