Homework 1 Data Science

Dustin O'Brien

September 14, 2024

Problem 1

Consider the following function,

$$f(\beta) = 3\beta^2 - 5\beta + 4$$
$$f'(\beta) = 6\beta - 5$$
$$f''(\beta) = 6$$

Since $f''(x) > 0$ when $x \in (-\infty, \infty)$ we know the following function is always concave up and thus must contain an absolute minimum. Using Fermat's theorem we know local extememum of $f(x)$ can only occur at $f'(x) = 0$

$$f'(\beta) = 6\beta - 5$$
$$6\beta - 5 = 0$$
$$6\beta = 5$$
$$\beta = \frac{5}{6}$$

Therefore the only possible minimum value is $f(\frac{5}{6})$

Problem 2

$$\mathbf{W}_0 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \qquad \beta_0 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$\nabla \mathbf{W}_0 = \begin{bmatrix} 0.2 & -0.3 \\ -0.1 & 0.2 \end{bmatrix} \qquad \nabla\ \beta_0 = \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix} \qquad \alpha = .01$$

Using Gradient decent algorithm we know

$$x_{n+1} = x_n - \alpha f'(x)$$

Thus,

$$\beta_1 = \beta_0 - \alpha \nabla\ \beta_0$$

$$\mathbf{W}_1 = \mathbf{W}_0 - \alpha \nabla \mathbf{W}_0$$

$$\beta_1 = \begin{bmatrix} 1 \\ -2 \end{bmatrix} - .01 * \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix}$$

$$\beta_1 = \begin{bmatrix} 1 \\ -2 \end{bmatrix} + \begin{bmatrix} -0.001 \\ 0.002 \end{bmatrix}$$

$$\beta_1 = \begin{bmatrix} .999 \\ -1.998 \end{bmatrix}$$

$$\mathbf{W}_1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} - .01 \begin{bmatrix} 0.2 & -0.3 \\ -0.1 & 0.2 \end{bmatrix}$$

$$\mathbf{W}_1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} -0.002 & 0.003 \\ 0.001 & -0.002 \end{bmatrix}$$

$$\mathbf{W}_1 = \begin{bmatrix} .998 & 2.003 \\ 3.001 & 3.998 \end{bmatrix}$$

Therefore,

$$\mathbf{W}_1 = \begin{bmatrix} .998 & 2.003 \\ 3.001 & 3.998 \end{bmatrix}, \qquad \beta_1 = \begin{bmatrix} .999 \\ -1.998 \end{bmatrix}$$

Problem 3

CODE:

```
#Gets The Value of The Derivative of f(x)
def derivf(x):
    return 6 * x - 5

#The loops that incrementally gets next value of f(x)
def gradientDecent(x=0, learningRate = .01, numIter = 1000):
    x = 0 #Simple Starting Value
    for i in range(numIter):
        x = x - learningRate * derivf(x) #Formula for gradient Decent

    return x


'''
#The one Liner
def gradientDecent(x=0,learningRate = .01, numIter = 10):
    return x if numIter == 0 else gradientDecent(x, learningRate, numIter - 1) - learningRa
```

```
'''

#Output with actual answer for comparison
print("Approx.: ", gradientDecent(x=0,learningRate=.01, numIter=10000))
print("Real Answer: ", 5/6)
```

Part 1: The code works by looping through the decent equation a defined number of iterations using a passed learning rate and start at the location x given and gives real minimum to compare to

Part 2: I chose a value of $\alpha = .01$ and numIter $= 10000$ these values were chosen as they gave nearly instantaeous results and accuracy so high that it was more accuate than real answer approximation

Part 3: I got an approximation of 0.8333333333333333 which is highly accurate

Part 4: I nailed it the only difference between approximation and 5 / 6 by computer was the latter had a rounding error

```
    import numpy as np

def gradientDecent(learningRate = .01, x = np.array([ 0 , 0 , 0 , 0 , 0 ]), iterations = 100
    X = np.array([[1,1,1,1,1],[1,2,4,8,16],[1,3,9,27,81],[1,4,16,64,256],[1,5,25,125,625]])
    y = np.array([5,31,121,341,781])
    XT = X.T

    for i in range(iterations):
        x = x - learningRate * (XT @ ((X @ x) - y))
    return x


print(gradientDecent(learningRate = .000001, x = np.array([0,0,0,0,0]), iterations = 1500000
```

Part 1:

$$x_{1500000} = \begin{bmatrix} .982 \\ 1.01 \\ 1.01 \\ 1.00 \\ 1.00 \end{bmatrix}$$

Part 2:

$$\alpha = .000001$$

Part 3: This would be highly beneficial in polynomial regression as the vector can outputted vector can represent coefficients of the polynomial and therefore minimize cost of a polynomial function thus regressing it.