CS156 (Introduction to AI), Spring 2021

Homework 1 submission

Roster Name: Neeval Kumar

Preferred Name (if different): Chosen Name

Student ID: 011877086

Email address: kumar.neeval@gmail.com

Any special notes or anything you would like to communicate to me about this homework submission goes in here.

References and sources

List all your references and sources here. This includes all sites/discussion boards/blogs/posts/etc. where you grabbed some code examples.

https://towardsdatascience.com/implement-gradient-descent-in-python-9b93ed7108d1

Solution

Gradient descent works by imagining a space on a mountain and you would like to work yourself down the mountain. Let's make it so x represents our place on the mountain. We will randomly generate an x between -100, 100 (this works well with our iterations).

In [9]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
from sympy import *
```

In [10]:
```
np.random.seed(1)
random_num = np.random.randint(-100,100)
print(f'Our chosen x will be {random_num}.')
```

```
Our chosen x will be -63.
```

To reach the minimum, or the bottom of the mountain, we need to use a learning rate, which will represent our steps taken down the mountain. For this assignment, we will use a learning rate of .1

The number of iterations will help us optimize our algorithm to reach the lowest height. We will use 100 iterations.

In [11]:
```
learning_rate = .1
num_iterations = 100
```

The algorithm will work by finding derivative and take a step in the opposite of the gradient (in the negative direction)

Stepping in the direction of the gradient will compute the highest ascent, so by doing the opposite, we are finding the min

In [12]:
```
def gradientdescent(random_num, learning_rate, num_iterations, y):
    # First find the derivative
    dydx = diff(y, x)

    # Make position equal to our random number, which represents our place on the hil
    position = random_num

    # Go through iterations and step through
    for i in range(num_iterations):
        previous_position = position
        position = position - learning_rate * dydx.evalf(subs= {x:previous_position})

    # At the last iteration, print the global minimum
        if i + 1 == num_iterations:
            print(f"\nGlobal Minimum is at x = {position}")
```

Driver code

In [8]:
```
x = symbols('x')
y = 3 * (x ** 2) + (2 * x) - 4
gradientdescent(random_num, learning_rate, num_iterations, y)
```

```
Global Minimum is at x = -0.333333333333333
```