CS4061: Week 1 Assignment

Conor McCauley - 17323203

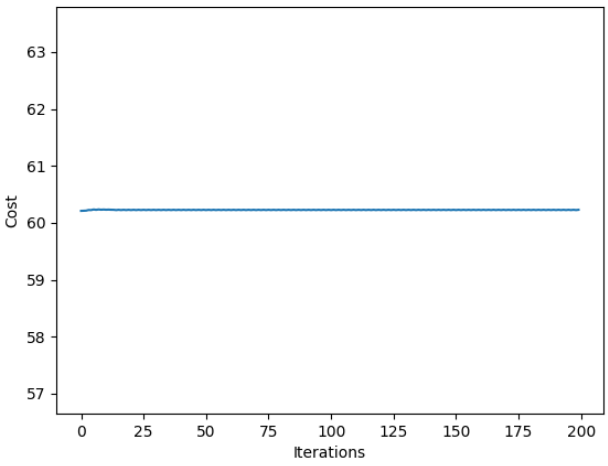
October 12, 2020

Question (a)

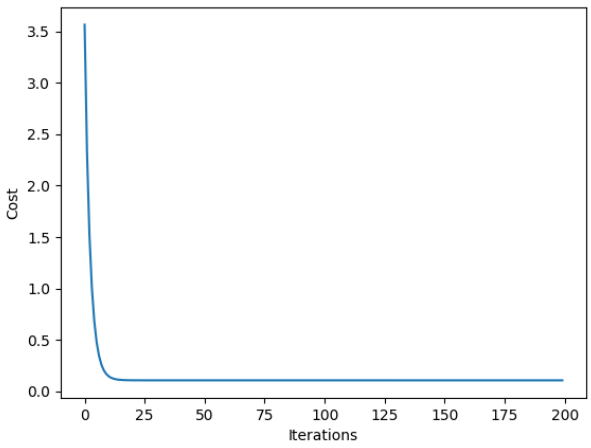
Dataset Identifier: `# id:5-2358-35`
See the appendix for code.

Question (b)

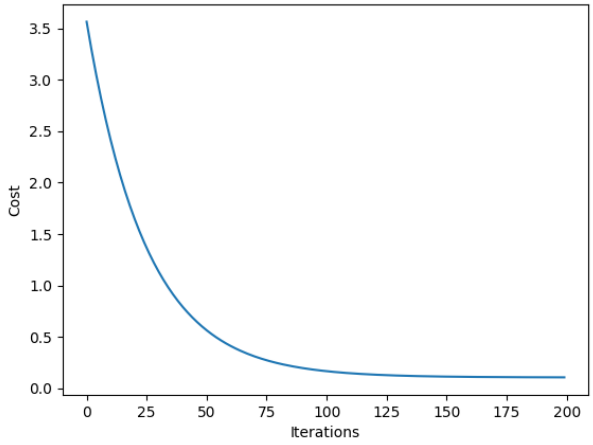
(i) The `part_i()` method in the code runs 200 iterations of the gradient descent algorithm for a number of different values of $\alpha = \{1.0, 0.1, 0.01, 0.001\}$. Both the input and output variables are normalised by subtracting the mean, μ, from each data point and dividing the result by the standard deviation, σ. The plotted values of $J(\theta)$ are shown below:



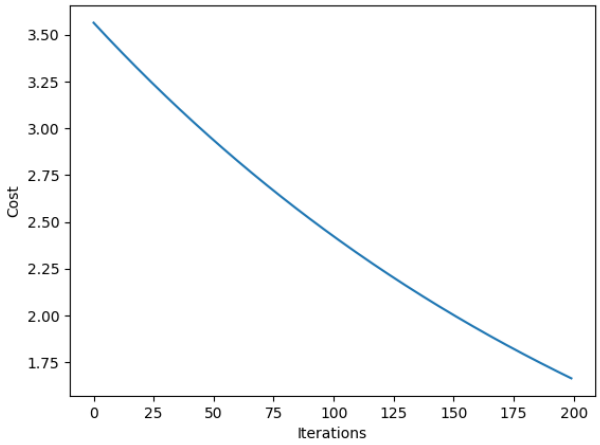
$\alpha = 1.0$



$\alpha = 0.1$



$\alpha = 0.01$

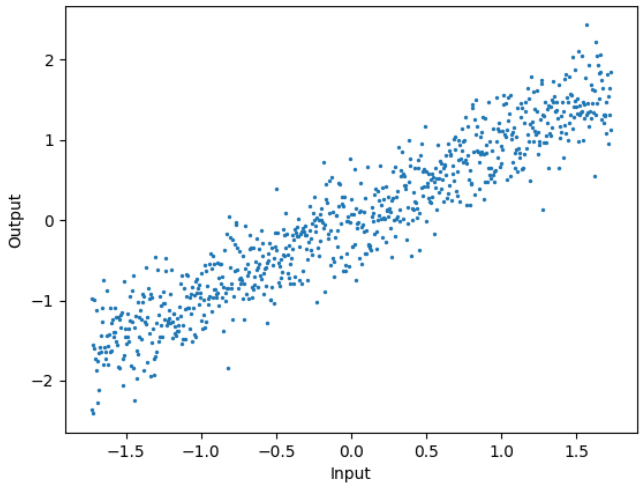


$\alpha = 0.001$

It is evident from the above plots that $\alpha = 0.1$ is the optimal learning rate as it arrives at suitable parameter values in fewer than 15 iterations. The plot for $\alpha = 1.0$ shows that this learning rate is much too high as no convergence will ever occur. The two remaining plots take too long to converge on optimal parameter values (with $\alpha = 0.001$ not even converging at all after 200 iterations).
(ii) The parameter values, θ_0, θ_1, for each value of α are as follows:

α	θ_0	θ_1
1.0	0.608	-0.812
0.1	0.000	0.945
0.01	0.011	0.914
0.001	0.407	-0.232

These values are consistent with the plotted values of $J(\theta)$ from the previous question: $\alpha = 1.0$ does not converge at all while neither $\alpha = 0.01$ or $\alpha = 0.001$ converge on the optimal parameter values fast enough. Once again, $\alpha = 0.1$ appears to be the optimal choice in this case.
(iii) The cost function at the final iteration for $\alpha = 0.1$ (the optimal learning rate) is 0.107. Using the `part_iii()` method in the code we can produce a scatter plot containing our normalised input and output variables. This results in the following graph:



Since we want predict a constant value, v, we will need to select a model of the form $y = 0x + v$. We will try to select v such that there are an equal number of points above and below the line $y = v$. Since our normalised data has a mean of 0 we know that the line $y = 0$ will produce the best result (this is also intuitively evident from the above graph). Running the cost function, $J(\theta)$ with 0 for both of our parameter values returns the following result: 1.000. This is significantly larger than the final cost of our trained model as is to be expected.
(iv) The `part_iv()` method in the code uses SciKit's `StandardScaler` to normalise the input and output variables and `LinearRegression` to train the model. This model results in an intercept (θ_0) of 0.000 and a coefficient (θ_1) of 0.945. Theses results are identical to those found in (i) for $\alpha = 0.1$ which indicates, once again, that this is the optimal learning rate out of those that were tested.

Appendix: Code

```
from random import uniform
from sklearn import linear_model, preprocessing
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

def normalise(data):
    # normalise using the mean and standard deviation
    mu = sum(data) / len(data)
    sigma = (sum(pow(d - mu, 2) for d in data) / len(data)) ** 0.5
    return list(map(lambda d: (d - mu) / sigma, data))

def h(theta_0, theta_1, x):
    return theta_0 + (theta_1 * x)

def J(theta_0, theta_1, X, Y):
    M = len(X)
    return sum(pow(h(theta_0, theta_1, X[i]) - Y[i], 2) for i in range(M)) / M

def gradient_descent(X, Y, alpha, theta_0, theta_1):
    M, ITERATIONS = len(X), 200
    costs = [None] * ITERATIONS
    for i in range(ITERATIONS):
        costs[i] = J(theta_0, theta_1, X, Y)
        theta_0 += -2 * alpha / M * sum(h(theta_0, theta_1, X[i]) - Y[i]
```

```
            for i in range(M))
        theta_1 += -2 * alpha / M * sum((h(theta_0, theta_1, X[i]) - Y[i]) * X[
            for i in range(M))
    return costs, theta_0, theta_1

def part_i(data):
    X = normalise([d[0] for d in data])
    Y = normalise([d[1] for d in data])
    # randomly select starting parameter values
    start_theta_0, start_theta_1 = uniform(-1, 1), uniform(-1, 1)
    for alpha in [1.0, 0.1, 0.01, 0.001]:
        costs, theta_0, theta_1 = gradient_descent(X, Y, alpha,
            start_theta_0, start_theta_1)
        print(
            'alpha = %.3f >> theta_0 = %.8f, theta_1 = %.8f, cost = %.8f' %
            (alpha, theta_0, theta_1, costs[-1])
        )
        plt.plot(costs)
        plt.xlabel('Iterations')
        plt.ylabel('Cost')
        plt.show()

def part_iii(data):
    X = normalise([d[0] for d in data])
    Y = normalise([d[1] for d in data])
    plt.scatter(X, Y, s=2)
    plt.xlabel('Input')
    plt.ylabel('Output')
    plt.show()

def part_iv(data):
    X = np.array([d[0] for d in data]).reshape(-1, 1)
    X = preprocessing.StandardScaler().fit_transform(X)
    Y = np.array([d[1] for d in data]).reshape(-1, 1)
    Y = preprocessing.StandardScaler().fit_transform(Y)
    model = linear_model.LinearRegression().fit(X, Y)
    print(
        'theta_0 = %.8f, theta_1 = %.8f' %
        (model.intercept_, model.coef_)
    )

data = pd.read_csv('dataset.csv', comment='#').values.tolist()
part_i(data)
part_iii(data)
part_iv(data)
```