ASSIGNMENT 1

Name: Michael Black
Student Number: 221402063
Date: 14/08/2023

I declare that this is my own, original work.

Signature: 

IMPLEMENTATION DETAILS

Stopping conditions: 5000 iterations
Initial weights range: -0.5 to 0.5
Training set size: 350
Test set size: 150
Values for η investigated: 0.000001 to 0.000000001
Activation function: y = x

RESULTS

Number of iterations (typically): 30000 iterations
Weights found (state which input is associated with each weight):
City is one hot encoded:
Durban: [1,0,0,0]
Cape Town: [0,1,0,0]
Johannesburg: [0,0,1,0]
Gqeberha: [0,0,0,1]
BIAS_WEIGHT = 112.07429904

City	City2	City3	City4	Sqr M	Nr Rooms	Nr Parking	Pool?	Nr Baths	Age	Business	Commute
-2,69745276	127,36347842	35,4644232	-53,86636716	5,61478759	55,87683519	89,66073854	-7.74155937	47,61606726	-2,66605556	-19,47582452	-2,02307653

Best η value: 0.000001
Sum Squared Error (SSE) on Training Set: 2300000
Average Price Difference on Training Set: R64000
SSE on Test Set: 1500000
Average Price Difference on Test Set: R75000

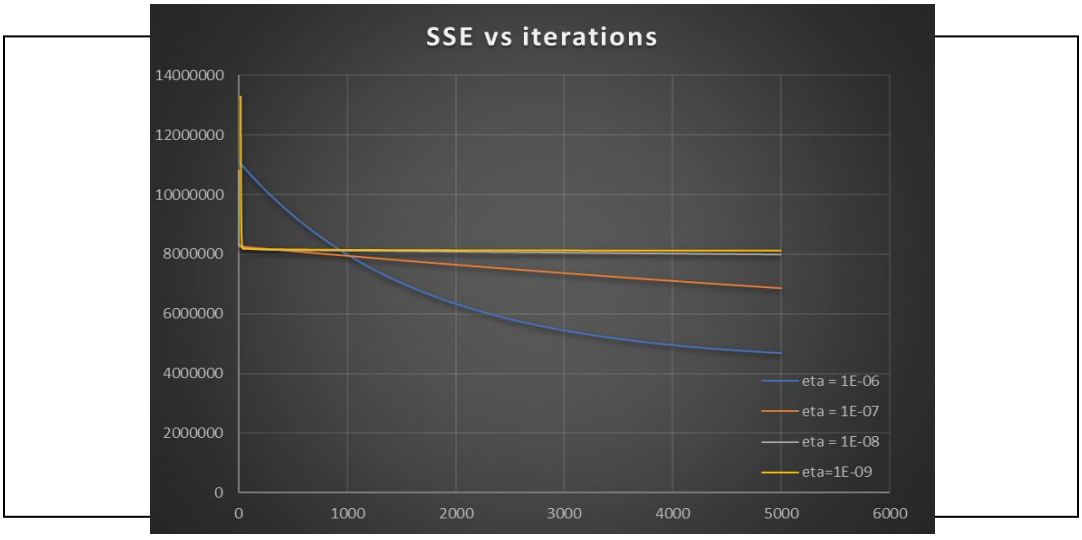

Figure 1: SSE vs iterations for various values of η

PREDICTIONS FOR PROPERTIES IN DEMO.CSV

1. 592	2. 1166	3. 1063	4. 677	5. 4313
6. 4626	7. 5454	8. 5200	9. 1924	10. 979
11. 3437	12. 3604	13. 1456	14. 367	15. 5731
16. 1059	17. 2785	18. 3784	19. 616	20. 4729
21. 1768	22. 4671	23. 2481	24. 810	25. 278
26. 3470	27. 1858	28. 3869	29. 4806	30. 5208
31. 2491	32. 3539	33. 3787	34. 1531	35. 4005
36. 1136	37. 3191	38. 2748	39. 1888	40. 850
41. 3599	42. 3449	43. 471	44. 2077	45. 2382
46. 5400	47. 3338	48. 2509	49. 1084	50. 3345

OBSERVATIONS

When training single neuron, a larger value for the learning rate reduced the error more than what a smaller step size would in this case. The larger the initial step size was, the less the SSE was after 5000 iterations. The SSE converges after 5000 iterations for the range of eta values that were investigated. The smaller eta values converged much sooner but had higher error than the largest value investigated which was 0.000001. With more iterations the smaller eta values would possibly give more accurate result but at a cost of more time.

Making the initial weight close to zero also yielded a better error than larger initial values, however this only occurred in some instances which means that it might not have affected it at all. Values larger than 0.000001 were not able to be calculated by the program. With 5000 iterations, including the bias made an error reduction of R5000.

```
import numpy as np #Used for creating and manipulating arrays easily
import csv #Used for reading and writing to csv files

cities = [] #Initialize array, array will be converted to numpy array
after it is populated
data = np.loadtxt('Properties.csv', delimiter=',', dtype=str) #Read data
from Properties file and load into numpy array for type string, headers
are manually removed

for row in range(500):
    for col in range(8): # manual one hot encoding of values in properties
        if(data[row,col] == 'Durban'):
            cities.append([1,0,0,0])
        if(data[row,col] == 'Cape Town'):
            cities.append([0,1,0,0])
        if(data[row,col] == 'Johannesburg'):
            cities.append([0,0,1,0])
        if(data[row,col] == 'Gqeberha'):
            cities.append([0,0,0,1])
        if(data[row, col] == 'No'):
            data[row, col] = 0
        elif(data[row,col] == 'Yes'):
            data[row, col] = 1

cities2 = np.array(cities) #Convert cities to numpy array
array1 = data[np.arange(0,500)][:, np.arange(1,10)] #Create new array
without cities column
result_data = np.hstack((cities2, array1)) #add one hot encoded cities to
result data
result_data = result_data.astype(float) #Convert array data type to float

#Training Data
training_inputs = result_data[np.arange(0,350)][:, np.arange(0,12)]
training_outputs = result_data[np.arange(0,350)][:, 12]

#Testing Data
test_inputs = result_data[np.arange(350, 500)][:, np.arange(0,12)]
test_outputs =result_data[np.arange(350, 500)][:, 12]

#Demo Data
cities = []#Reset cities
data = np.loadtxt('Demo.csv', delimiter=',', dtype=str) #Load data from
Demo.csv file
for row in range(50):
```

```
    for col in range(8): #Manual one hot encoding
        if(data[row,col] == 'Durban'):
            cities.append([1,0,0,0])
        if(data[row,col] == 'Cape Town'):
            cities.append([0,1,0,0])
        if(data[row,col] == 'Johannesburg'):
            cities.append([0,0,1,0])
        if(data[row,col] == 'Gqeberha'):
            cities.append([0,0,0,1])
        if(data[row, col] == 'No'):
            data[row, col] = 0
        elif(data[row,col] == 'Yes'):
            data[row, col] = 1

cities2 = np.array(cities) #Convert cities to numpy array
array1 = data[np.arange(0,50)][:, np.arange(1,9)] #Create new array
without cities column
result_data = np.hstack((cities2,array1)) #add one hot encoded cities to
result data
result_data = result_data.astype(float) #Convert data to float

#Demo inputs
demo_inputs = result_data[np.arange(0,50)][:, np.arange(0,12)]

weights = (np.random.rand(12) - 0.5)/2 #Initialize weights to values
between -0.25 and 0.25
bias_weight = np.random.rand(1)/4 #Initialize bias weight to value between
0 and 0.25
print("Weights before training: ")
print(weights)

learning_rate = 0.000001 #Optimal value

#training loop
Squared_error = [] #Create array for squared error, this dat is written to
excel and used for data visualization
error = 0
sse = 0

for j in range(30000): # Iterates 30000 times
    print(j)
    for i in range(350):
        output = np.dot(weights, training_inputs[i]) + bias_weight*(1)
#Calculate the output
        error = training_outputs[i] - output # Calculate the error
```

```
        bias_adjust = -learning_rate*2*error*(1)
        bias_weight = bias_weight - bias_adjust #Update the bias weight
        for k in range(12):
            adjust = -learning_rate*2*error*(training_inputs[i,k])
            weights[k] = weights[k] - adjust #Update input weights
        sse += error*error #Calculate SSE
    Squared_error.append(sse) #Add SSE to array
    sse = 0  #Reset the SSE
SQ = np.array(Squared_error) #Convert to a numpy array
SQ = SQ.astype(int) #Convert datatype to integer

print("Weights after training: ", weights)
print("The bias weight is: ", bias_weight)

with open('Outputs.csv', mode='w', newline='') as file: # Write data to
the CSV file
    writer = csv.writer(file)
    writer.writerows(SQ)

Squared_error = [] #Reset array
training_errors = [] #Reset array
for l in range(349):
    output = np.dot(weights, training_inputs[l]) + bias_weight
    error = abs(training_outputs[l] - output)
    training_errors.append(error)
    Squared_error.append(error*error)

#SSE on training
print("Training set SSE: ", sum(Squared_error))
#Average error on training values
avg = sum(training_errors)/len(training_errors)*1000
print("The average price difference for the training set is R", avg)


errors = []
Squared_error = []
for l in range(144):
    output = np.dot(weights, test_inputs[l]) + bias_weight
    error = abs(test_outputs[l] - output)
    errors.append(error)
    Squared_error.append(error*error)

#SSE on test set
print("Test set SSE: ", sum(Squared_error))
```

```
#Average calculation on test set
average = (sum(errors)/len(errors)*1000)
print("The average price difference for the test set is R", average)



for i in range(50): # Calculate demo outputs
    output = np.dot(weights, demo_inputs[i]) + bias_weight
    print(output)
```