***Assignment 1***

Name: James Shepherd

Student Number: 217257968

Date: 01 August 2021

I declare that this is my own, original work.



Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Implementation Details*

Stopping Conditions: SSE <= 250

Initial Weights Initialization Range: [-1, 1] excluding 0

Training Set size: 60

Test Set size: 10

Values for η investigated: 0.001, 0.00001, 0.000005, 0.000001

Activation function: linear

Learning Rule: wi(t+1) = wi(t) - η(-2(tp − yp)zi,p

*Results*

Number of iterations (typically): 85

Weights found:

0.5941135176275564

0.14808877761519015

-0.08230670065046464

0.5550307061730382

Best η value: 0.00001

Sum Squared Error (SSE) on Training Set: 250

SSE on Test Set: 1626

**Note**: The results for the first η value were much higher than the others. Plotting them on the same graph made the other values indistinguishable so I used two graphs.

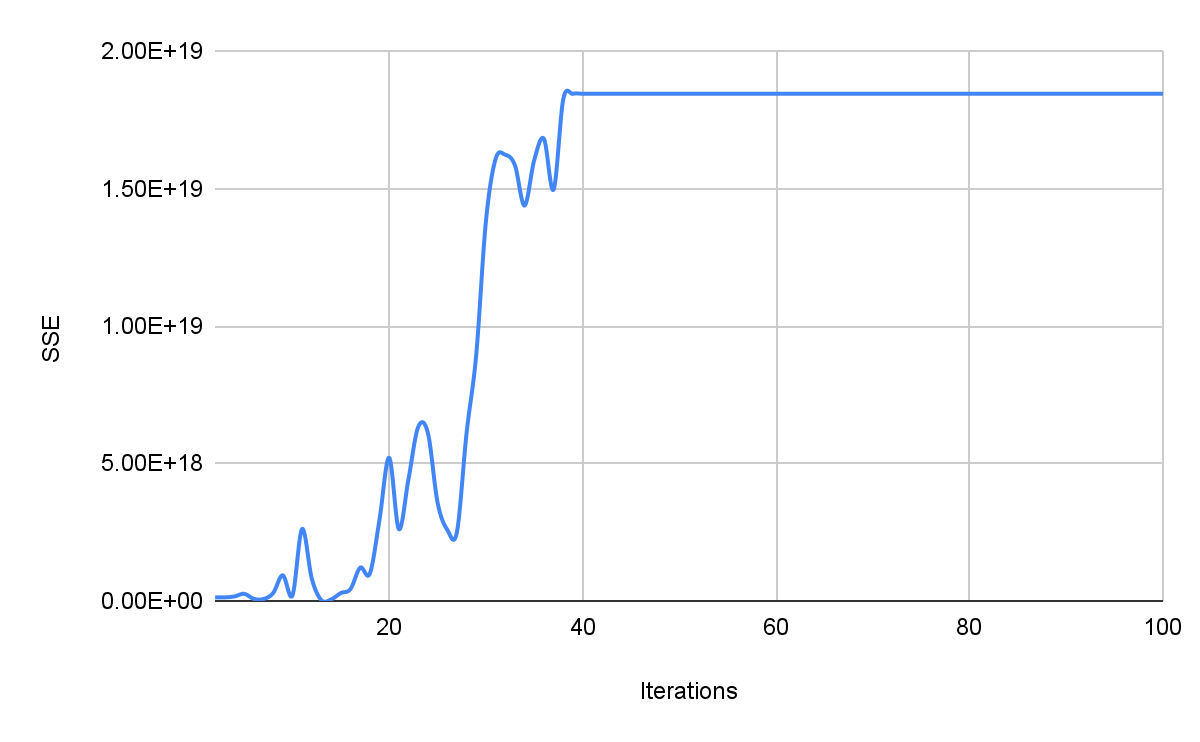


Figure 1: SSE vs iterations for η = 0.001

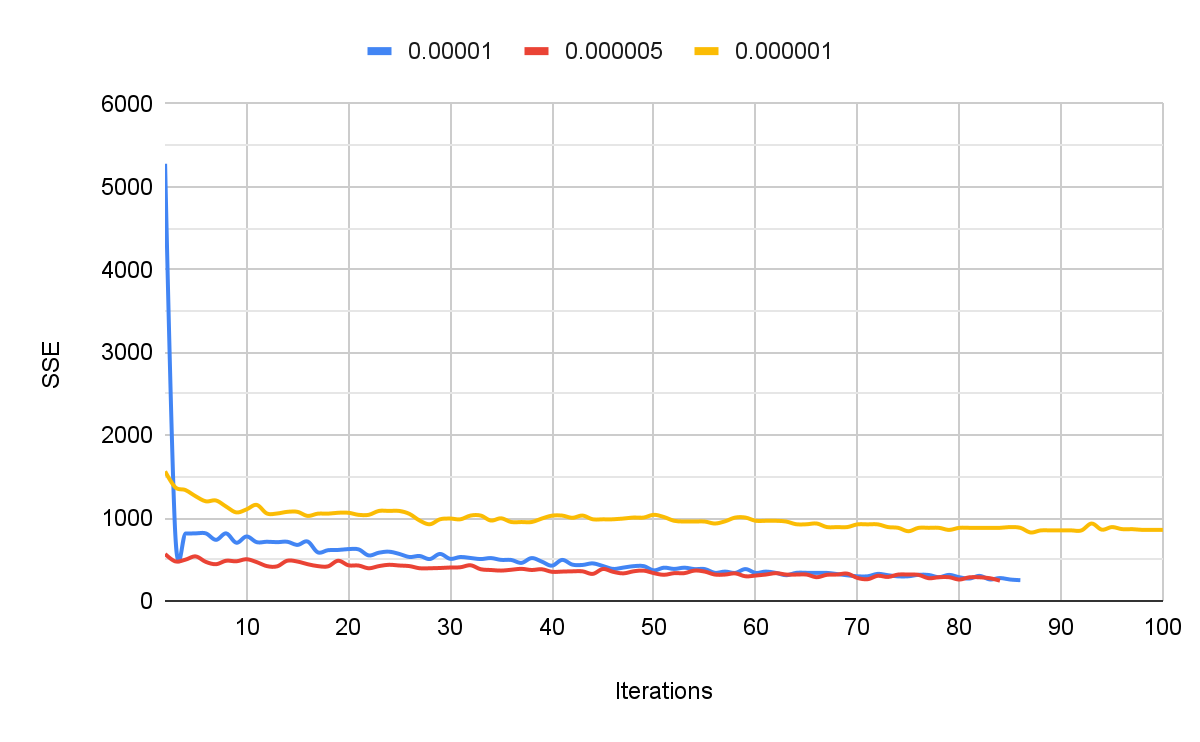


Figure 2: SSE vs iterations for various values of η

*Evaluation*

Performance on evaluation data:

| Mark 1: | 64 |
| --- | --- |
| Mark 2: | 52 |
| Mark 3: | 68 |
| Mark 4: | 35 |
| Mark 5: | 58 |
| Mark 6: | 64 |
| Mark 7: | 39 |
| Mark 8: | 80 |
| Mark 9: | 31 |
| Mark 10: | 70 |

*Observations*

I found that the training for a single neuron is not very complex and doesn’t take very long to run (at least for the sample sizes we used).

The stopping criteria and η value both play a big role in the success and efficiency of the algorithm. The η value in particular can be responsible for vastly inaccurate results if not chosen correctly. In this case the η value was much smaller than I expected and the number of iterations was also lower than I was expecting.  
Unfortunately it takes quite a bit of trial and error to narrow down which values work best. It makes sense that the more you know and understand about the problem, the faster you will be able to find these optimal values, because you will have more accurate starting estimates and you will also know what effect changing the values will have. If you don’t understand the problem in-depth then you simply have to try something, change it and see what happens.

I also found it interesting that minimizing the SSE of the training set does not necessarily minimize the SSE of the testing set and thus doesn’t produce the best results. I believe this is due to over-training the algorithm where it becomes too familiar with the training set.

I think it solved the problem fairly well because although my test data SSE was much higher than my training data SSE, the estimated marks for the test data were only about 10%-15% off on average. Considering the fact that there are so many aspects that affect a student’s mark (and we are only using three values), it would be difficult to get much more accurate than that.

That said, a neural network or simply a bigger training set would probably give better results.

public class Main {

static HashMap<Integer, ArrayList<Integer>> *patterns* = new HashMap<>();

static ArrayList<Integer> *results* = new ArrayList<>();

public static void main(String[] args) throws IOException {

BufferedReader console = new BufferedReader(

new InputStreamReader(System.*in*));

System.*out*.println("Enter training file:");

String fileName = console.readLine();

*readFile*(fileName);

Neuron neuron = new Neuron();

neuron.gradientDescent(*patterns*, *results*);

System.*out*.println("Training complete \n");

*patterns*.clear();

*results*.clear();

System.*out*.println("Enter test file:");

fileName = console.readLine();

*readFile*(fileName);

double SSE = 0;

for (int i = 0; i < *patterns*.size(); i++) {

int mark = neuron.run(*patterns*.get(i));

SSE += Math.*pow*(*results*.get(i) - mark,2);

System.*out*.println(*results*.get(i) + " - " + mark);

}

System.*out*.println("SSE = " + SSE);

*patterns*.clear();

*results*.clear();

System.*out*.println("Enter evaluation file:");

fileName = console.readLine();

*readFile*(fileName);

for (int i = 0; i < *patterns*.size(); i++) {

int mark = neuron.run(*patterns*.get(i));

System.*out*.println(mark);

}

}

public static void readFile(String fileName) {

try {

File file = new File(fileName);

Scanner reader = new Scanner(file);

int patternCount = 0;

while (reader.hasNextLine()) {

String line = reader.nextLine();

String[] patternString = line.split(",");

ArrayList<Integer> pattern = new ArrayList<>();

for (int i = 0; i < 3; i++)

pattern.add(Integer.*parseInt*(patternString[i]));

*patterns*.put(patternCount++, pattern);

if (patternString.length == 4)

try {

*results*.add(Integer.*parseInt*(patternString[3]));

} catch (Exception ignored){ }

}

reader.close();

} catch (FileNotFoundException e) {

System.*out*.println("Error reading from file.");

e.printStackTrace();

}

}

}

public class Neuron {

private ArrayList<Double> weights;

public Neuron() {}

public int run(ArrayList<Integer> pattern){

int sum = 0;

for (int i = 0; i < pattern.size(); i++)

sum += weights.get(i) \* pattern.get(i);

sum += weights.get(weights.size()-1);

return sum;

}

public void gradientDescent(HashMap<Integer, ArrayList<Integer>> patterns, ArrayList<Integer> results){

*// Initialise weights to small random values*

Random random = new Random();

weights = new ArrayList<>();

for (int i = 0; i < patterns.get(0).size(); i++)

weights.add(random.nextDouble() \* (Math.*pow*(-1,random.nextInt(2))));

weights.add(random.nextDouble() \* (Math.*pow*(-1, random.nextInt(2))));

for (double w : weights)

System.*out*.print(w + ", ");

System.*out*.println();

*// Train*

double SSE = Double.*MAX\_VALUE*;

int count = 0;

double n = 0.0001;

while (SSE > 250) {

SSE = 0;

*// for each pattern*

for (int p = 0; p < weights.size(); p++) {

ArrayList<Integer> pattern = patterns.get(p);

int expected = results.get(p);

*// adjust each weight (except bias)*

int i;

for (i = 0; i < weights.size()-1; i++)

weights.set(i, weights.get(i) - (n \* (-2 \* (expected - run(pattern)) \* pattern.get(i))));

*// adjust bias weight*

weights.set(i, weights.get(i) - (n \* (-2 \* (expected - run(pattern)))));

*// Calculate Sum Squared Error*

SSE += Math.*pow*(expected - run(pattern),2);

}

count++;

System.*out*.println(count + " iterations. SSE = " + SSE);

}

System.*out*.println("SSE = " + SSE);

for (double w : weights)

System.*out*.print(w + ", "); }}