***Assignment 2***

Name: James Shepherd

Student Number: 217257968

Date: 16 September 2021

I declare that this is my own, original work.

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

*Implementation Details*

Stopping Conditions: SSE <= 0.4

Initial Weights: (-1, 1) excluding 0

Training Set size: 26

Test Set size: 26

Values for η investigated: 1, 0.1, 0.01

Activation functions: sigmoid

Number of hidden neurons: 20

The network is presented with a vector representing a particular letter. The vector consists of 105 binary values.  
These input values are used as input for 20 hidden layer neurons, each of which applied the sigmoid activation function.The output of these 20 hidden layer neurons then constitutes the input values for 5 output layer neurons, each of which also applies the sigmoid activation function. The output of each output layer neuron is then rounded to the nearest int to ensure that a binary value is output.

The final output of the network is thus a 5 digit binary sequence (one digit from each output layer neuron) which corresponds with the letter from the original input vector.

*Results*

Number of iterations (typically): 185

Best η value: 0.1

Sum Squared Error (SSE) on Training Set: 1.5

SSE on Test Set: 17

Number correctly classified on Training Set: 25

Number correctly classified on Test Set: 7

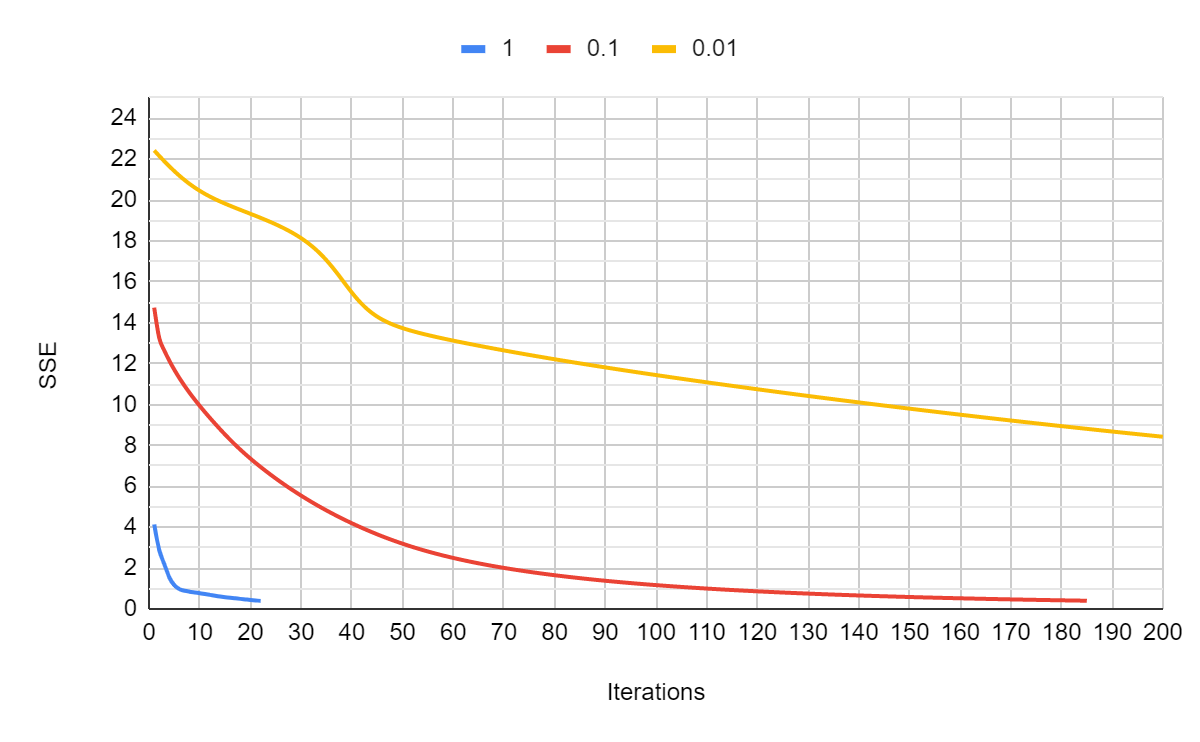


Figure 1: SSE vs iterations for various values of η

*Investigated Techniques*

Additional Techniques Investigated:

| **Technique** | **SSE on training set** | **SSE on test set** |
| --- | --- | --- |
| Noise Injection | 0.3975834094 | 0.5 |
| Appropriate Weight Initialization | 0.3989075138 | 15.5 |
| Dynamic Learning Rate | 0.3837321388 | 14.5 |

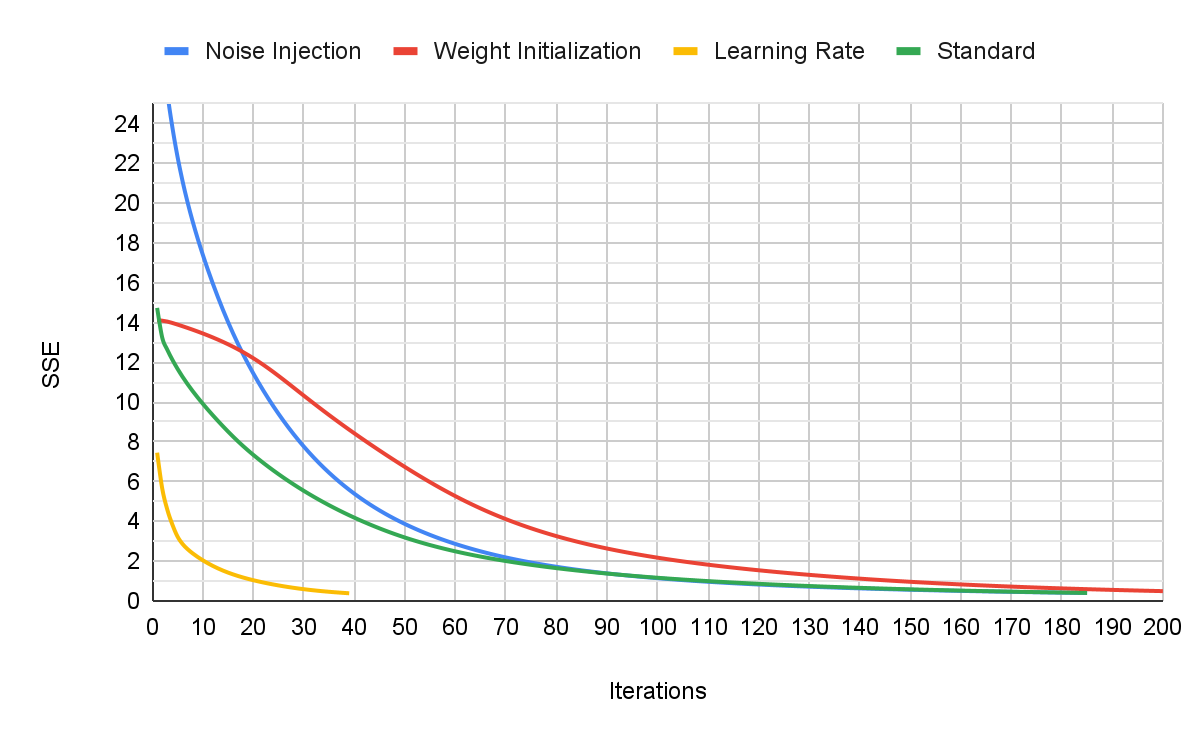


Figure 2: SSE vs iterations for the three techniques

*Observations*

Training a neural network takes quite a while because there are a lot of variables that need to be calibrated (number of hidden layer neurons, learning rate, stopping condition, tau for dynamic learning rates, etc.). Additionally, these variables can’t be calibrated individually because they all affect each other.  
I also learned that you can’t just aim for the lowest SSE in the shortest number of iterations because this doesn’t always give the best results.

Regarding the additional techniques investigated:  
I found that noise injection was extremely effective in improving the generalization ability of the network. Originally the network could easily recognize the same patterns it saw in the training set but struggled to deal with variations in the input patterns. After noise injection was implemented, the network could recognize variations much better.  
Initializing the weights differently actually had a negative effect on the network. It seems that the original weight initialization worked better. It is possible that the new weight initialization method required me to re-calibrate the other variables, however I didn’t test this. I wanted to make as few changes as possible for the comparisons. Ultimately the new weights produced the same results but had to train a little longer.  
Using a dynamic learning rate drastically reduced the training time and slightly improved the SSE on the test data. This technique does require calibrating an extra variable (tau) but, when working with bigger projects which would normally take a day or 2 to train, this method would come in very handy and calibrating the extra variable would be a small price for the benefit gained.

It is also possible that these techniques would behave differently if implemented together. For the sake of comparison, I only implemented each one separately.

Overall, I think the network solved the problem quite well. I think that, with more noise injection and several other methods working together, this network would be able to identify the letters quite accurately.

**public class** Main {

**static int** *numInputs* = 15 \* 7;

**static int** *numHiddenNeurons* = 20;

**static int** *numOutputs* = 5;

**static** HashMap<Integer, **int**[]> *patterns* = **new** HashMap<>();

**static double**[][] *v* = **new double**[*numHiddenNeurons*][*numInputs*+1];

**static double**[][] *w* = **new double**[*numOutputs*][*numHiddenNeurons*+1];

**static** HashMap<Integer, **int**[]> *results* = **new** HashMap<>();

**static** HashMap<String, **int**[]> *alpha* = **new** HashMap<>() {{

put(**"A"**, **new int**[] {0, 0, 0, 0, 1});

put(**"B"**, **new int**[] {0, 0, 0, 1, 0});

put(**"C"**, **new int**[] {0, 0, 0, 1, 1});

put(**"D"**, **new int**[] {0, 0, 1, 0, 0});

put(**"E"**, **new int**[] {0, 0, 1, 0, 1});

put(**"F"**, **new int**[] {0, 0, 1, 1, 0});

put(**"G"**, **new int**[] {0, 0, 1, 1, 1});

put(**"H"**, **new int**[] {0, 1, 0, 0, 0});

put(**"I"**, **new int**[] {0, 1, 0, 0, 1});

put(**"J"**, **new int**[] {0, 1, 0, 1, 0});

put(**"K"**, **new int**[] {0, 1, 0, 1, 1});

put(**"L"**, **new int**[] {0, 1, 1, 0, 0});

put(**"M"**, **new int**[] {0, 1, 1, 0, 1});

put(**"N"**, **new int**[] {0, 1, 1, 1, 0});

put(**"O"**, **new int**[] {0, 1, 1, 1, 1});

put(**"P"**, **new int**[] {1, 0, 0, 0, 0});

put(**"Q"**, **new int**[] {1, 0, 0, 0, 1});

put(**"R"**, **new int**[] {1, 0, 0, 1, 0});

put(**"S"**, **new int**[] {1, 0, 0, 1, 1});

put(**"T"**, **new int**[] {1, 0, 1, 0, 0});

put(**"U"**, **new int**[] {1, 0, 1, 0, 1});

put(**"V"**, **new int**[] {1, 0, 1, 1, 0});

put(**"W"**, **new int**[] {1, 0, 1, 1, 1});

put(**"X"**, **new int**[] {1, 1, 0, 0, 0});

put(**"Y"**, **new int**[] {1, 1, 0, 0, 1});

put(**"Z"**, **new int**[] {1, 1, 0, 1, 0});

}};

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

BufferedReader reader = **new** BufferedReader(

**new** InputStreamReader(System.***in***));

*// Training*

*readFile*(**"TrainingData.txt"**);

*train*();

System.***out***.println(**"Continue?"**);

reader.readLine();

*//region Validation*

*patterns*.clear();

*results*.clear();

*readFile*(**"ValidationData.txt"**);

**double** SSE = 0;

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

**double**[] o = *run*(*patterns*.get(p), **false**);

**double** E = 0;

**for** (**int** k = 0; k < o.**length**; k++)

E += Math.*pow*(*results*.get(p)[k] - o[k],2);

SSE += 0.5 \* E;

**int**[] intO = **new int**[o.**length**];

**for** (**int** x = 0; x < intO.**length**; x++)

intO[x] = (**int**)o[x];

System.***out***.println(Arrays.*toString*(*results*.get(p)) + **" - "** + Arrays.*toString*(intO));

}

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

*//endregion*

System.***out***.println(**"Continue?"**);

reader.readLine();

*//region Testing*

*patterns*.clear();

*results*.clear();

*readFile*(**"TestData.txt"**);

SSE = 0;

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

**double**[] o = *run*(*patterns*.get(p), **false**);

**double** E = 0;

**for** (**int** k = 0; k < o.**length**; k++)

E += Math.*pow*(*results*.get(p)[k] - o[k],2);

SSE += 0.5 \* E;

**int**[] intO = **new int**[o.**length**];

**for** (**int** x = 0; x < intO.**length**; x++)

intO[x] = (**int**)o[x];

System.***out***.println(Arrays.*toString*(*results*.get(p)) + **" - "** + Arrays.*toString*(intO));

}

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

*//endregion*

}

**public static void** train() {

*//region Initialise weights*

Random random = **new** Random();

**double** min = -1/Math.*sqrt*(*numInputs*);

**double** max = 1/Math.*sqrt*(*numInputs*);

*// Initialize hidden weights*

**for** (**int** x = 0; x < *v*.**length**; x++)

**for** (**int** y = 0; y < *v*[x].**length**; y++){

**do** {

*v*[x][y] = random.nextDouble() \* (Math.*pow*(-1, random.nextInt(2)));

*//v[x][y] = min + (max - min) \* random.nextDouble();*

} **while** (*v*[x][y] == 0);

}

*//Initialize output weights*

**for** (**int** x = 0; x < *w*.**length**; x++) {

**for** (**int** y = 0; y < *w*[x].**length**; y++) {

**do** {

*w*[x][y] = random.nextDouble() \* (Math.*pow*(-1, random.nextInt(2)));

*//w[x][y] = min + (max - min) \* random.nextDouble();*

} **while** (*w*[x][y] == 0);

}

}

*//endregion*

*// Train*

**double** SSE = Double.***MAX\_VALUE***;

**int** count = 0;

**double** n = 0.1;

**double** tau = 1000;

**while** (SSE > 1) {

SSE = 0;

*//double n = n0 \* Math.pow(Math.E, -count/tau); // Dynamic learning rate*

*// for each pattern*

**for** (**int** p = 0; p < *patterns*.size()-1; p++) {

**int**[] z = *patterns*.get(p);

**int**[] expected = *results*.get(p);

*// adjust each w*

**for** (**int** k = 0; k < *w*.**length**; k++)

**for** (**int** j = 0; j < *v*.**length**; j++) {

**double**[] o = *run*(z, **true**);

**double** y = *y*(z, j);

*w*[k][j] -= n \* (-2 \* (expected[k] - o[k]) \* o[k] \* (1 - o[k]) \* y);

}

*// adjust each v*

**for** (**int** j = 0; j < *v*.**length**; j++)

**for** (**int** i = 0; i < z.**length**; i++) {

**double**[] o = *run*(z, **true**);

**double** y = *y*(z, j);

**double** diff = 0;

**for** (**int** k = 0; k < *w*.**length**; k++)

diff += -2 \* (expected[k] - o[k]) \* o[k] \* (1 - o[k]) \* *w*[k][j] \* y \* (1 - y) \* z[i];

*v*[j][i] -= n \* diff;

}

*// Calculate Sum Squared Error*

**double**[] o = *run*(z, **true**);

**double** E = 0;

**for** (**int** x = 0; x < o.**length**; x++)

E += Math.*pow*(expected[x] - o[x],2);

SSE += 0.5 \* E;

}

count++;

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

}

System.***out***.println(**"Avg. SSE = "** + SSE/*patterns*.size());

}

**public static double** y(**int**[] z, **int** j) {

**double** sum = 0;

*// for each input*

**for** (**int** i = 0; i < z.**length**; i++)

sum += *v*[j][i] \* z[i];

sum = *sigmoid*(sum);

**return** sum;

}

**public static double**[] run(**int**[] z, **boolean** training) {

**double**[] o = **new double**[*numOutputs*];

*// For each output neuron*

**for** (**int** k = 0; k < o.**length**; k++) {

**double** sum = 0; *// output sum*

*// for each hidden neuron*

**for** (**int** j = 0; j < *v*.**length**; j++)

sum += *w*[k][j] \* *y*(z, j);

sum = *sigmoid*(sum);

o[k] = training ? sum : Math.*round*((**float**)sum); *// round final answers but not training because we only want 0 or 1*

}

**return** o;

}

**private static double** sigmoid(**double** x){

**return** 1 / (1 + Math.*pow*(Math.***E***, -x));

}

**public static void** readFile(String fileName) {

**try** {

File file = **new** File(fileName);

Scanner reader = **new** Scanner(file);

**int** patternCount = 0;

**while** (reader.hasNextLine()) {

*// Get result*

String line = reader.nextLine();

**char**[] c = line.toCharArray();

line = Character.*toString*(c[c.**length**-1]);

*results*.put(patternCount, *alpha*.get(line));

*// Get pattern as string*

line = reader.nextLine();

**for** (**int** i = 0; i < 14; i++)

line += **","** + reader.nextLine();

line += **",1"**; *// bias input*

*// Convert pattern to int[]*

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

**int**[] pattern = **new int**[*numInputs* + 1];

**for** (**int** i = 0; i < patternString.**length**; i++)

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

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

}

reader.close();

} **catch** (FileNotFoundException e) {

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

e.printStackTrace();

}

}

}