1506801 Submission 2

Jack Neilson

December 10, 2017

Activity 4

A

Data Structures

A 1-dimensional array of type "double" is used for both the input and output of the neural network, as it can be used to represent the vector described in the given paper.

The synaptic matrix is modelled using an Array2DRowRealMatrix, a data structure provided by the apache foundation (library included in given source file). This data structure is similar to a 2-dimensional array of type "double", however it allows for the easy selection of columns. It was chosen as it closely resembles the structure of the synaptic matrix in the paper.

Pseudocode

Training the synaptic matrix, set the position in the synaptic matrix i, j to 1 where the input at index i is 1 and the output at index j is 1.

```
train(double* input, double* output) {
	for (i = 0; i < input.length; i++) {
		for (j = 0; j < output.length; j++) {
			if (input[i] == 1 && input[j] == 1) {
				matrix[i][j] = 1;
			}
		}
	}
}
```

Testing the synaptic matrix, set the threshold value to be the lower of the two integrals of the input or output vectors then compare it to the integral of each column of the matrix to find the recalled output.

```
test(double* input, double* originalInput, int outputLength) {
	double[] output = new double[outputLength];
	double threshold;
	if (integrate(input) > integrate(originalInput){
		threshold = integrate(originalInput);
	} else {
		threshold = integrate(input);
	}
	for (i = 0; i < outputLength; i++) {
		output[i] = compare(integrate(matrix.column[i]), threshold);
	}
}
```

Integrating a vector to find the sum, used when finding threshold values and recalling output.

```
integrate(double* firing) {
	int total = 0;
	for (i = 0; i < firing.length; i++) {
		total += firing[i];
	}
}
```

Simple comparison of integrated column of synaptic matrix to threshold value to find recalled output

```
compare(double sum, double threshold) {
	if (sum >= threshold) {
		return 1;
	} else {
		return 0;
	}
}
```

Observations

In the paper provided, the neural network is observed to be tolerant of both missing and noisy input across multiple learned patterns. The implementation of the network confirms these observations, as can be seen by the testing done in the NeuralNetwork class. The correct output is produced when the network is presented with missing data, noisy data and distorted data.

B

6

The network implemented began to make mistakes in recall when 15 of the 36 neurons were strengthened, and two patterns were learned. Thus, the load parameter of the network is;

$$load = \frac{patterns\ stored}{input\ layer\ size}$$

$$load = \frac{3}{6}$$

$$load = \frac{1}{2}$$

7

I could not find an equation to calculate a maximum load parameter for a given network, only the current load parameter which is shown above.

D

Since we are providing the network input in the form of a 1-dimensional array to represent a 2-dimensional grid, combining three 2-dimensional grids in to a single input can be done simply by collating the 1-dimensional array representation of each input. It should be noted that the order in which these inputs are collated will have an effect on learning and recall. Psuedocode for this is given below:

```
collate(double* input1, double* input2, double* input3) {
	double[] retval = new double[input1.length + input2.length + input3.length];
	int pos = 0;
	for (int i = 0; i < input1.length; i++) {
		retval[pos] = input1[i];
		pos++;
	}
	for (int i = 0; i < input2.length; i++) {
		retval[pos] = input2[i];
		pos++;
	}
	for (int i = 0; i < input3.length; i++) {
		retval[pos] = input3[i];
		pos++;
	}
	return retval;
}
```

F

I could not find any techincal implementation details in the provided paper.

Compilation Instructions

The program may be compiled and ran using the standard java command line tools, adding the provided library to the classpath. For example:

java -cp .:../lib/commons-math3-3.6.1/commons-math3-3.6.1.jar NeuralNetwork

Appendix

Full Code

NeuralNetwork.java

```
public class NeuralNetwork {
	public static final double[] DEFAULT_INPUT = {0,1,0,1};
	public static final double[] DEFAULT_OUTPUT = {1,0,1,0};
	public static final double[] INCOMPLETE_INPUT = {0,0,0,1};
	public static final double[] NOISY_INPUT = {1,1,0,1};
	public static final double[] LARGE_INPUT_1 = {0,1,0,0,0,1};
	public static final double[] LARGE_INPUT_2 = {0,0,0,1,0,0};
	public static final double[] LARGE_OUTPUT_1 = {1,0,0,0,1,0};
	public static final double[] LARGE_OUTPUT_2 = {1,0,0,0,1,1};

	public static void main(String[] args) {
		//Question B.1.
		//Create a new synaptic matrix, then test with input and output of
		    ↪ size 4
		System.out.println("B.1.");
		SynapticMatrix synapticMatrix = new SynapticMatrix(DEFAULT_INPUT,
		    ↪ DEFAULT_OUTPUT);

		double[] generatedOutput = synapticMatrix.test(DEFAULT_INPUT,
		    ↪ DEFAULT_INPUT, DEFAULT_OUTPUT.length);
		System.out.println(synapticMatrix);
		System.out.println("Correct output: " + printArray(DEFAULT_OUTPUT));
		System.out.println("Generated output: " + printArray(generatedOutput))
		    ↪ ;
		System.out.println();


		//Question B.2.
		//Test the generated synaptic matrix with incomplete input
		System.out.println("B.2.");
		generatedOutput = synapticMatrix.test(INCOMPLETE_INPUT, DEFAULT_INPUT,
		    ↪  DEFAULT_OUTPUT.length);
		System.out.println("Correct output: " + printArray(DEFAULT_OUTPUT));
		System.out.println("Generated output: " + printArray(generatedOutput))
		    ↪ ;
		System.out.println();


		//Question B.3.
		//Test the generated synaptic matrix with noisy input
		System.out.println("B.3.");
		generatedOutput = synapticMatrix.test(NOISY_INPUT, DEFAULT_INPUT,
		    ↪ DEFAULT_OUTPUT.length);
		System.out.println("Correct output: " + printArray(DEFAULT_OUTPUT));
		System.out.println("Generated output: " + printArray(generatedOutput))
		    ↪ ;
```

```
System.out.println();


//Question B.4.
//Generate and test a new synaptic matrix with 2 inputs and outputs of
    ↪  size 6
System.out.println("B.4");
SynapticMatrix largeSynapticMatrix = new SynapticMatrix(LARGE_INPUT_1,
    ↪  LARGE_OUTPUT_1);
largeSynapticMatrix.train(LARGE_INPUT_2, LARGE_OUTPUT_2);

generatedOutput = largeSynapticMatrix.test(LARGE_INPUT_1,
    ↪ LARGE_INPUT_1, LARGE_OUTPUT_1.length);
System.out.println(largeSynapticMatrix);
System.out.println("Correct output: " + printArray(LARGE_OUTPUT_1));
System.out.println("Generated output: " + printArray(generatedOutput))
    ↪ ;
System.out.println();


//Question B.5.
//Test the new synaptic matrix with distorted input
System.out.println("B.5.");
double[] distortedInput = {1,1,0,0,0,0};
generatedOutput = largeSynapticMatrix.test(LARGE_INPUT_1,
    ↪ LARGE_INPUT_1, LARGE_OUTPUT_1.length);

System.out.println("Correct output: " + printArray(LARGE_OUTPUT_1));
System.out.println("Generated output: " + printArray(generatedOutput))
    ↪ ;
System.out.println();


//Question B.6.
/*
 * Code below will make the network mistake the first input (
     ↪ largeInput) with the second input, recalling the wrong output
     ↪ pattern.
 * Neurons strengthened = 15/36
 * Load parameter = (patterns stored / input layer size) = 3/6 = 1/2
 */
System.out.println("B.6.");
double[] errorInput = {0,1,0,0,1,1};
double[] errorOutput = {0,0,0,0,0,1};

largeSynapticMatrix.train(errorInput, errorOutput);
generatedOutput = largeSynapticMatrix.test(errorInput, errorInput,
    ↪ LARGE_OUTPUT_1.length);
System.out.println("Correct output: " + printArray(errorOutput));
System.out.println("Generated Output: " + printArray(generatedOutput))
    ↪ ;
```

```
                System.out.println();


                //Question C
                //Tests the synaptic matrix using floating point weights as opposed to
                    ↪  binary.
                System.out.println("C.");
                System.out.println(printArray(largeSynapticMatrix.testSigmoid(
                    ↪ LARGE_INPUT_1, LARGE_INPUT_1, LARGE_INPUT_1.length)));
                System.out.println();


                //Question D
                //Tests a synaptic matrix with 3 simultaneus inputs
                System.out.println("D.");
                ThreeInputs.main(null);
                System.out.println();


                //Question E
                //Takes the output of the first layer as the input of the second layer
                    ↪ , acting as a hidden layer.
                System.out.println("E.");
                double[] inputLayer = {0,1,0,1,0,0};
                double[] intermediateLayer = {1,0,0,0,1,0};
                double[] outputLayer = {1,0,0,0,0,0};

                SynapticMatrix hiddenLayer = new SynapticMatrix(inputLayer,
                    ↪ intermediateLayer);
                SynapticMatrix output = new SynapticMatrix(intermediateLayer,
                    ↪ outputLayer);

                generatedOutput = output.test(hiddenLayer.test(inputLayer, inputLayer,
                    ↪  outputLayer.length), hiddenLayer.test(inputLayer, inputLayer,
                    ↪ outputLayer.length), outputLayer.length);
                System.out.println("Correct output: " + printArray(outputLayer));
                System.out.println("Generated Output: " + printArray(generatedOutput))
                    ↪ ;
        }

        //Helper function to print array to StDOUT
        public static String printArray(double[] array) {
                String ret = "";
                for (int i = 0; i < array.length; i++) {
                        ret += array[i] + " ";
                }
                return ret;
        }
}
```

SynapticMatrix.java

```
import org.apache.commons.math3.linear.Array2DRowRealMatrix;
import org.apache.commons.math3.linear.RealMatrix;

public class SynapticMatrix {
	public static final double[] DEFAULT_INPUT = {0,1,0,1};
	public static final double[] DEFAULT_OUTPUT = {1,0,1,0};
	public static final double[] INCOMPLETE_INPUT = {0,0,0,1};
	public static final double[] NOISY_INPUT = {1,1,0,1};

	private RealMatrix synapticMatrix;

	public SynapticMatrix(double[] input, double[] output) {
		generateSynapticMatrix(input, output);
	}

	//Returns the recalled output given an input vector
	public double[] test(double[] input, double[] originalInput, int outputLength
	    ↪ ) {
		double threshold;
		double[] weights = new double[synapticMatrix.getColumn(0).length];
		double[] output = new double[outputLength];
		for (int i = 0; i < weights.length; i++) {
			weights[i] = 1;
		}
		double inputSum = integrate(weights, input);

		if (inputSum < integrate(weights, originalInput)) {
			threshold = inputSum;
		} else {
			threshold = integrate(weights, originalInput);
		}
		for (int i = 0; i < outputLength; i++) {
			output[i] = compare(integrate(weights, synapticMatrix.
			    ↪ getColumn(i)), threshold);
		}
		return output;
	}

	//Returns the recalled output with a simoid function given an input vector
	public double[] testSigmoid(double[] input, double[] originalInput, int
	    ↪ outputLength) {
		double[] weights = new double[synapticMatrix.getColumn(0).length];
		double[] output = new double[outputLength];
		for (int i = 0; i < weights.length; i++) {
			weights[i] = 1;
		}
		for (int i = 0; i < outputLength; i++) {
			output[i] = compareSigmoid(integrate(weights, synapticMatrix.
			    ↪ getColumn(i)));
		}
```

```
        return output;
    }

    //Train an association between an input and an output
    public void train(double[] input, double[] output) {
        for (int i = 0; i < input.length; i++) {
            for (int j = 0; j < output.length; j++) {
                if (input[i] == 1 && output[j] == 1) {
                    synapticMatrix.setEntry(i, j, 1);
                }
            }
        }
    }

    //Instantiate the synaptic matrix on the first set of input
    public void generateSynapticMatrix(double[] input, double[] output) {
        double[][] matrix = new double[input.length][output.length];
        for (int i = 0; i < input.length; i++) {
            for (int j = 0; j < output.length; j++) {
                if (input[i] == 1 && output[j] == 1) {
                    matrix[i][j] = 1;
                } else {
                    matrix[i][j] = 0;
                }
            }
        }
        this.synapticMatrix = new Array2DRowRealMatrix(matrix);
    }

    //Applies weights to the input vector
    private double integrate(double[] weights, double[] firing) {
        int total = 0;
        for (int i = 0; i < firing.length; i++) {
            total += (firing[i] * weights[i]);
        }
        return total;
    }

    //Compares the sum of a column to a threshold value
    private double compare(double sum, double threshold) {
        if (sum >= threshold) {
            return 1;
        } else {
            return 0;
        }
    }

    //Applies the sigmoid function to the sum of a column
    private double compareSigmoid(double sum) {
        return 1 / (1 + Math.pow(Math.E, -sum));
    }
```

```
	@Override
	public String toString() {
		return this.synapticMatrix.toString();
	}
}
```

ThreeInputs.java

```
public class ThreeInputs {

	//Question D
	/*
	 * Evaluates 3 inputs at the same time by collating them in to a single input
	    ↪ , then passing
	 * them to the synaptic matrix along with the size of the output.
	 */
	public static void main(String[] args) {
		double[] input1 = {0,1,0,1};
		double[] input2 = {0,1,1,0};
		double[] input3 = {0,0,0,1};
		double[] output = {1,0,0,1};

		double[] collatedInput = collate(input1, input2, input3);
		SynapticMatrix synapticMatrix = new SynapticMatrix(collatedInput,
		    ↪ output);
		double[] generatedOutput = synapticMatrix.test(collatedInput,
		    ↪ collatedInput, output.length);
		System.out.println("Correct output: " + NeuralNetwork.printArray(
		    ↪ output));
		System.out.println("Generated output: " + NeuralNetwork.printArray(
		    ↪ generatedOutput));
	}

	//Collate 3 inputs in to a single array
	public static double[] collate(double[] input1, double[] input2, double[]
	    ↪ input3) {
		double[] retval = new double[input1.length + input2.length + input3.
		    ↪ length];
		int pos = 0;
		for (int i = 0; i < input1.length; i++) {
			retval[pos] = input1[i];
			pos++;
		}
		for (int i = 0; i < input2.length; i++) {
			retval[pos] = input2[i];
			pos++;
		}
		for (int i = 0; i < input3.length; i++) {
			retval[pos] = input3[i];
			pos++;
```

```
                }
                return retval;
        }
}
```