4/22/2017

Joseph Tran

Neural Networks and Fuzzy Logic

Assignment 1

Table of Contents

[Question 1.1. 3](#_Toc480620187)

[Question 1.2. 4](#_Toc480620188)

[Question 2.1. 5](#_Toc480620189)

[Question 2.2. 6](#_Toc480620190)

# Question 1.1. Discrete Perceptron training

Code behind Question 1.1.

% augmented input vectors

x1 = [0.8, 0.5, 0, 0.1, 1];

x2 = [0.2, 0.1, 1.3, 0.9, 1];

x3 = [0.9, 0.7, 0.3, 0.3, 1];

x4 = [0.2, 0.7, 0.8, 0.2, 1];

x5 = [1, 0.8, 0.5, 0.7, 1];

x6 = [0, 0.2, 0.3, 0.6, 1];

% each of the augmented vectors are placed into a single vector to churn

% through

y = [x1; x2; x3; x4; x5; x6]';

% associated outputs

d = [1, -1, 1, -1, 1, -1];

% given lambda

lambda = 1.5;

% given cycles

cycles = 10;

% starting weight

w = [0.2309, 0.5839, 0.8436, 0.4764, -0.6475]';

% a counter for the cycles to be measure against

inputCounter = 1;

% setting up the output matrix

[dRows, dCols] = size(x1');

output = zeros(dRows, cycles);

for index = 1:cycles

w = variablecorrection(w, lambda, y(:, inputCounter), d(:, inputCounter));

inputCounter = inputCounter + 1;

if inputCounter > size(d)

inputCounter = 1;

end

output(:, index) = w;

end

disp(output);

% weight correction formula given

function output = variablecorrection(w, lambda, y, d)

output = w + 0.5 \* (lambda \* abs(w' \* y) / (y' \* y))...

\* (d - sign(w' \* y)) \* y;

end

## Calculate the final weight

The above code provides the following output:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0.3087 | 0.2150 | 0.3923 | 0.3892 | 0.3892 | 0.3892 | 0.4886 | 0.4886 | 0.4886 | 0.4644 |
| 0.6325 | 0.5857 | 0.7236 | 0.7128 | 0.7128 | 0.7128 | 0.7749 | 0.7749 | 0.7749 | 0.6900 |
| 0.8436 | 0.2345 | 0.2936 | 0.2812 | 0.2812 | 0.2812 | 0.2812 | 0.2812 | 0.2812 | 0.1842 |
| 0.4861 | 0.0644 | 0.1235 | 0.1204 | 0.1204 | 0.1204 | 0.1329 | 0.1329 | 0.1329 | 0.1086 |
| -0.5502 | -1.0188 | -0.8218 | -0.8372 | -0.8372 | -0.8372 | -0.7129 | -0.7129 | -0.7129 | -0.8343 |

Note that the cycle number is the first row and the input provided is entry below the cycle number.

The final weight at cycle 10 is:

|  |
| --- |
| 10 |
| 0.4644 |
| 0.6900 |
| 0.1842 |
| 0.1086 |
| -0.8343 |

Each cycle represents the weights evolving to match the expected output given by the inputs. As noted that the final result is at 10 cycles which is the final weight that meets the training data provided.

## Show that the final weight provides the correct classification of the entire training set

To determine if the final weight provides the correct classification for the training set above, additional code was established to run the final weight against all the inputs and the actual output is measured against the expected output via subtraction.

Anything that does not fill will not be zero.

The validation code is below:

finalWeight = [0.464350117283198;0.689974826342637;0.184158957165242;0.108589082355343;-0.834298947076565];

% copied from q11

inputs = [0.8,0.2,0.9,0.2,1,0;0.5,0.1,0.7,0.7,0.8,0.2;0,1.3,0.3,0.8,0.5,0.3;0.1,0.9,0.3,0.2,0.7,0.6;1,1,1,1,1,1];

% associated outputs

outputs = [1, -1, 1, -1, 1, -1];

[iRows, iCols] = size(inputs);

patternErrors = zeros(1, iCols);

validation(inputs, outputs, finalWeight)

disp(patternErrors);

function validation(inputs, outputs, weight)

[iRows, iCols] = size(inputs);

for index = 1:iCols

error = calculateVariation(outputs(:, index), weight, inputs(:, index));

patternErrors(:, index) = error;

end

end

function error = calculateVariation(expectedOutput, weight, input)

actualOutput = sign(weight' \* input);

disp([expectedOutput, actualOutput]);

error = (expectedOutput - actualOutput);

end

The results are as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Inputs | 1 | 2 | 3 | 4 | 5 | 6 |
| Errors | 0 | 0 | 0 | 0 | 0 | 0 |

As noted that when placed back into a section of a system (one that does not feedforward) it meets all the expected outputs with zero errors. This shows that the final weight as mentioned above provides the correct classification of the training data.

## Plot the pattern error curve

Calculating the cycle error curve is simply determining the error at which the system is evolving at.

This can be calculated via the equation of:

Where:

* Ep is the p-th error step, where p is the pattern
* dp is the expected output of the p-th pattern
* zp is the actual output from the activation function from the p-th pattern

In essence it is the variation of the actual output in comparison to the expected output.

As such some modification were made to the code to capture the pattern error run against an activation function and the expected output.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Cycles** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **Inputs** | 1 | 2 | 3 | 4 | 5 | 6 | 1 | 2 | 3 | 4 |
| **Variation** | 2 | -2 | 2 | -2 | 0 | 0 | 2 | 0 | 0 | -2 |
| **0.5\*Error2** | 2 | 2 | 2 | 2 | 0 | 0 | 2 | 0 | 0 | 2 |

It is observed that as the cycle progresses, the errors are minimised and adjusted. As noted that after the 10th cycle, the final weight has been obtained.

## Plot the cycle error curve

In accordance to the equation of:

Where:

* Ec is the error cycle
* P is the patterns that are processed
* dp is the expected output of the p-th pattern
* zp is the actual output from the p-th pattern
* Ep is the pattern cycle

In essence cycle error curve is assessing how for each “evolved” weight how well it fits the expected output

# Question 1.2. Continuous Perceptron training

## Calculate w7

## Calculate the weight vector w301 after 50 cycles

## Plot the cycle error curve

## How would w7 and w301 classify the entire training set?

The weights have been modelled after the current set of data. As data is passed through the neural network, the weights would slowly converge on a function that approximates the generalised features of each input.

# Question 2.1. Flight simulation

## Find R = M X A

The code that was used is as follows:

M = [0, 0.25, 0.75, 1, 0.75, 0.25, 0];

A = [0, 0.3, 0.6, 1, 0.6, 0.3, 0];

R = getRelation(M', A);

function output = getRelation(a, b)

output = min(a, b);

end

R is calculated to be via reading the minimum of both M’ and A:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0.725 | 0.73 | 0.735 | 0.74 | 0.745 | 0.75 | 0.755 |
| 8350 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8400 | 0 | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 | 0 |
| 8450 | 0 | 0.3 | 0.6 | 0.75 | 0.6 | 0.3 | 0 |
| 8500 | 0 | 0.3 | 0.6 | 1 | 0.6 | 0.3 | 0 |
| 8550 | 0 | 0.3 | 0.6 | 0.75 | 0.6 | 0.3 | 0 |
| 8600 | 0 | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 | 0 |
| 8650 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## Find A1a = M1 ο R via max-min composition

Using the R matrix that was calculated from the previous question, it is reused to calculate A1a. The code is as follows.

M1 = [0, 0.5, 0.8, 1, 0.6, 0.2, 0];

A1a = maxMinComposition(M1', R);

function output = maxMinComposition(a, b)

output = max(min(a,[],2),... % read the rows of a

max(b,[],1)); % read the columns of b

end

A1a is calculated to be:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0.725 | 0.73 | 0.735 | 0.74 | 0.745 | 0.75 | 0.755 |
| 8350 | 0 | 0.3 | 0.6 | 1 | 0.6 | 0.3 | 0 |
| 8400 | 0.5 | 0.5 | 0.6 | 1 | 0.6 | 0.5 | 0.5 |
| 8450 | 0.8 | 0.8 | 0.8 | 1 | 0.8 | 0.8 | 0.8 |
| 8500 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 8550 | 0.6 | 0.6 | 0.6 | 1 | 0.6 | 0.6 | 0.6 |
| 8600 | 0.2 | 0.3 | 0.6 | 1 | 0.6 | 0.3 | 0.2 |
| 8650 | 0 | 0.3 | 0.6 | 1 | 0.6 | 0.3 | 0 |

Each row of M1’ and R have their minimum values calculated and then combined via a max comparison between the two respective rows to form each cell of this matrix.

## Find A1b = M1 ο R via sum-product composition

# Question 2.2. Laser Beam Alignment

## Calculate the defuzzified voltage output via Mean of Maximum (MOM)

Given e = 3.2 and ce = -0.47

### Fuzzification of E

### Fuzzification of CE

### Defuzzified output voltage

## Calculate the defuzzified voltage output via Centre of Area (COA)

Given e = 3.2 and ce = -0.47

### Fuzzification of E and CE

### Total Area

### Total Moment

### Defuzzified output voltage