Universidade de Coimbra

Departamento de Engenharia Informática

Aprendizagem Computacional

Optical Character Recognition

Autor:
António Carlos Lima
2011166926

Autor:
David Cardoso
2011164039

October 17, 2014

CONTENTS

LIST OF FIGURES

ACRONYMS

OCR Optical Character Recognition

AM Associative Memory

AF Activation Function

LF Learning Function

HB Hebb's Rule

LN Linear Neuron

1

INTRODUCTION

1.1 DIGIT RECOGNITION

Neuronal networks models will be developed for character recognition problems. The characters to be recognized are the 10 Arabic numerals:

Figure 1: Recognizable arabic numerals

We assume that each character is defined by a matrix composed of binary (0/1) elements. In particular, we assume that the digits are defined as a 16x16 matrix. For example, the following matrix can represent the digit ”0” (zero) supposedly manually traced by a user in some device.

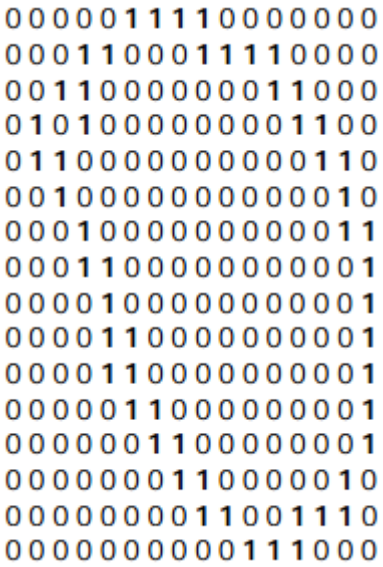

Figure 2: ”0”’s internal matrix representation

Another core assumption is that the written numerals must be similar to their Arial font counterparts, for better precision, as variations will compromise the neural network’s recognition accuracy.

Figure 3: Arial’s arabic numerals

2

DATA SET

2.1 HOW DOES THE DATA SET INFLUENCE THE PERFORMANCE OF THE CLASSIFICATION SYSTEM?

At first we tried, as was suggested, a training data set consisting of 50 examples (5 for each numeric character). However, we soon realized that this was a rather small amount of samples to train our neural network. Some test examples would be classified correctly but the majority would be ambiguously or just wrongly classified.

We interpreted these early results as an indicator that a larger training data set was needed, and so we agreed on feeding our network 50 train examples for each numeric character, adding up to a total of 500 training cases. The results improved substantially for various configurations, as we will later show in the "Results" chapter.

After testing the perfect characters (Arial's arabic numerals) we also found that these could not be accurately classified, therefore we also included them in the training data set to improve the neural network's recognition accuracy.
Thus, we have a grand total of 510 training samples.

3

NEURAL NETWORK ARCHITECTURE

3.1 ASSOCIATIVE MEMORY AND CLASSIFIER

The first module, the **associative memory**, has as input the vector P1 (dimension 256,1), that defines the character to be classified, corresponding to the binary matrix (dimension 16,16).

This Associative Memory can be seen as a "filter" or "corrector":

- if the input character is not perfect, the associative memory has the capacity to provide an output P2 (dimension 256,1) that is a "more perfect character".

The associative memory is a neural network consisting of:

- One single layer
- Linear activation functions
- Without bias

It is characterized by:

$$P_2 = W_P * P_1$$

The weights, $W_P(256, 256)$, are evaluated using the pseudo-inverse method or the Hebb rule,

$$W_P = T * pinv(P)$$

where T (256, Q) are the desired Q outputs, for a given P inputs (256,Q).

The second neural network module is the **classifier**.

The input is the output of the associative memory (P2), or the original character to be classified (P1), in case associative memory was not used. The output (A) is the class where the digit belongs.

For the digits to be classified '1' '2' '3' '4' '5' '6' '7' '8' '9' '0', the following classes are assumed [1, 2, 3, 4, 5, 6, 7, 8, 9, 0].

It is assumed that the classifier is a neural network (see chapter 5) consisting of:

- One single layer
- A linear or non-linear activation function, namely
  1. Perceptron
  2. Linear
  3. Sigmoidal
- With bias in each neuron

It is characterized by:

$$A = f(W_N * P_2 + b)$$

Where matrix W N has dimensions (10,256) and b dimensions (10,1). The input (P2) is a vector of dimension (256,1). The output (A) has dimension (10,1).

Concerning the activation function there are three alternatives:

- Perceptron : $A = hardlim(W_N * P_2 + b)$
- Linear : $A = W_N * P_2 + b$
- Sigmoidal : $A = logsig(W_N * P_2 + b)$

For example, considering as input the digit defined in the introduction (zero), the output should belong to class 0:

$$A = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Figure 4: "0"'s class associations

The neural network parameters should be evaluated using the perceptron rule, if harlim is used, or the gradient method, if purlin or logsig are used.

3.2 CLASSIFICATION

Figure 5: Classifier model of our neural network

As the image shows, the characters (vectors with dimensions 256x1) are directly provided to the classification system. There is no pre-filtering of the data therefore, the classifier must have a considerable generalization capability. The output is the same as the one previously detailed.

For this report we will consider simulations that run their training for a maximum of 1000 epochs whilst trying to achieve a performance lesser or equal to 0.000001.

4

RESULTS

4.1 NOTATION

In order to minimize the space required to present our simulation's results we use a special numbering code for the inclusion of associative memory, weighing method and activation function applied to the neural network. As such, the following are the possible representations for a given x_y_z test case:

- Presence of an Associative Memory
  1. Yes
  2. No
- Weighing Method
  1. Transpose weighing method
  2. Hebb's Rule
- Activation Function and, weight and bias Learning Function
  1. Sigmoidal with Gradient descent
  2. Linear with Gradient descent
  3. Hard-Limit with Perceptron

We also use a shortened versions of our sample and test sets, e.g.: W input - N, which follows these guidelines:

- W:
  - perfect input: control test sample with the 10 Arial numerals
  - drawn input: hand-written test sample with 50 numerals (5 for each numeral)
- N: number of training samples provided plus the desired output Arial numerals

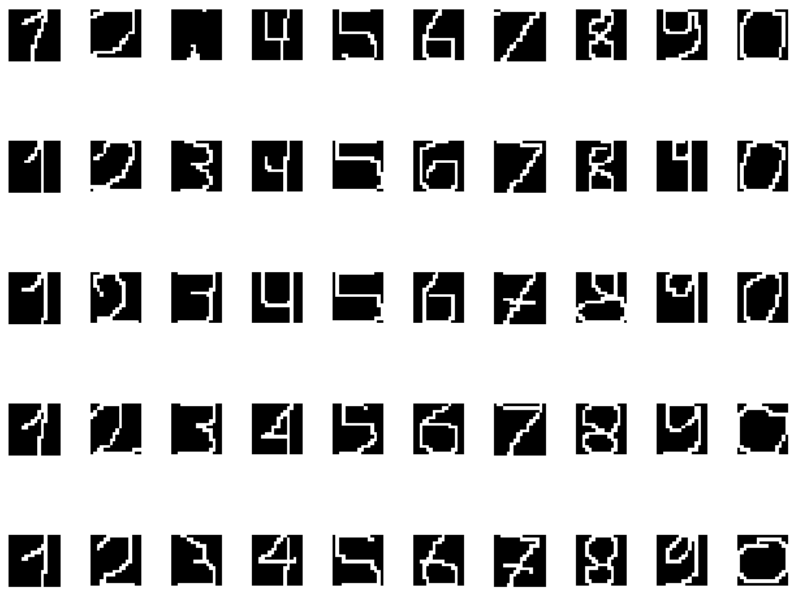

Figure 6: ”drawn inputs” test input

Figure 7: ”perfect inputs”, the control test input

So, for example, test case ”2_1_3 -> perfect input - 500” means ”No associative memory, using the transpose weighing method, on a Hard-limit activation function with a Perceptron learning function, applied to the Perfect Arial input and with 500 training case sample”. You now understand why we chose this code notation instead of plain english.

4.2 IS THE CLASSIFICATION SYSTEM ABLE TO ACHIEVE THE MAIN OBJECTIVE?

Correctly Classified

case	perfect input – 500	drawn input – 500	perfect input – 100	drawn input – 100
1_1_1	1	5	1	5
1_1_2	3	7	1	5
1_1_3	1	5	1	5
1_2_1	1	6	1	3
1_2_2	9	33	10	27
1_2_3	9	24	10	16
2_1_1	0	6	1	2
2_1_2	9	25	10	12
2_1_3	10	41	10	29
2_2_1	0	9	0	4
2_2_2	9	25	10	5
2_2_3	10	43	10	29

Figure 8: Table with number of correctly classified test cases for each scenario and input type

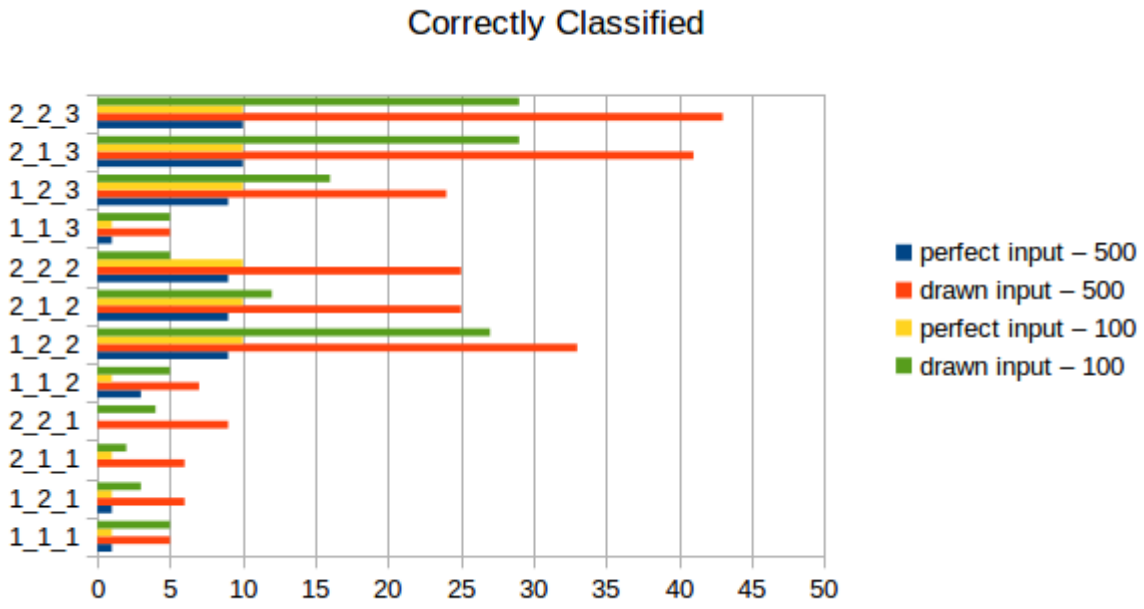

Figure 9: Graphic with number of correctly classified test cases for each scenario and input type

As we can see from these results, the main objective, correct classification of digits, is achieved, even if minimally, in almost all training scenarios. Scenarios 1_1_1, 1_1_2, 1_1_3, 1_2_1, 2_1_1 and 2_2_1 would theoretically be able to better classify the inputs given enough training epochs to achieve the desired precision.

4.3 WHICH IS THE PERCENTAGE OF WELL CLASSIFIED DIGITS? WHICH IS THE PERCENTAGE OF WELL CLASSIFIED NEW INPUTS?

As we can see from these results, the main objective, correct classification of digits, is achieved in only a subset of all the possible scenarios, namely 2_1_3 and 2_2_3 score the highest with about 75% and 76.7% total correct classifications, respectively, whilst honorable mentions go to 1_2_2, 1_2_3 and 2_1_2 with about 65.8%, 49% and 47% total correct classifications, respectively.

4.4 HOW IS THE GENERALIZATION CAPACITY?

We can clearly observe in figure 9 on page 14 that a substantially bigger training set (5x) is enough to yield more correct classifications for the drawn input (albeit it's not so clear what its effect is on the control test).

4.5 IS THE CLASSIFICATION SYSTEM ROBUST ENOUGH TO GIVE CORRECT OUTPUTS WHEN NEW INPUTS ARE NOT PERFECT?

Observing the "drawn input - 500" and "drawn input - 100" columns of 8 on page 14 we can conclude that for imperfect inputs ("drawn input") the best results are achieved using the Linear and Hard-Limit activation functions using, respectively, the gradient and perceptron learning functions.

4.6 OTHER DATA

The following images and tables neatly sum up other gather information about running time, number of epochs needed and attained performance for all test scenarios.

Time

case	perfect input – 500	drawn input – 500	perfect input – 100	drawn input – 100
1_1_1	00:10:38	00:10:04	00:02:21	00:02:27
1_1_2	00:10:52	00:10:16	00:01:28	00:01:21
1_1_3	00:00:15	00:00:13	00:00:00	00:00:01
1_2_1	00:10:21	00:10:02	00:02:22	00:02:21
1_2_2	00:10:02	00:10:30	00:01:30	00:01:23
1_2_3	00:00:12	00:00:13	00:00:01	00:00:00
2_1_1	00:10:06	00:10:45	00:02:24	00:02:15
2_1_2	00:10:16	00:10:39	00:01:27	00:01:22
2_1_3	00:00:12	00:00:13	00:00:01	00:00:00
2_2_1	00:10:25	00:10:46	00:02:38	00:02:18
2_2_2	00:10:16	00:10:22	00:01:33	00:01:20
2_2_3	00:00:14	00:00:11	00:00:01	00:00:01

Figure 10: Table with simulation time for each scenario and input type

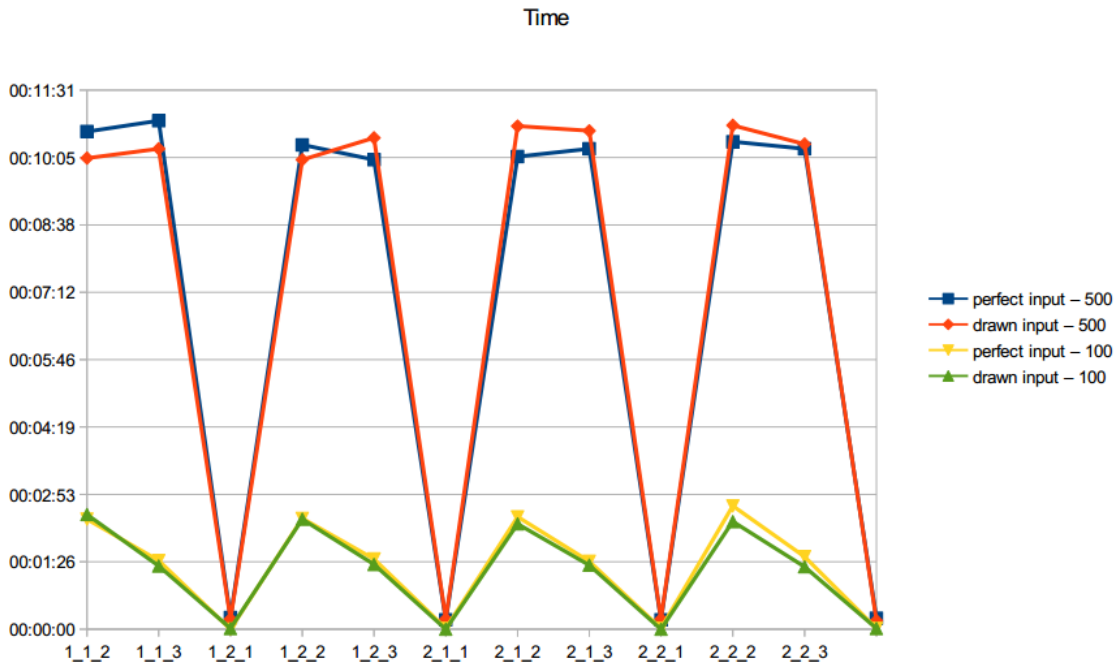

Figure 11: Graphic with simulation time cases for each scenario and input type

Performance

case	perfect input – 500	drawn input – 500	perfect input – 100	drawn input – 100
1_1_1	4580	4580	990	990
1_1_2	275	275	0,000000994	0,00000099
1_1_3	0	0	0	0
1_2_1	4120	4160	990	990
1_2_2	274	275	0,000000991	0,000000998
1_2_3	0	0	0	0
2_1_1	4580	4130	990	990
2_1_2	274	275	0,000000995	0,000000985
2_1_3	0	0	0	0
2_2_1	4130	4580	990	990
2_2_2	274	275	0,000000985	0,000000996
2_2_3	0	0	0	0

Figure 12: Table with achieved performance for each scenario and input type

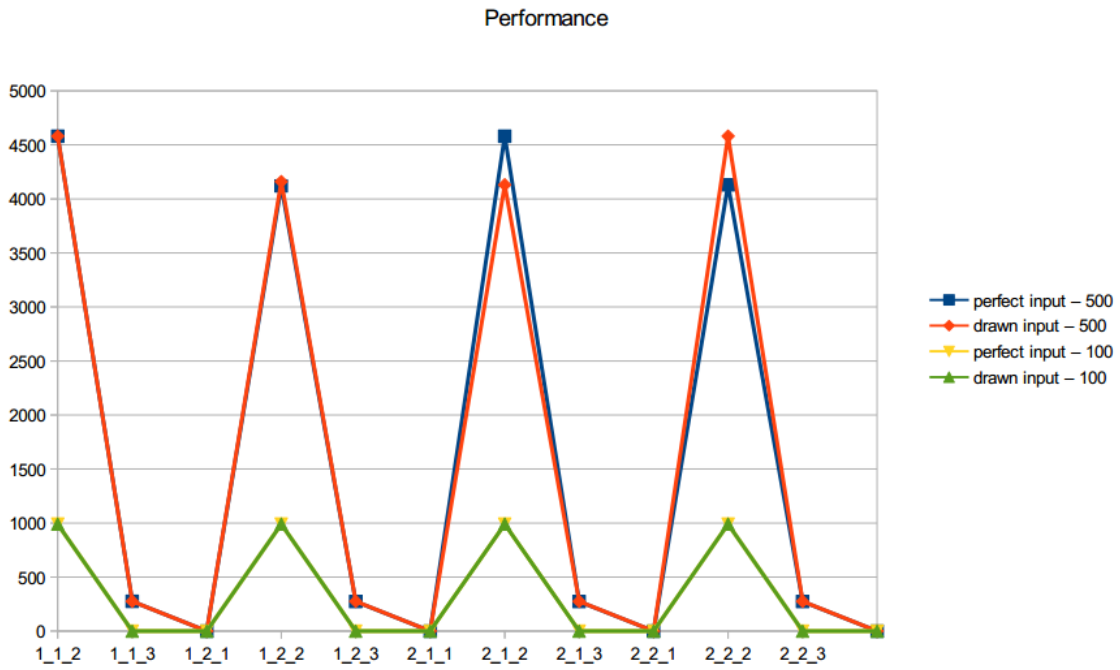

Figure 13: Graphic with achieved performance for each scenario and input type

Epochs

case	perfect input – 500	drawn input – 500	perfect input – 100	drawn input – 100
1_1_1	1000	1000	1000	1000
1_1_2	1000	1000	600	590
1_1_3	33	34	9	11
1_2_1	1000	1000	1000	1000
1_2_2	1000	1000	602	600
1_2_3	31	31	13	10
2_1_1	1000	1000	1000	1000
2_1_2	1000	1000	596	596
2_1_3	30	32	11	10
2_2_1	1000	1000	1000	1000
2_2_2	1000	1000	601	577
2_2_3	35	30	12	12

Figure 14: Table with number of epochs necessary for each scenario and input type

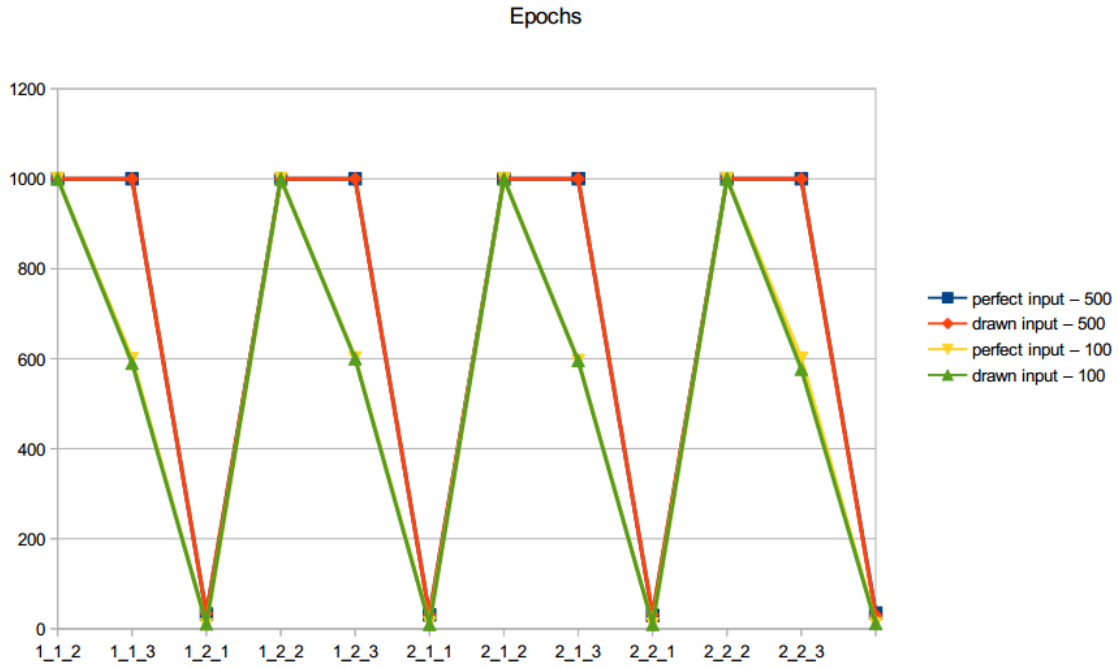

Figure 15: Graphic with number of epochs necessary for each scenario and input type

Everything combined

	Perfect input - 500 training cases			
case	Correctly Classified	Epochs	Time	Performance
1_1_1	1	1000	00:10:38	4580
1_1_2	3	1000	00:10:52	275
1_1_3	1	33	00:00:15	0
1_2_1	1	1000	00:10:21	4120
1_2_2	9	1000	00:10:02	274
1_2_3	9	31	00:00:12	0
2_1_1	0	1000	00:10:06	4580
2_1_2	9	1000	00:10:16	274
2_1_3	10	30	00:00:12	0
2_2_1	0	1000	00:10:25	4130
2_2_2	9	1000	00:10:16	274
2_2_3	10	35	00:00:14	0

	Perfect input – 100 training cases			
case	Correctly Classified	Epochs	Time	Performance
1_1_1	1	1000	00:02:21	990
1_1_2	1	600	00:01:28	0,000000994
1_1_3	1	9	00:00:00	0
1_2_1	1	1000	00:02:22	990
1_2_2	10	602	00:01:30	0,000000991
1_2_3	10	13	00:00:01	0
2_1_1	1	1000	00:02:24	990
2_1_2	10	596	00:01:27	0,000000995
2_1_3	10	11	00:00:01	0
2_2_1	0	1000	00:02:38	990
2_2_2	10	601	00:01:33	0,000000985
2_2_3	10	12	00:00:01	0

	Drawn input - 500 training cases			
case	Correctly Classified	Epochs	Time	Performance
1_1_1	5	1000	00:10:04	4580
1_1_2	7	1000	00:10:16	275
1_1_3	5	34	00:00:13	0
1_2_1	6	1000	00:10:02	4160
1_2_2	33	1000	00:10:30	275
1_2_3	24	31	00:00:13	0
2_1_1	6	1000	00:10:45	4130
2_1_2	25	1000	00:10:39	275
2_1_3	41	32	00:00:13	0
2_2_1	9	1000	00:10:46	4580
2_2_2	25	1000	00:10:22	275
2_2_3	43	30	00:00:11	0

	Drawn input - 100 training cases			
case	Correctly Classified	Epochs	Time	Performance
1_1_1	5	1000	00:02:27	990
1_1_2	5	590	00:01:21	0,00000099
1_1_3	5	11	00:00:01	0
1_2_1	3	1000	00:02:21	990
1_2_2	27	600	00:01:23	0,000000998
1_2_3	16	10	00:00:00	0
2_1_1	2	1000	00:02:15	990
2_1_2	12	596	00:01:22	0,000000985
2_1_3	29	10	00:00:00	0
2_2_1	4	1000	00:02:18	990
2_2_2	5	577	00:01:20	0,000000996
2_2_3	29	12	00:00:01	0

Figure 16: Tables with all information for each scenario and input type

4.7 WHICH ARCHITECTURE PROVIDES BETTER RESULTS: ONLY THE CLASSIFIER OR THE ASSOCIATIVE MEMORY AND THE CLASSIFIER?

The classifier on its own provides better results than its counterparts which make use of the associative memory. Our hypothesis for this behaviour is later described in the conclusions section of this report.

4.8 WHICH IS THE BEST ACTIVATION FUNCTION: HARD-LIMIT, LINEAR OR SIGMOIDAL?

Overall, with an Associative Memory, the Linear activation function with the Gradient learning function has the best results, however, without an Associative Memory we reckon that the Hard-Limit activation function with the Perceptron learning function is the top contender.

4.9 DOES THE HEBB RULE PERFORM WELL?

Hebb vs Linear

Linear Neuron

case	500 LN – perfect input	500 LN – drawn input	100 LN – perfect input	100 LN – drawn input
1_1_1	1	5	1	5
1_1_2	3	7	1	5
1_1_3	1	5	1	5
2_1_1	0	6	1	2
2_1_2	9	25	10	12
2_1_3	10	41	10	29

Hebb's Rule

case	500 HR – perfect input	500 HR – drawn input	100 HR – perfect input	100 HR – drawn input
1_2_1	1	6	1	3
1_2_2	9	33	10	27
1_2_3	9	24	10	16
2_2_1	0	9	0	4
2_2_2	9	25	10	5
2_2_3	10	43	10	29

Figure 17: Table comparing Hebb's Rule with Linear Neuron results

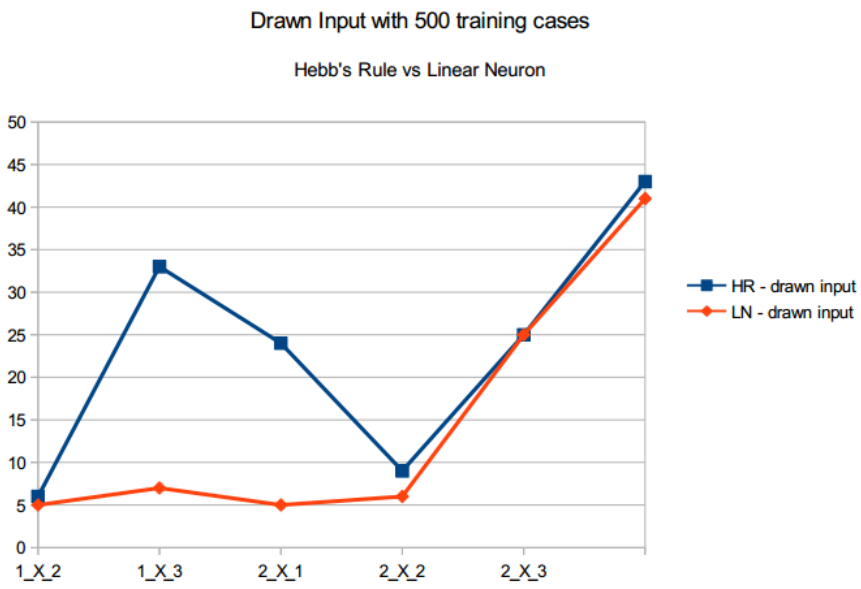

Figure 18: Graphic comparing Hebb's Rule with Linear Neuron results for drawn input and 500 training samples

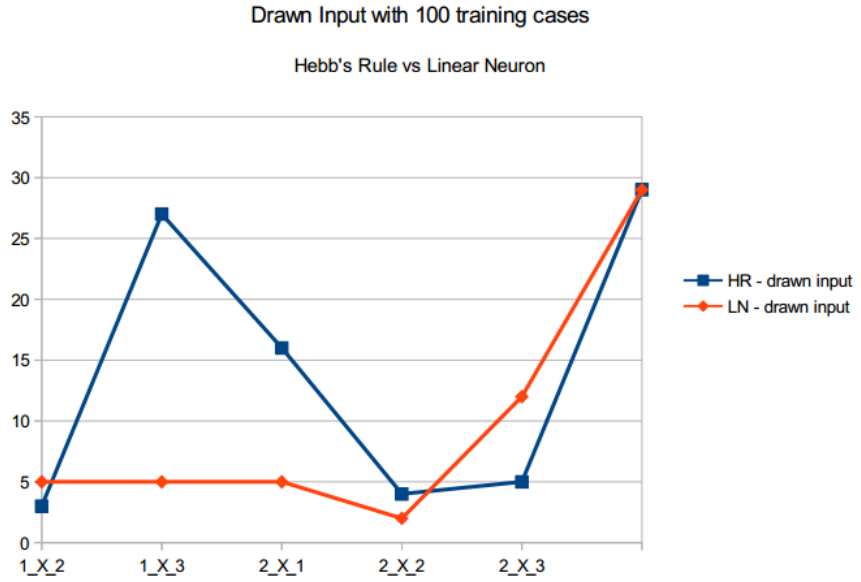

Figure 19: Graphic comparing Hebb's Rule with Linear Neuron results for drawn input and 100 training samples

Figure 20: Graphic comparing Hebb's Rule with Linear Neuron results for perfect input and 500 training samples

Figure 21: Graphic comparing Hebb's Rule with Linear Neuron results for perfect input and 100 training samples

As the results on 17 on page 20 (as well as the graphs for this table) show, Hebb's Rule provides an overall substantial improvement to the networks' classifying capabilities in contrast to a simple Linear Neuron.

5

CONCLUSIONS

Given enough training cases and desired target outputs (a couple thousand or more per desired output set), a neural network can successfully recognize human input and correctly classify it with a very high degree of accuracy.

We couldn't help but notice that the use of an Associative Memory (AM) doesn't boost our neural networks' number of correctly classified numerals. We believe this is because for this report we only had to use a single layer in our neural network, which in turn isn't enough for the AM's effects to take place. Since as AM is a means of pattern recognition, having the benefits of a multi-layered feedforward network might be better than the simple connections we were presented with. Another aspect that could be affecting our results is the sheer number of training examples possibly being too low, hence the created associations aren't as strong as they could possibly be.

We can now create all sorts of simple, relatively fast and relatively accurate Recognition Networks for any problem that already has desired outputs, whose possibilities are, for all intents and purposes, infinite.

6

SIMULATIONS

The following are all the simulations that were ran for this report, first the trained neural network followed by its classification of the input.

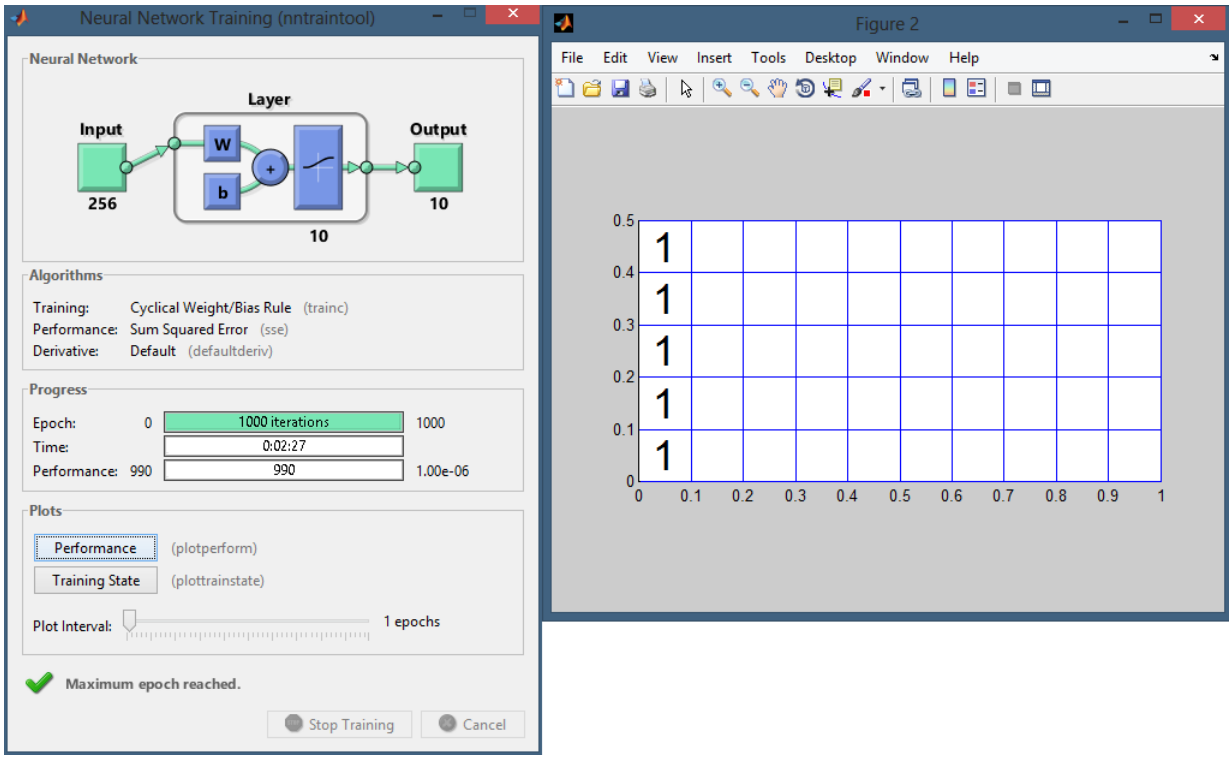

Figure 22: Associative Memory, using the transpose weighing method, Sigmoidal with Gradient descent, 100 train input values, perfect Arial numerals.

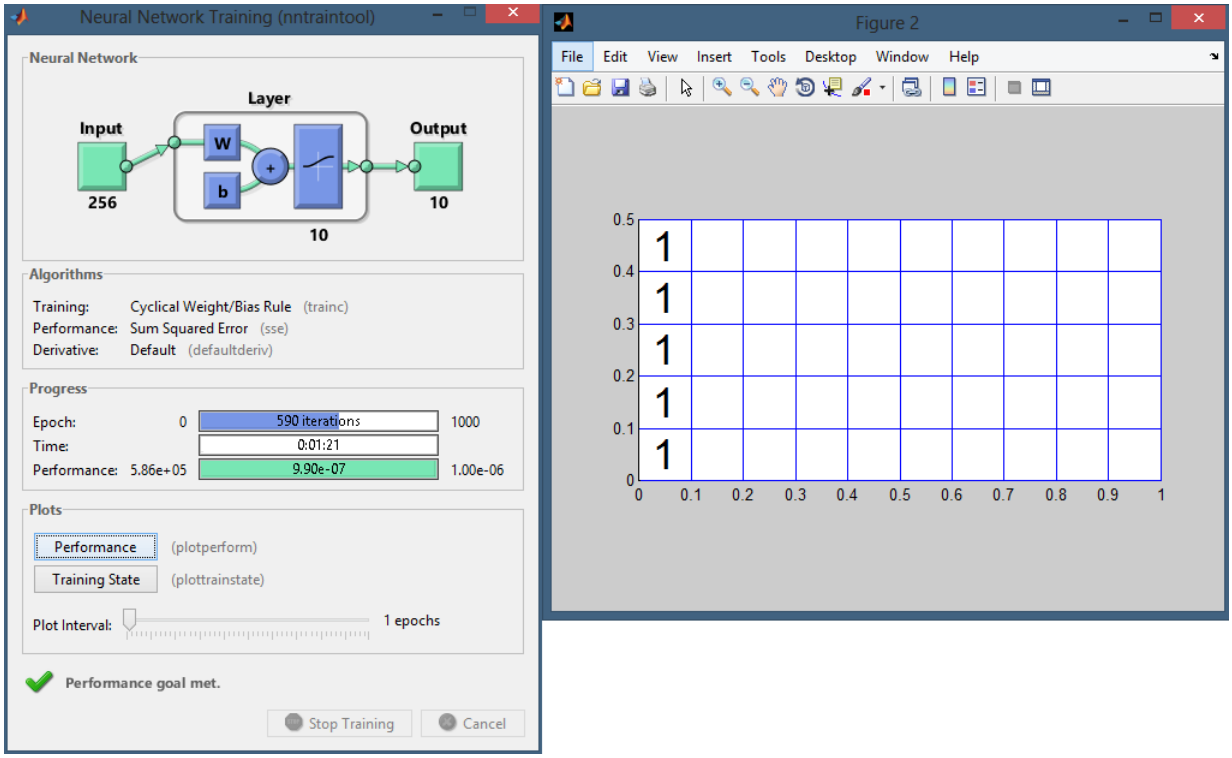

Figure 23: Associative Memory, using the transpose weighing method, Linear with Gradient descent, 100 train input values, perfect Arial numerals.

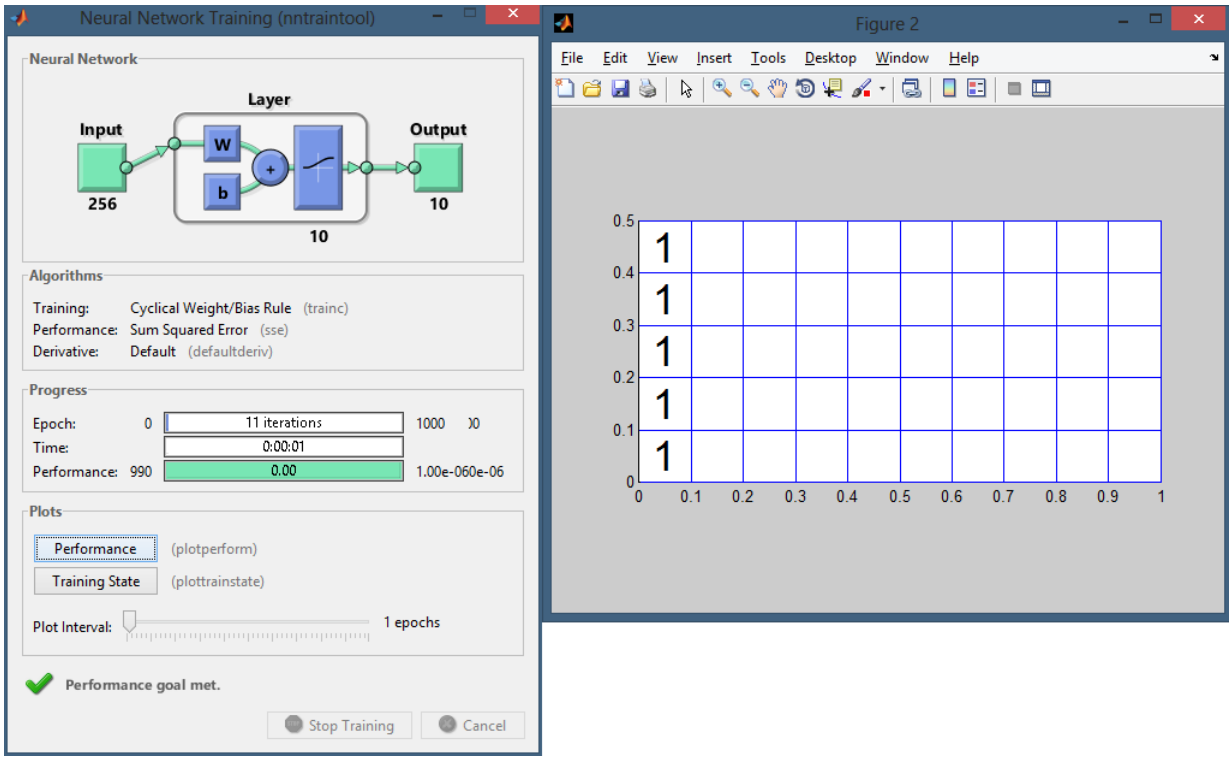

Figure 24: Associative Memory, using the transpose weighing method, Hard-Limit with Perceptron, 100 train input values, perfect Arial numerals.

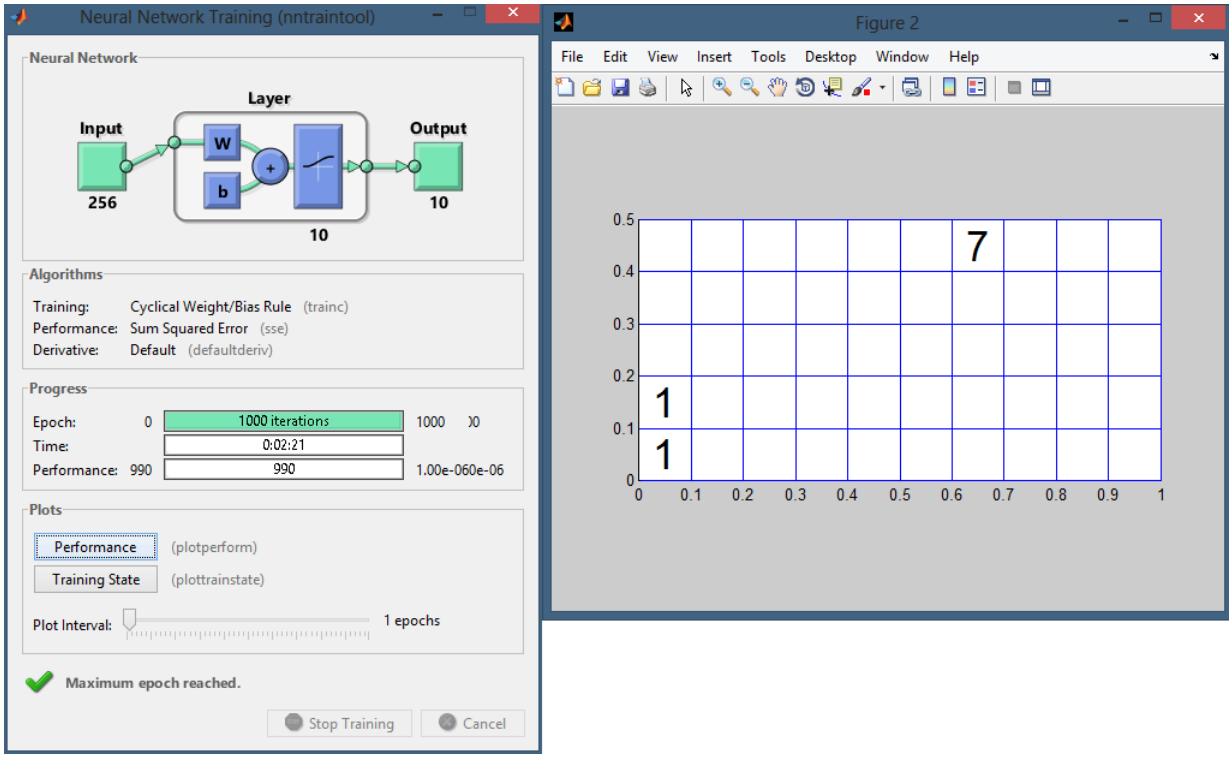

Figure 25: Associative Memory, using the Hebb's rule, Sigmoidal with Gradient descent, 100 train input values, perfect Arial numerals.

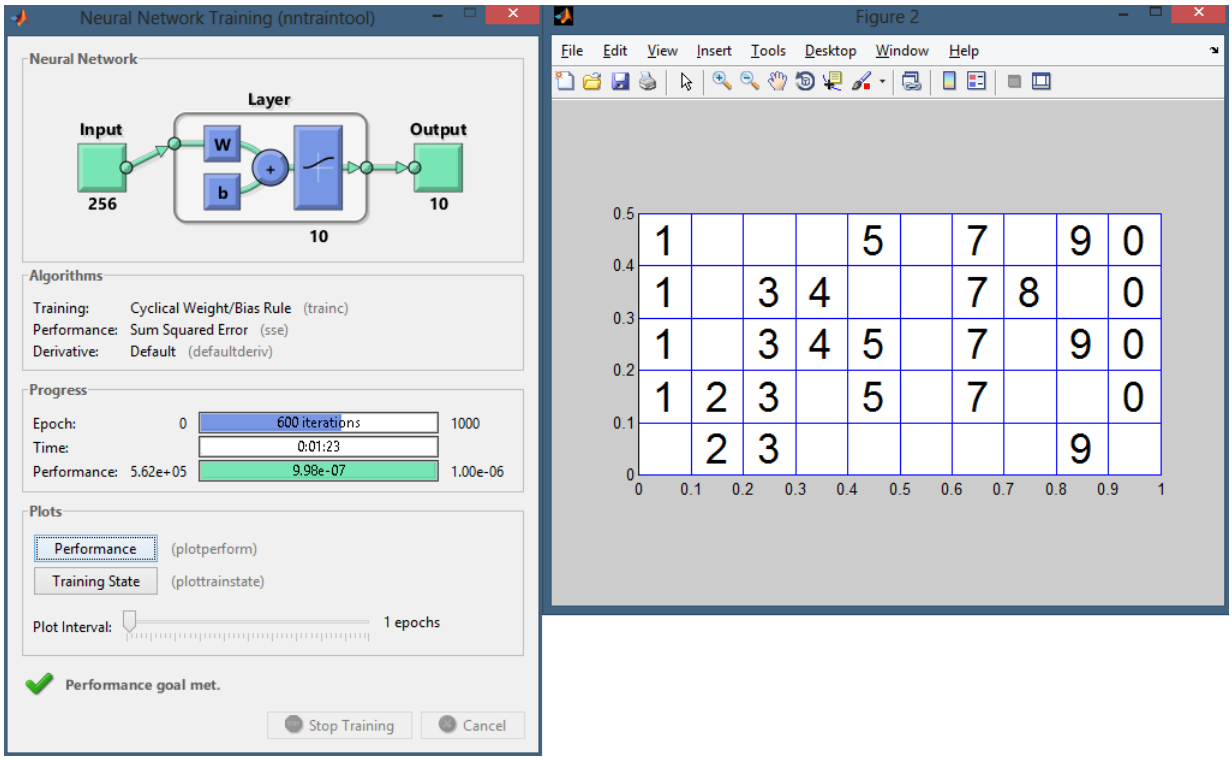

Figure 26: Associative Memory, using the Hebb's rule, Linear with Gradient descent, 100 train input values, perfect Arial numerals.

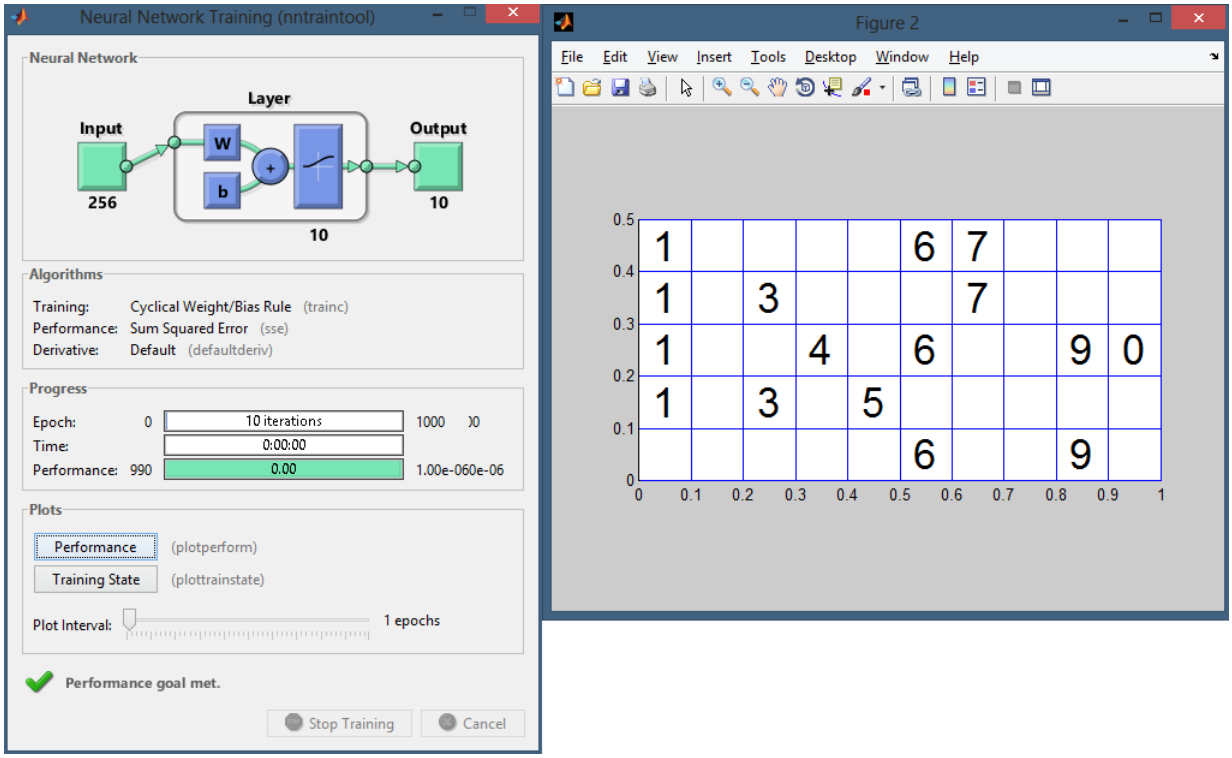

Figure 27: Associative Memory, using the Hebb's rule, Hard-Limit with Perceptron, 100 train input values, perfect Arial numerals.

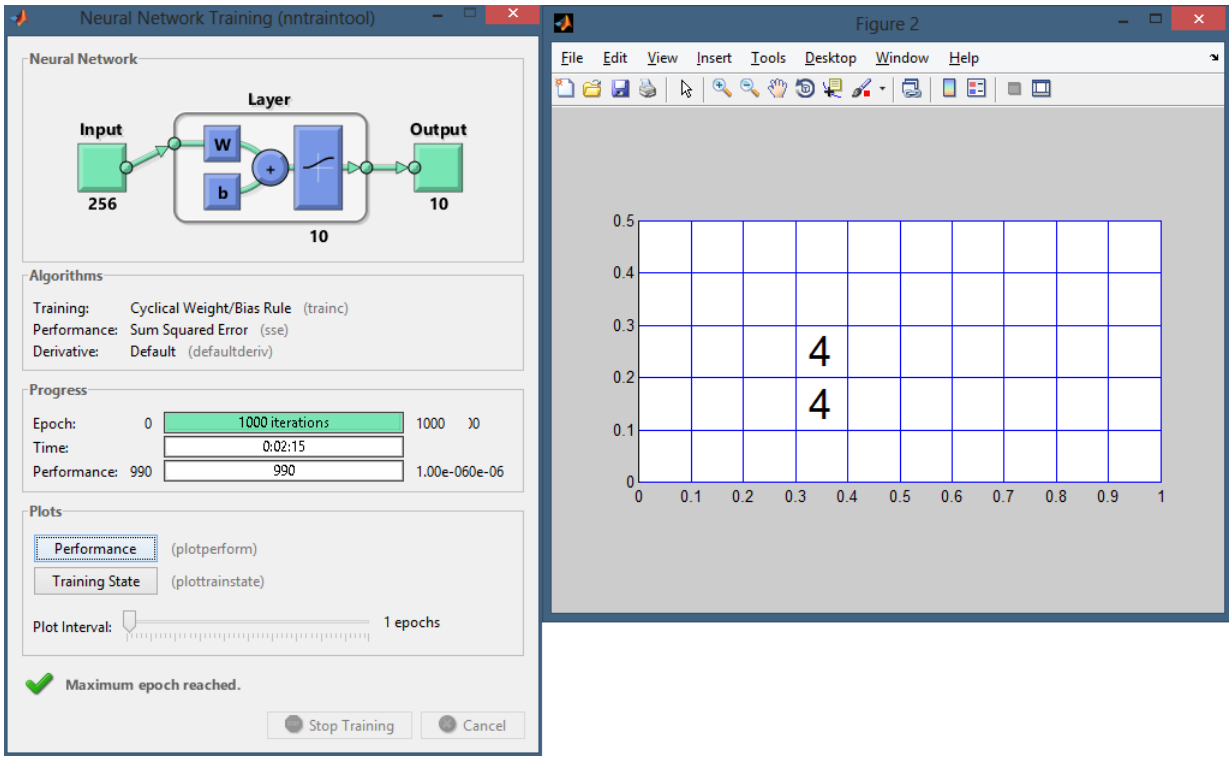

Figure 28: No Associative Memory, Sigmoidal with Gradient descent, 100 train input values, perfect Arial numerals.

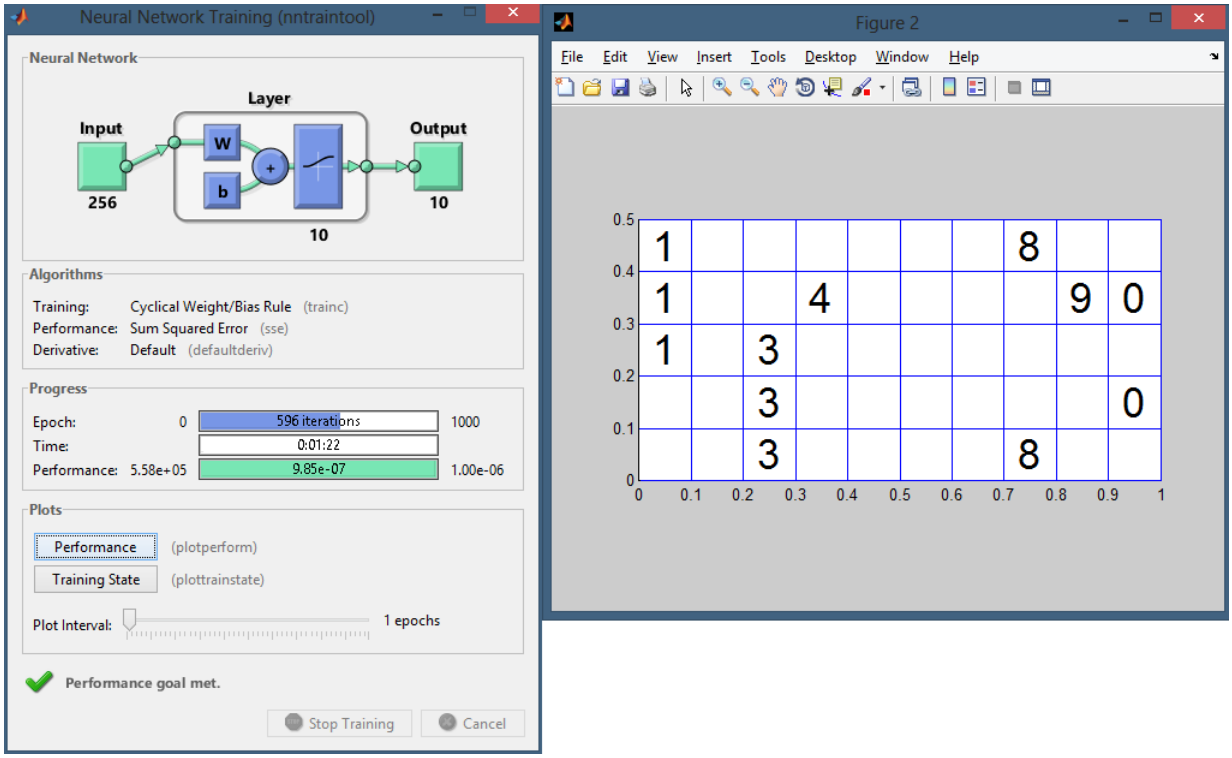

Figure 29: No Associative Memory, Linear with Gradient descent, 100 train input values, perfect Arial numerals.

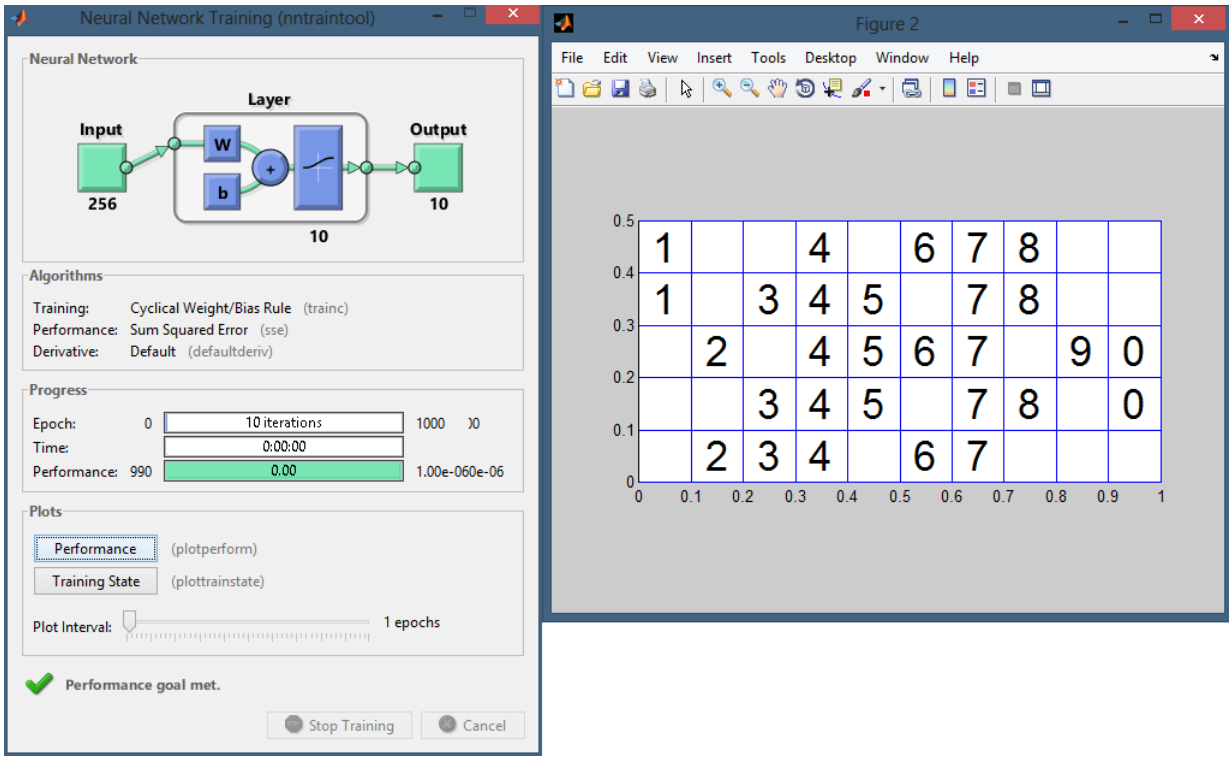

Figure 30: No Associative Memory, Hard-Limit with Perceptron, 100 train input values, perfect Arial numerals.

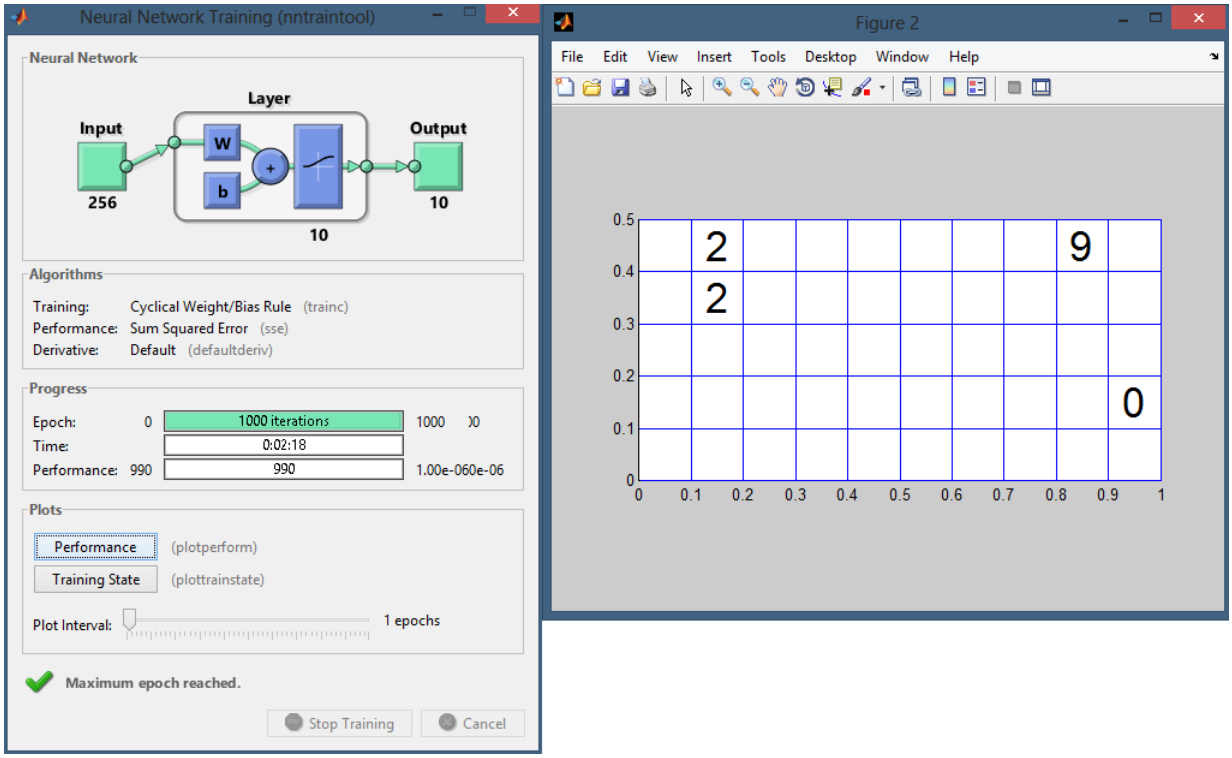

Figure 31: No Associative Memory, Sigmoidal with Gradient descent, 100 train input values, perfect Arial numerals.

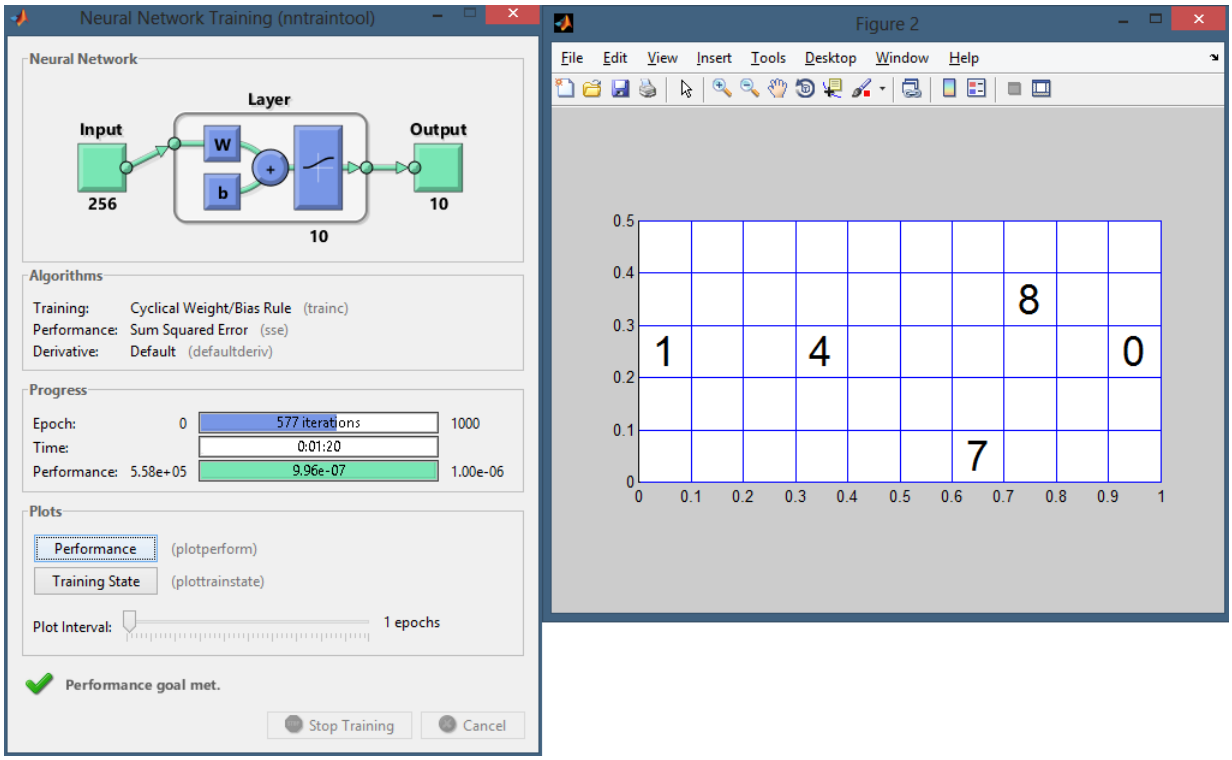

Figure 32: No Associative Memory, Linear with Gradient descent, 100 train input values, perfect Arial numerals.

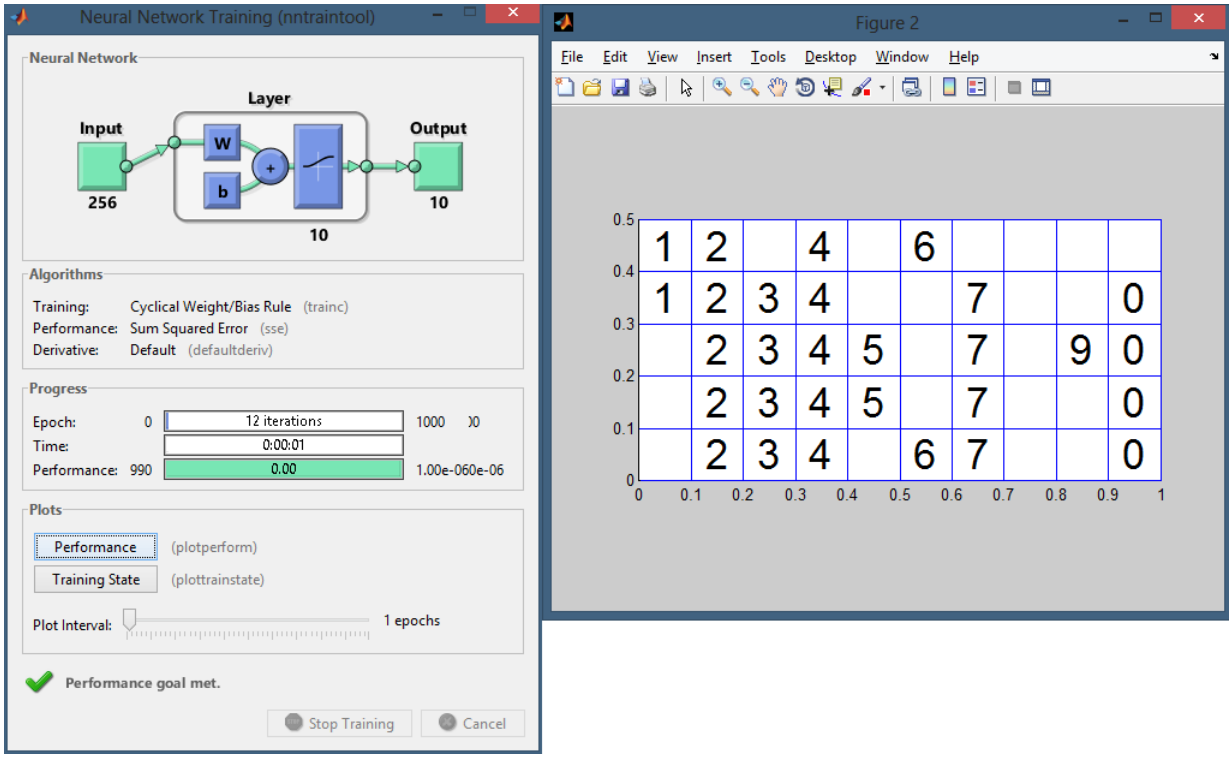

Figure 33: No Associative Memory, Hard-Limit with Perceptron, 100 train input values, perfect Arial numerals.

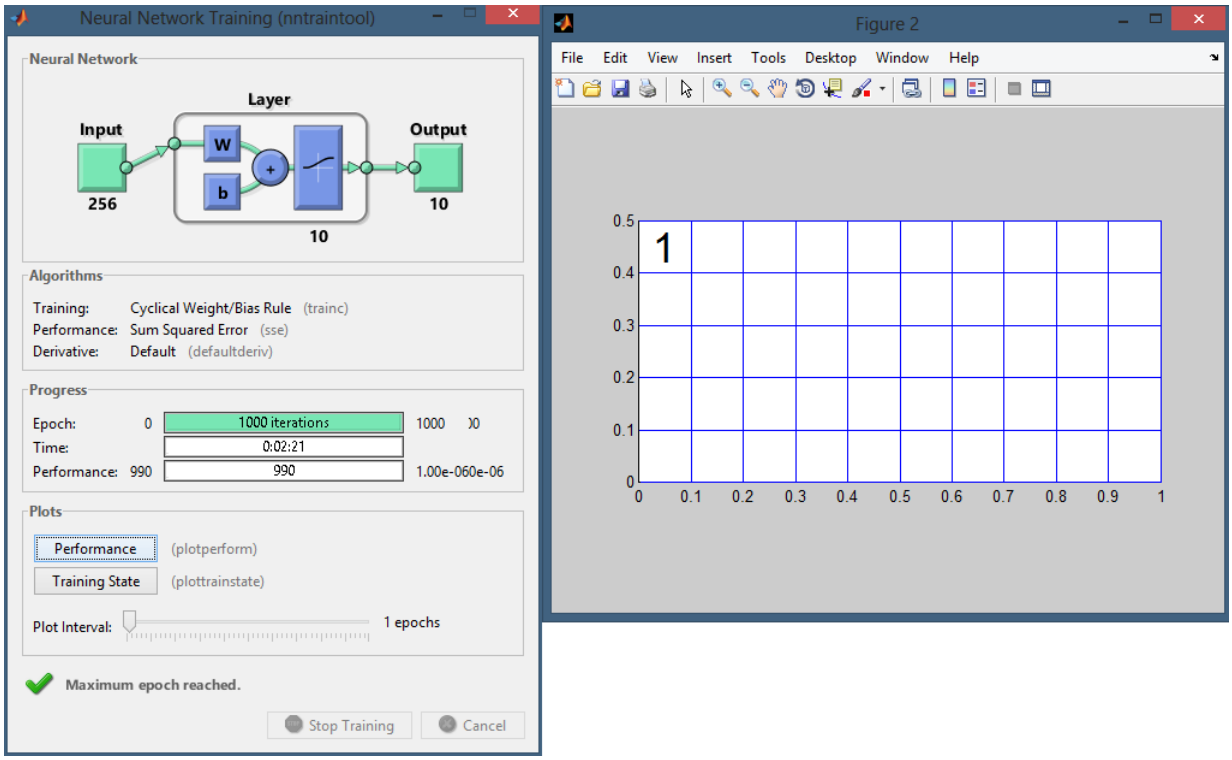

Figure 34: Associative Memory, using the transpose weighing method, Sigmoidal with Gradient descent, 100 train input values, perfect Arial numerals.

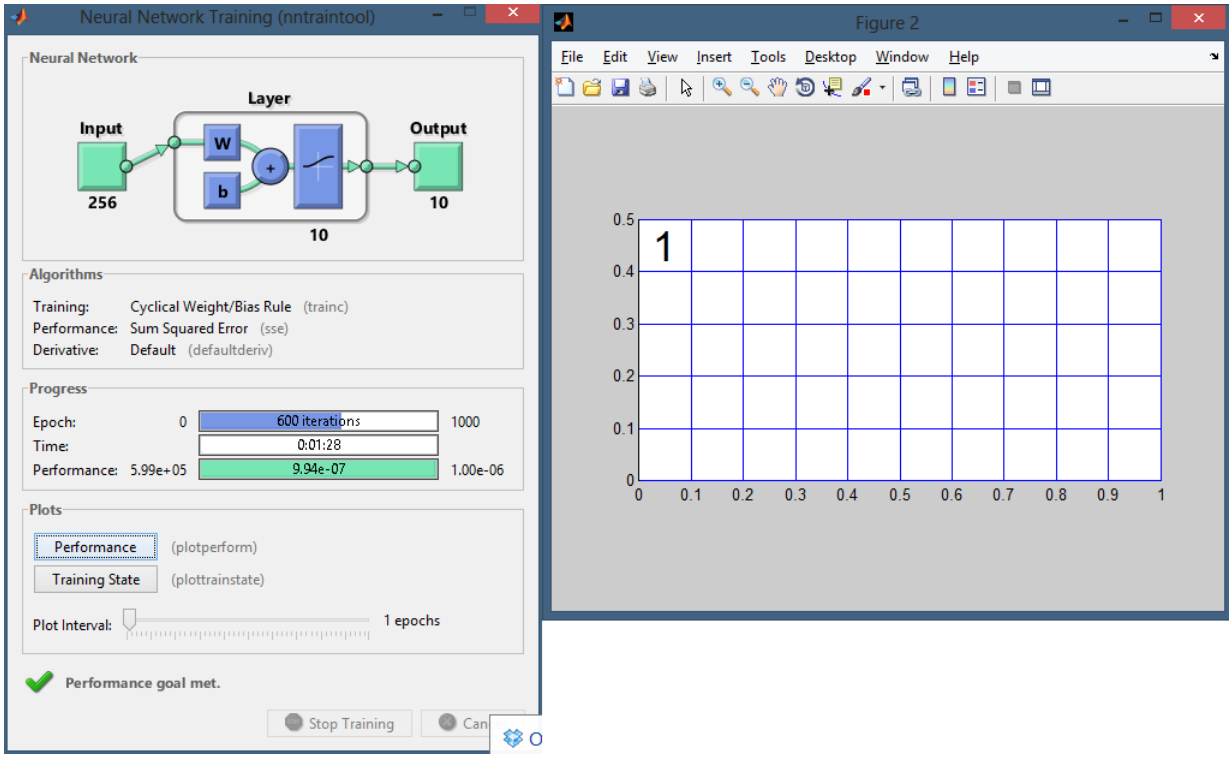

Figure 35: Associative Memory, using the transpose weighing method, Linear with Gradient descent, 100 train input values, perfect Arial numerals.

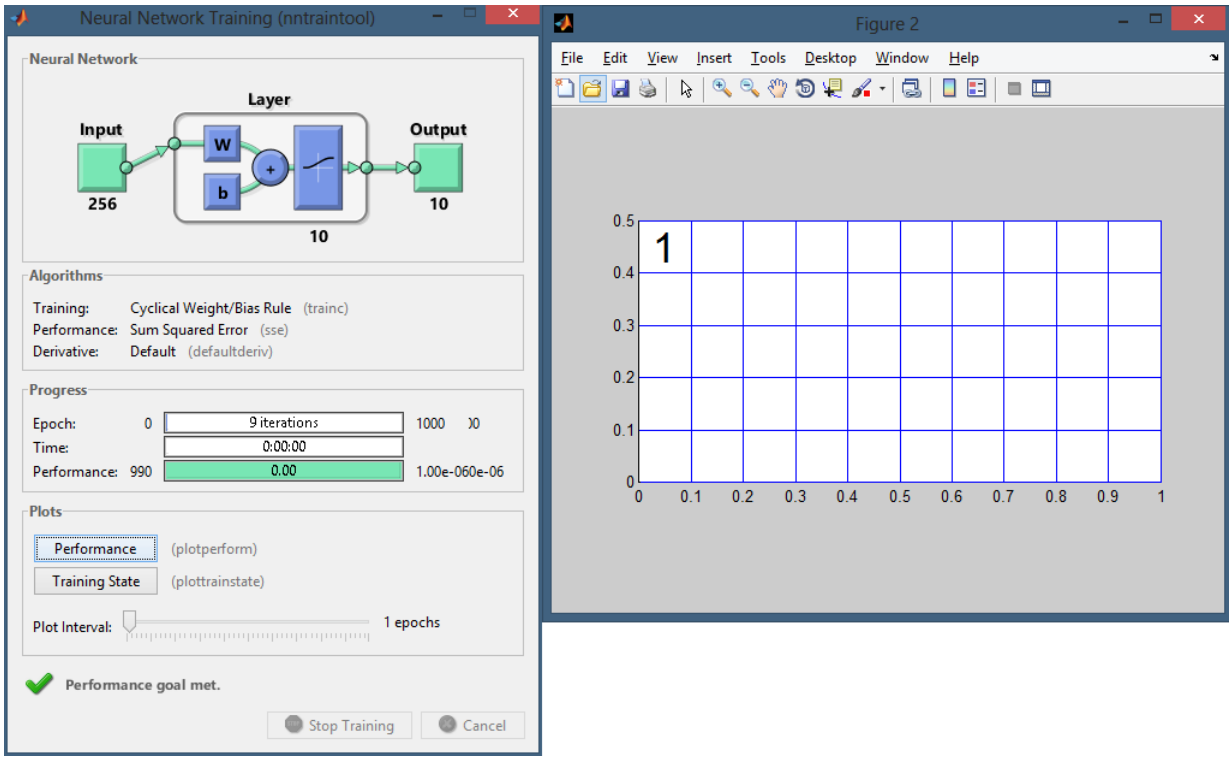

Figure 36: Associative Memory, using the transpose weighing method, Hard-Limit with Perceptron, 100 train input values, perfect Arial numerals.

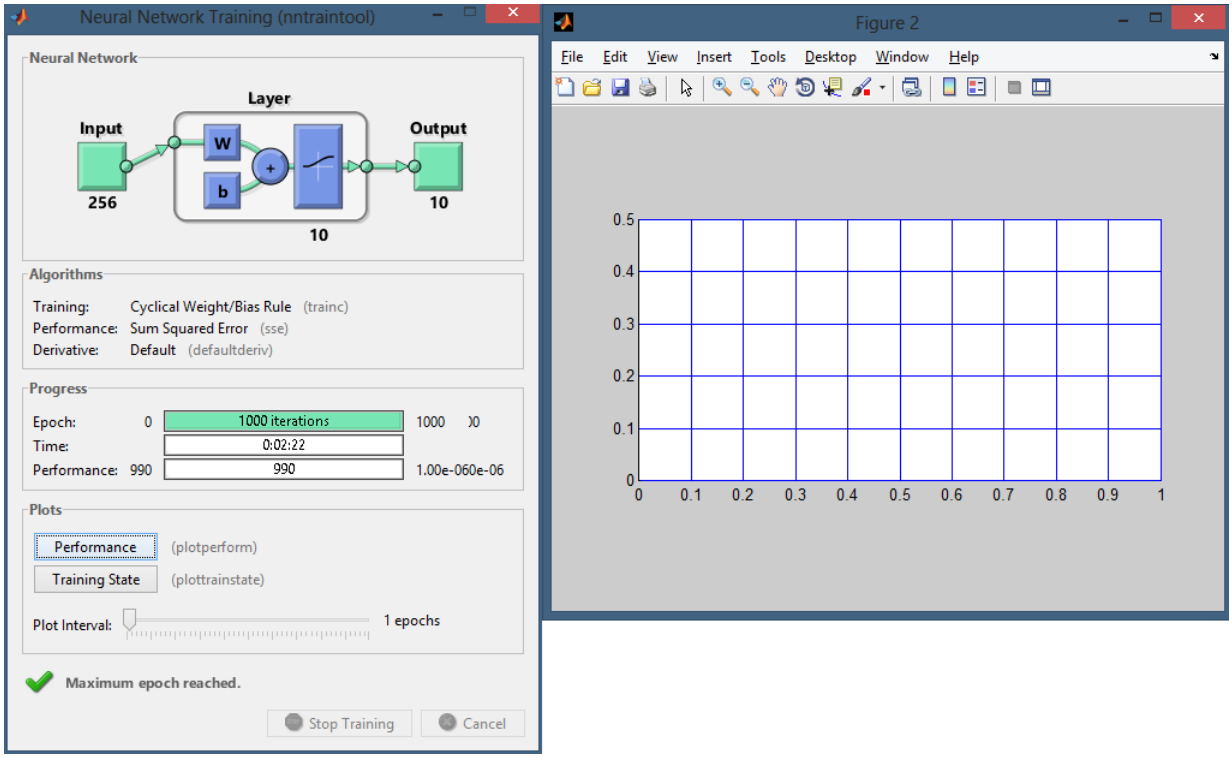

Figure 37: Associative Memory, using the Hebb's rule, Sigmoidal with Gradient descent, 100 train input values, perfect Arial numerals.

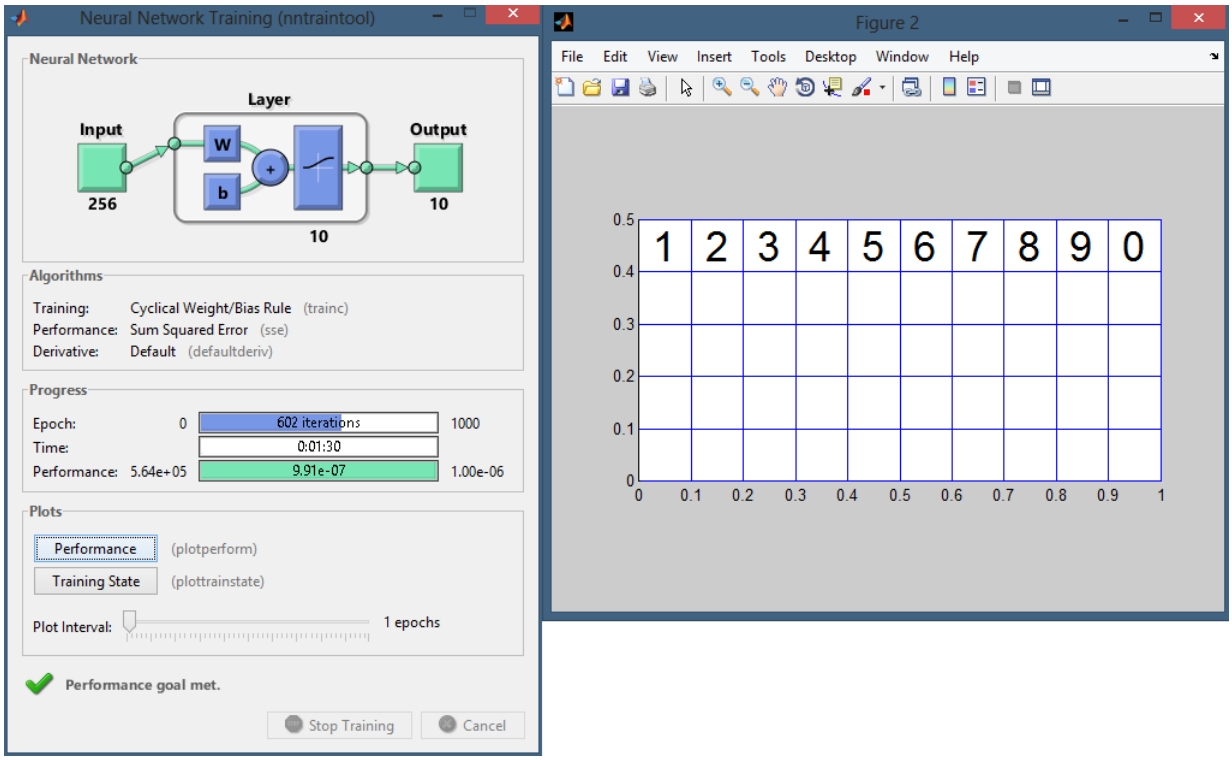

Figure 38: Associative Memory, using the Hebb's rule, Linear with Gradient descent, 100 train input values, perfect Arial numerals.

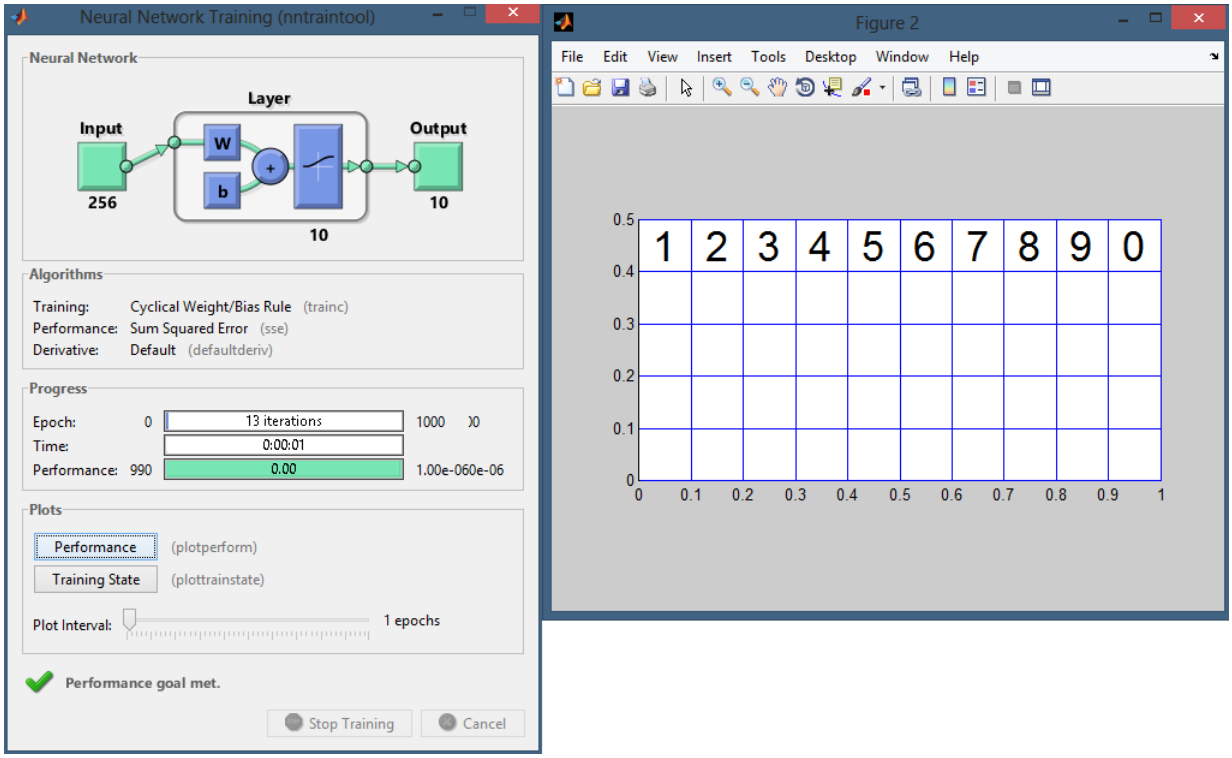

Figure 39: Associative Memory, using the Hebb's rule, Hard-Limit with Perceptron, 100 train input values, perfect Arial numerals.

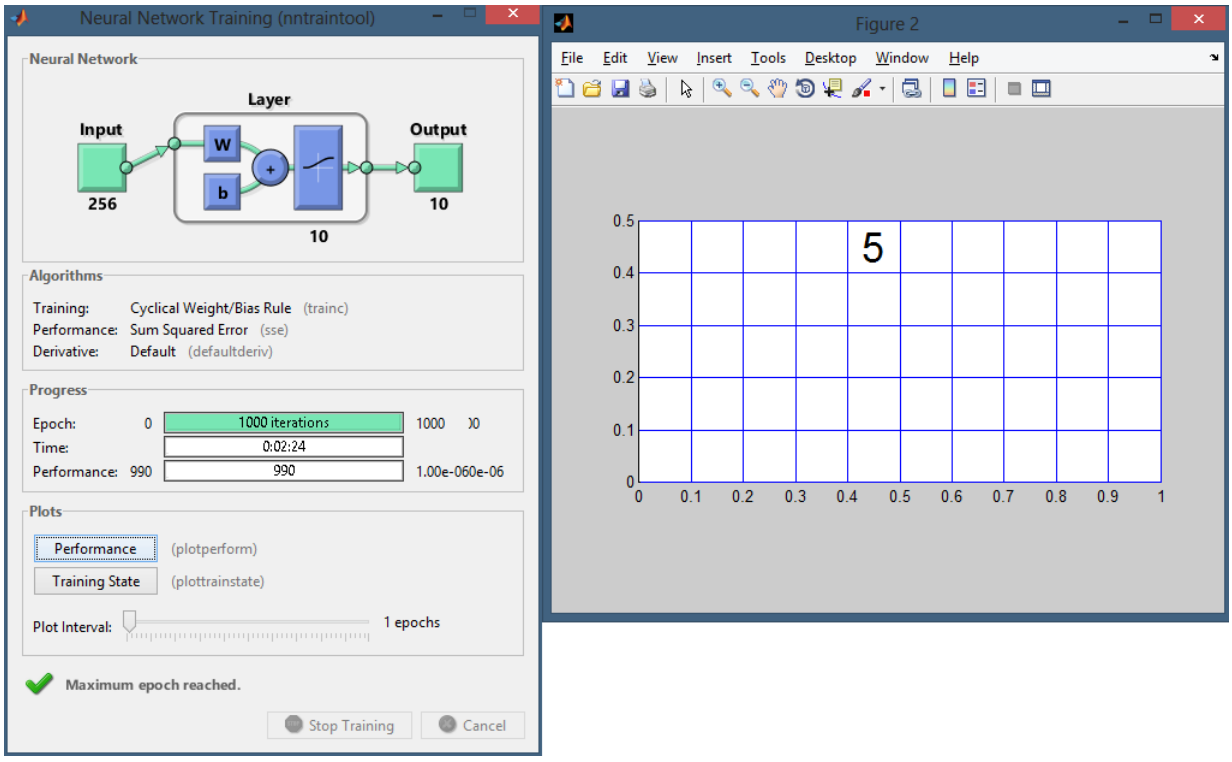

Figure 40: No Associative Memory, Sigmoidal with Gradient descent, 100 train input values, perfect Arial numerals.

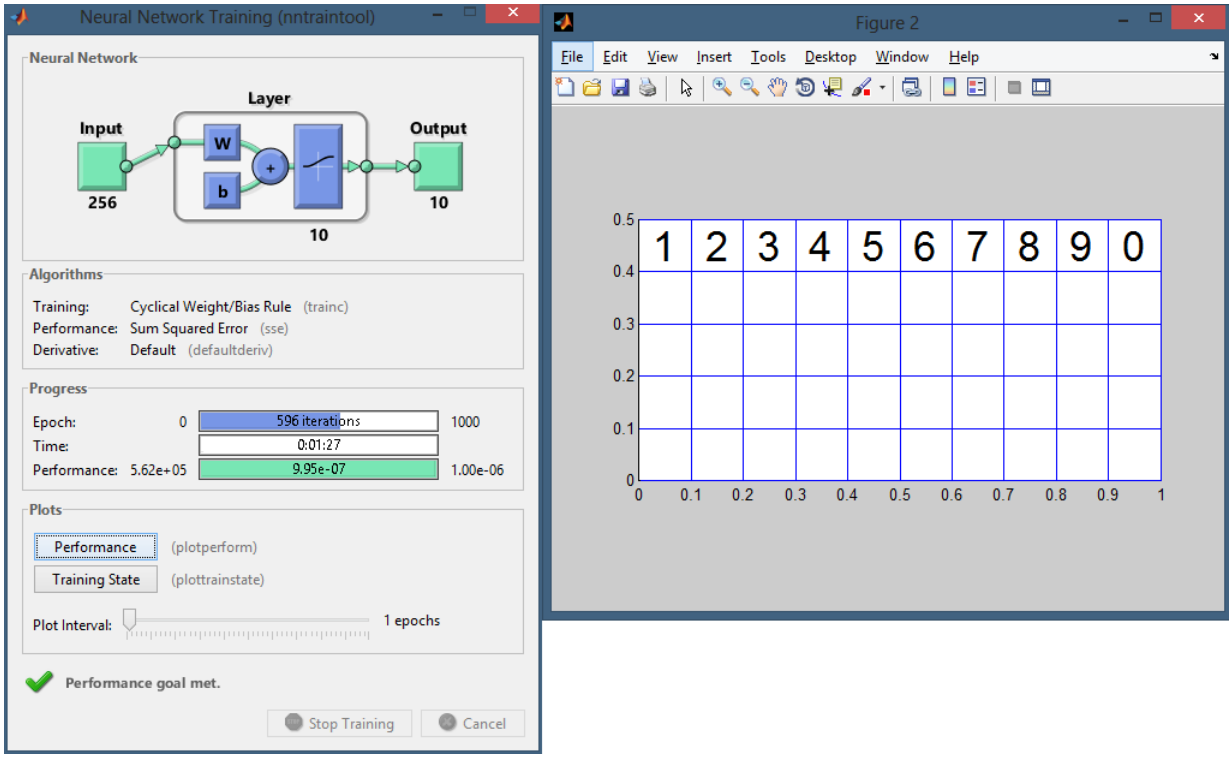

Figure 41: No Associative Memory, Linear with Gradient descent, 100 train input values, perfect Arial numerals.

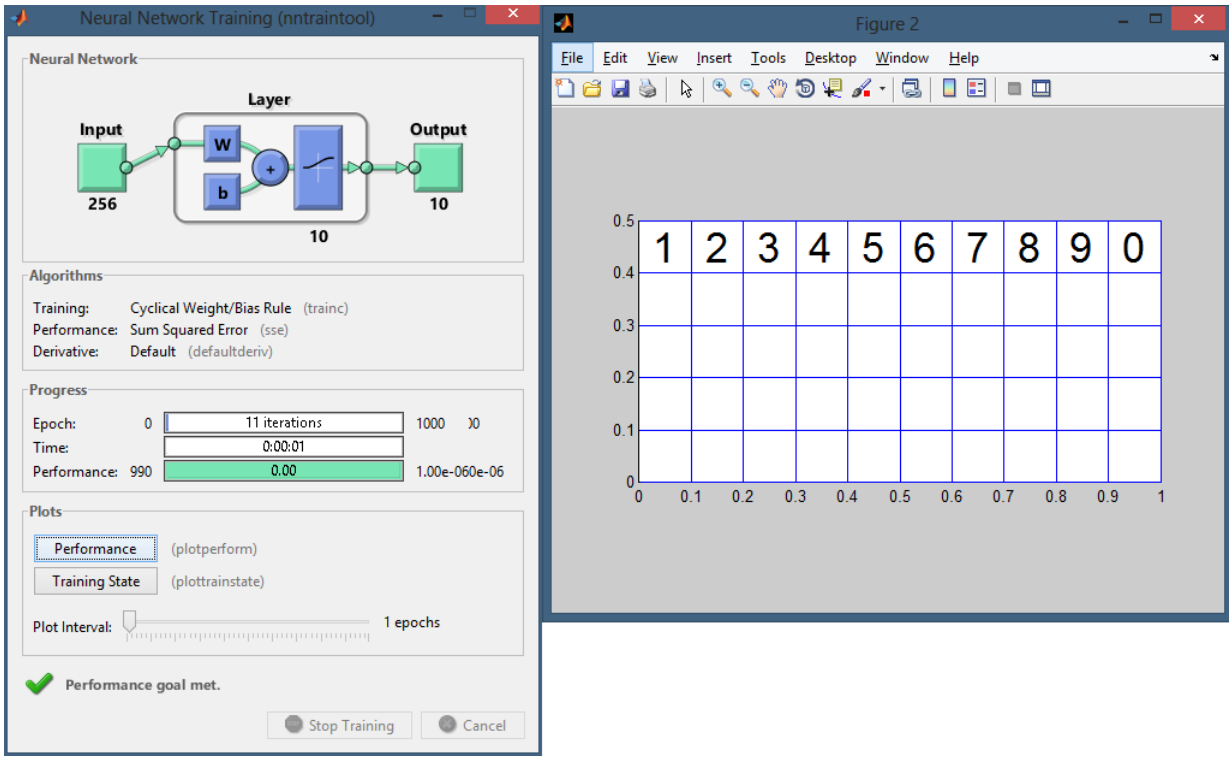

Figure 42: No Associative Memory, Hard-Limit with Perceptron, 100 train input values, perfect Arial numerals.

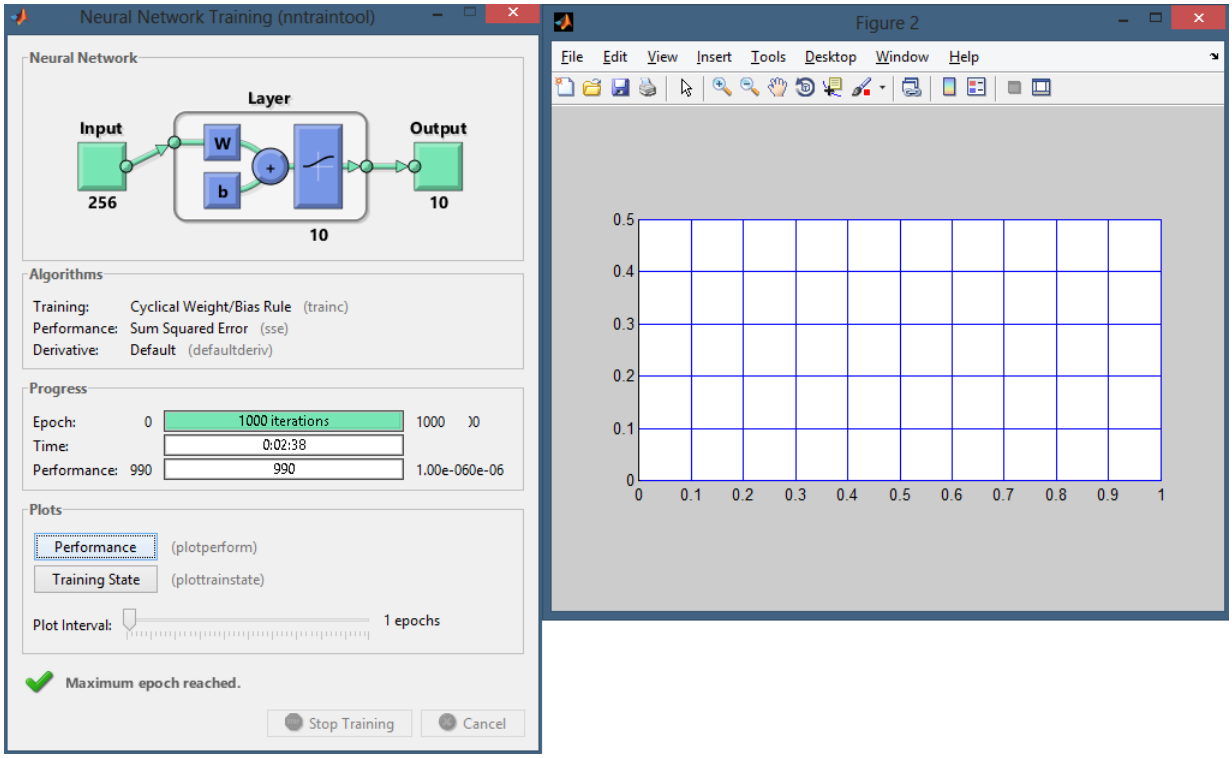

Figure 43: No Associative Memory, Sigmoidal with Gradient descent, 100 train input values, perfect Arial numerals.

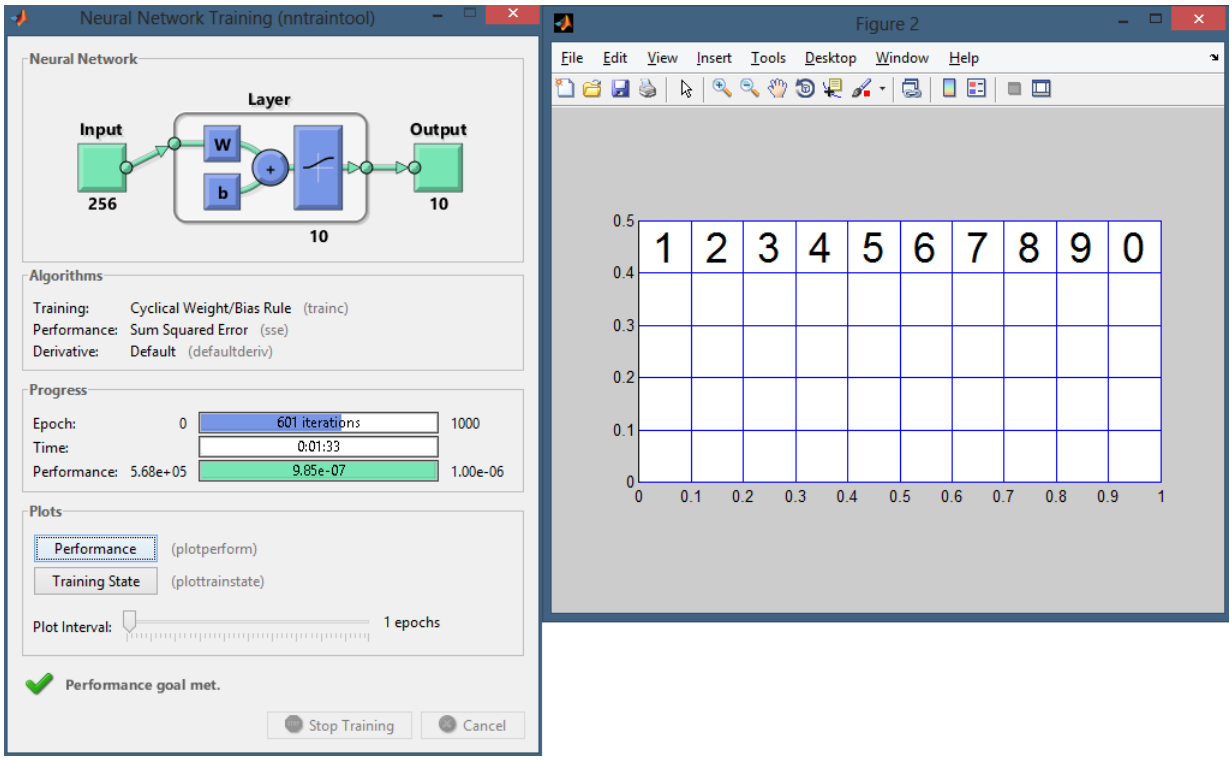

Figure 44: No Associative Memory, Linear with Gradient descent, 100 train input values, perfect Arial numerals.

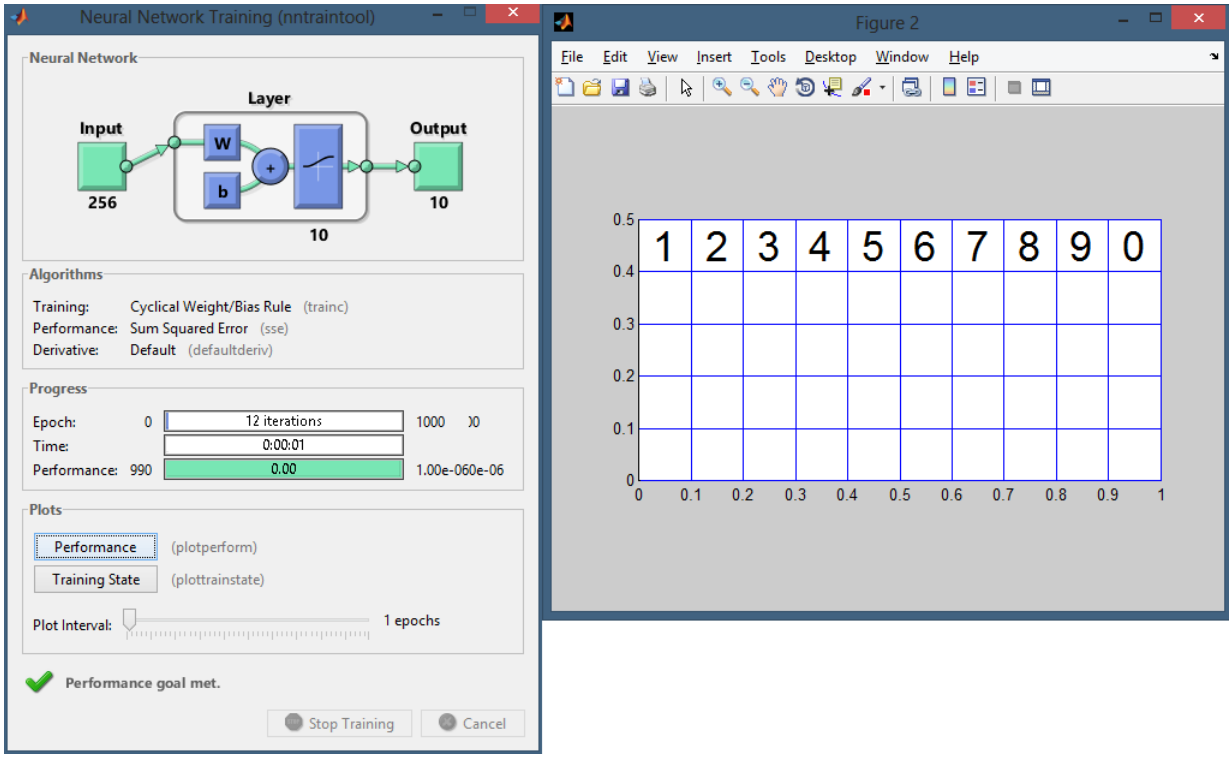

Figure 45: No Associative Memory, Hard-Limit with Perceptron, 100 train input values, perfect Arial numerals.

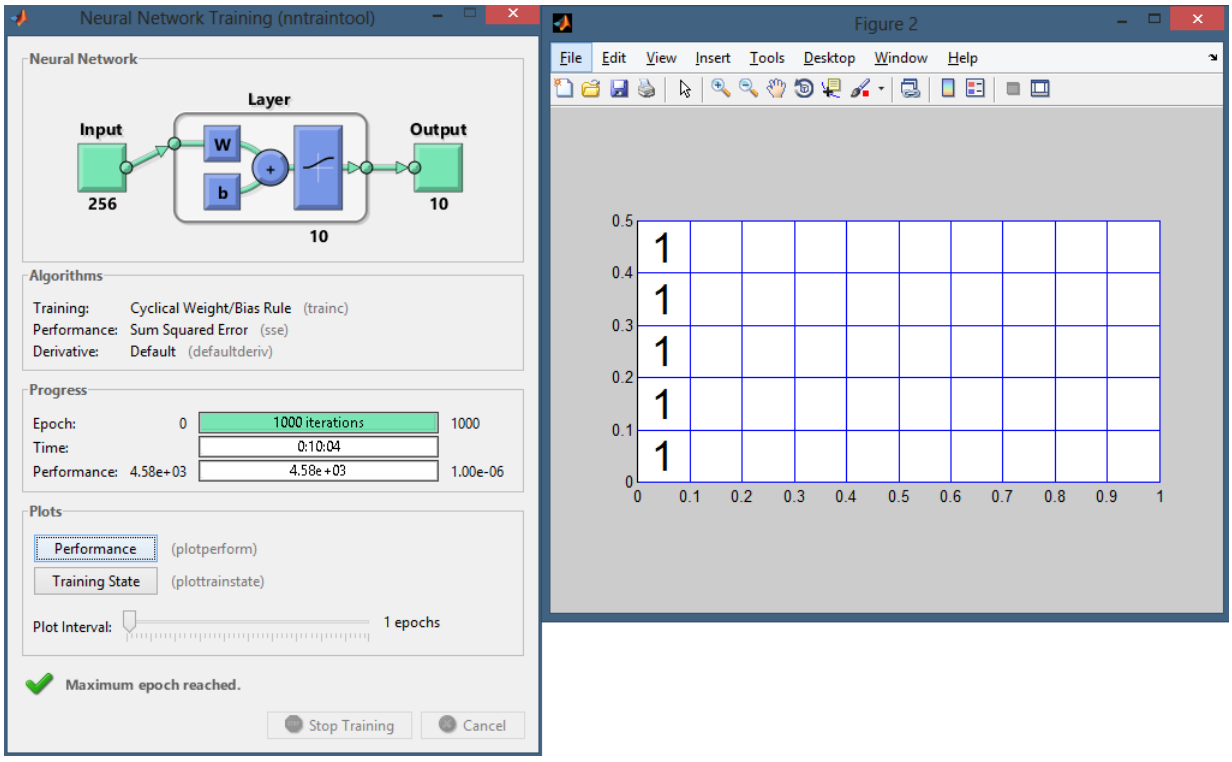

Figure 46: Associative Memory, using the transpose weighing method, Sigmoidal with Gradient descent, 500 train input values, perfect Arial numerals.

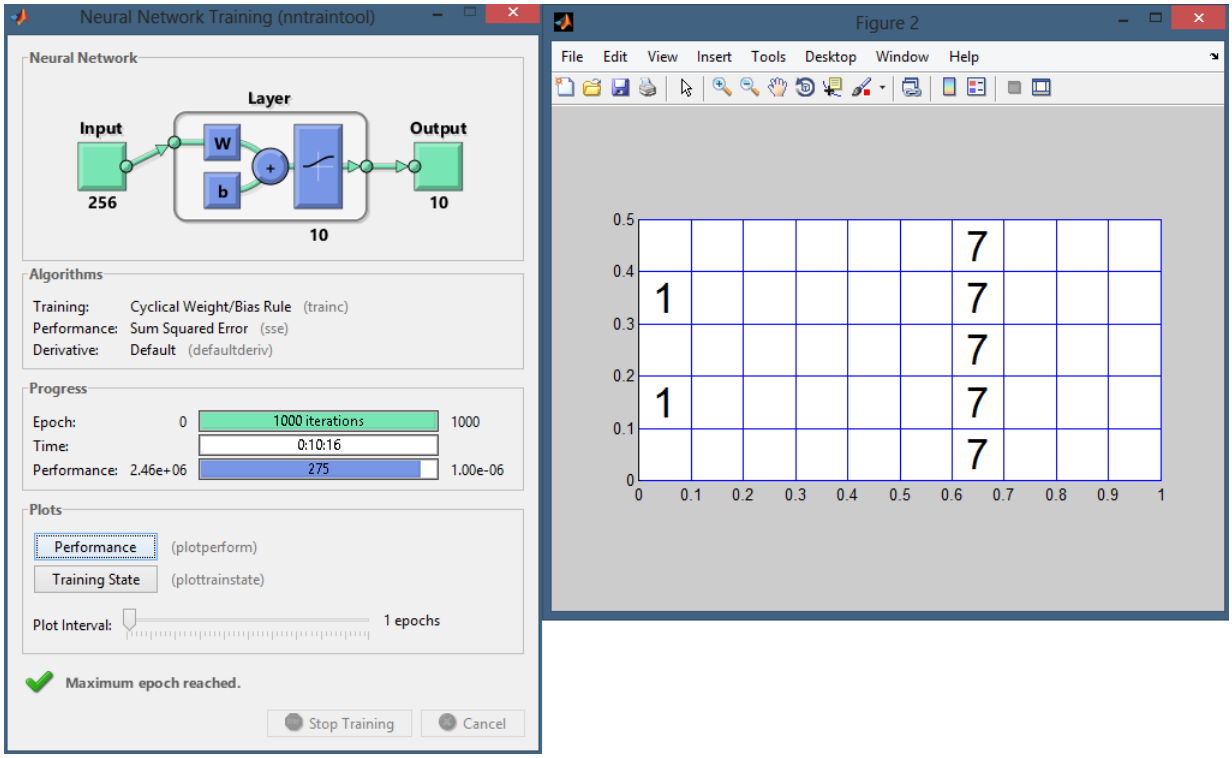

Figure 47: Associative Memory, using the transpose weighing method, Linear with Gradient descent, 500 train input values, perfect Arial numerals.

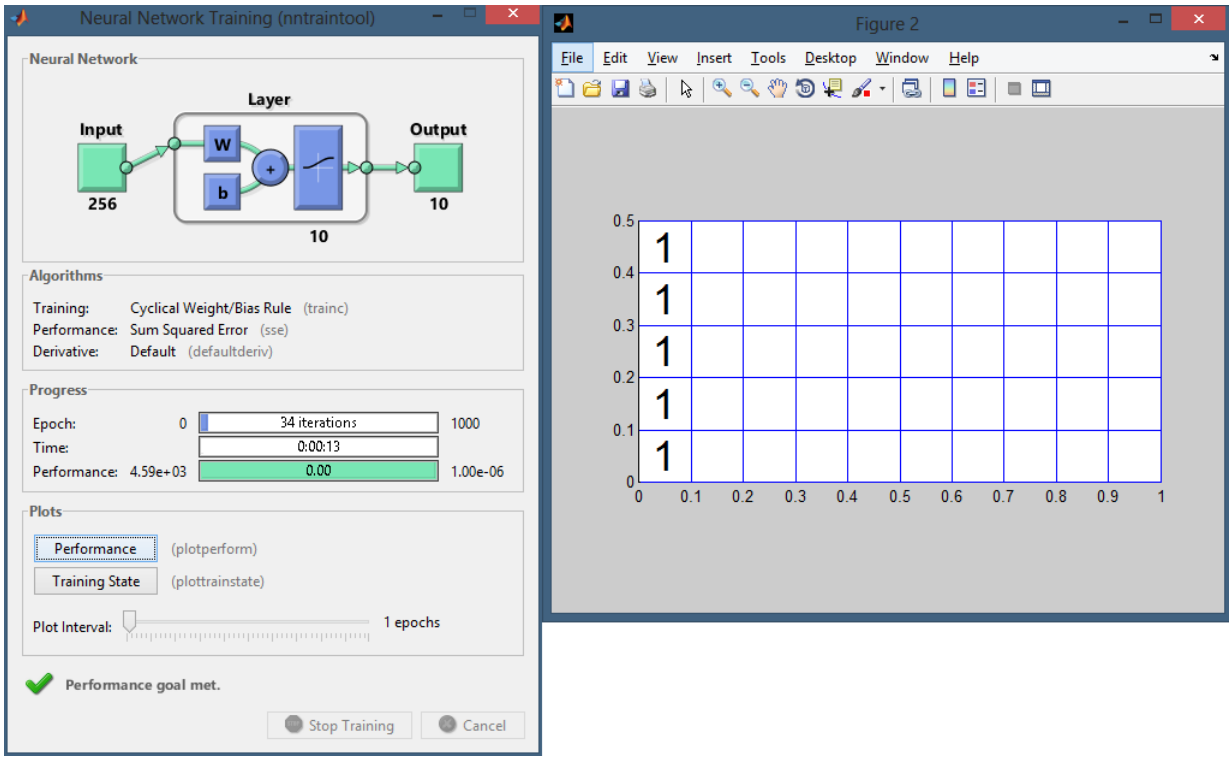

Figure 48: Associative Memory, using the transpose weighing method, Hard-Limit with Perceptron, 500 train input values, perfect Arial numerals.

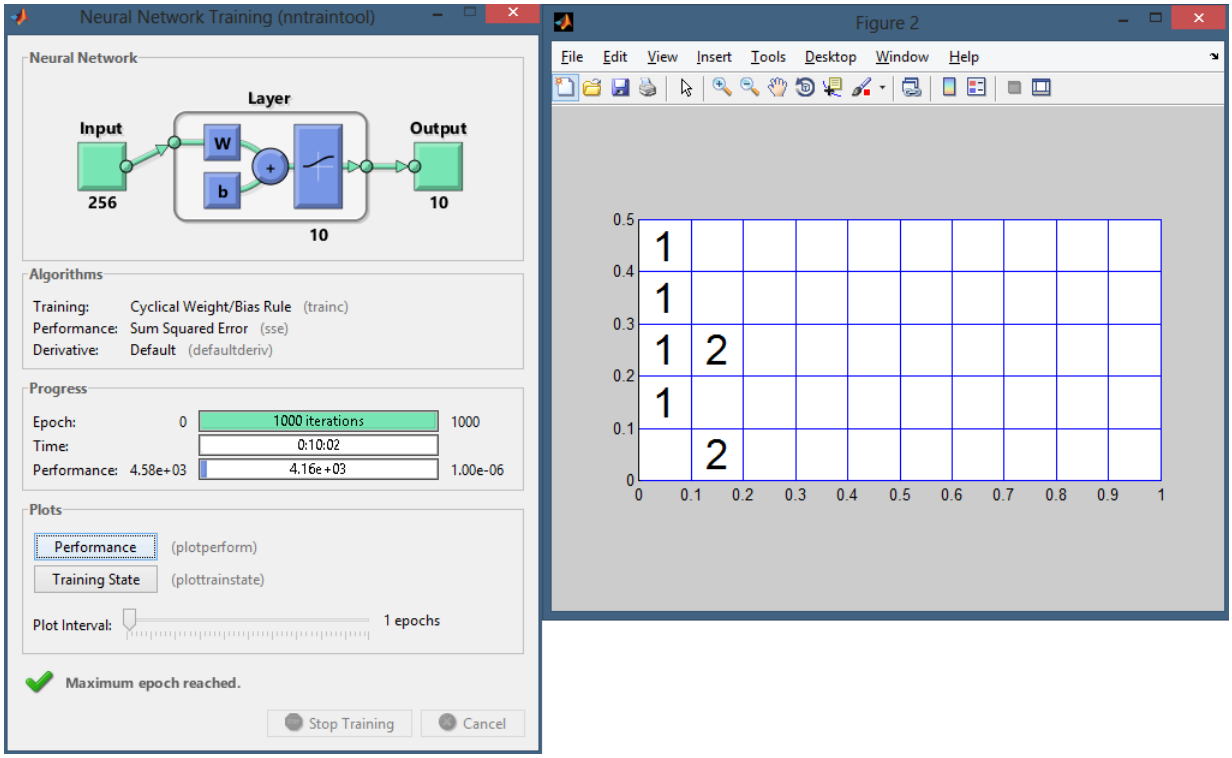

Figure 49: Associative Memory, using the Hebb's rule, Sigmoidal with Gradient descent, 500 train input values, perfect Arial numerals.

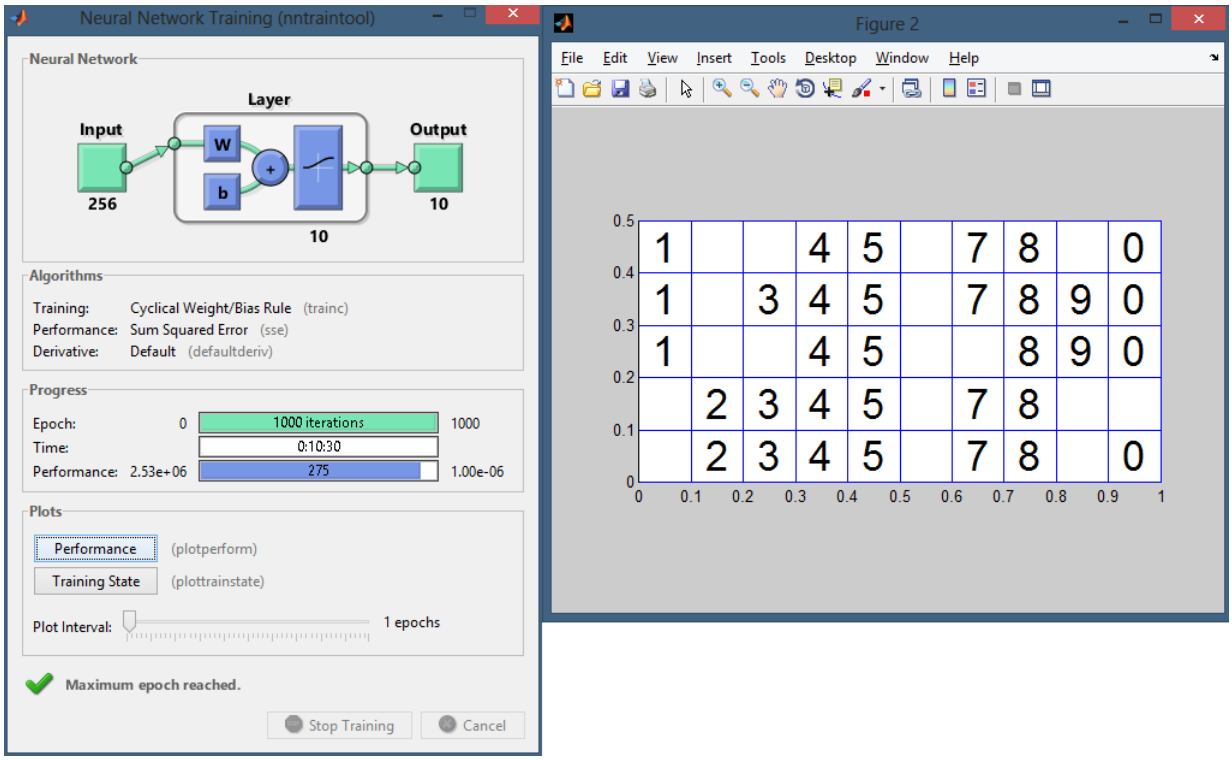

Figure 50: Associative Memory, using the Hebb's rule, Linear with Gradient descent, 500 train input values, perfect Arial numerals.

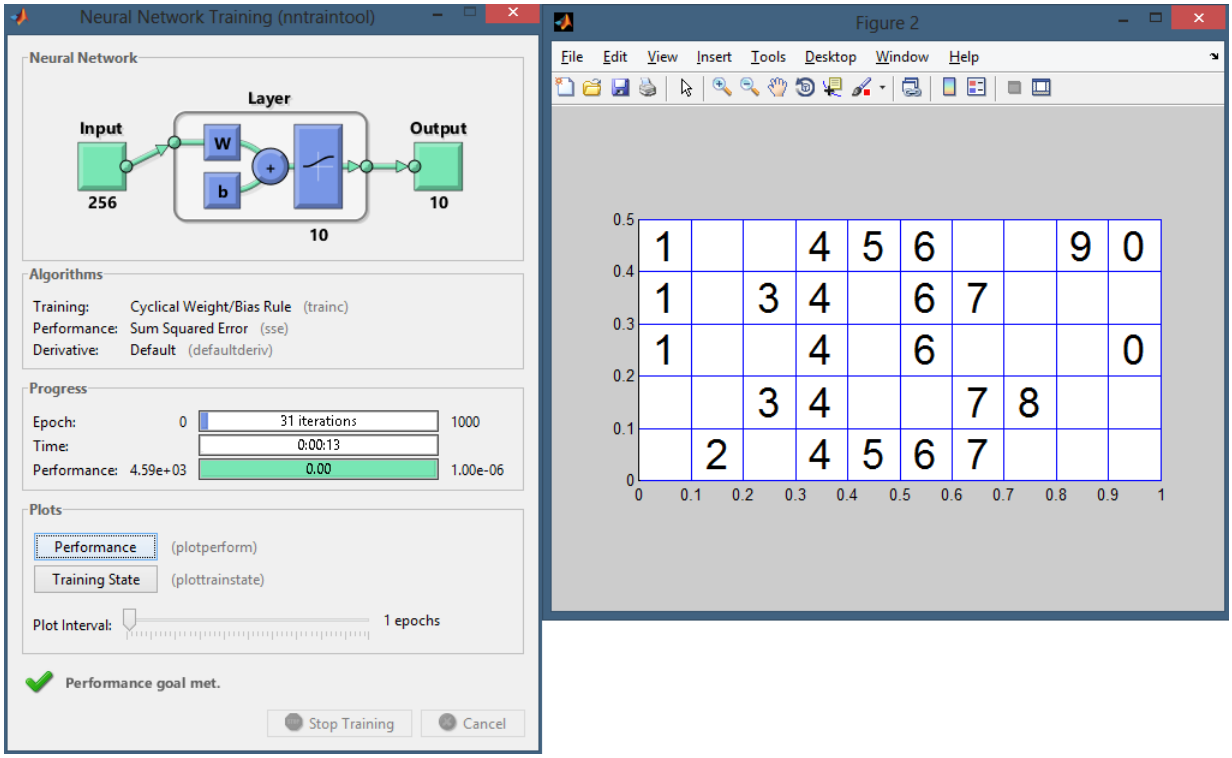

Figure 51: Associative Memory, using the Hebb's rule, Hard-Limit with Perceptron, 500 train input values, perfect Arial numerals.

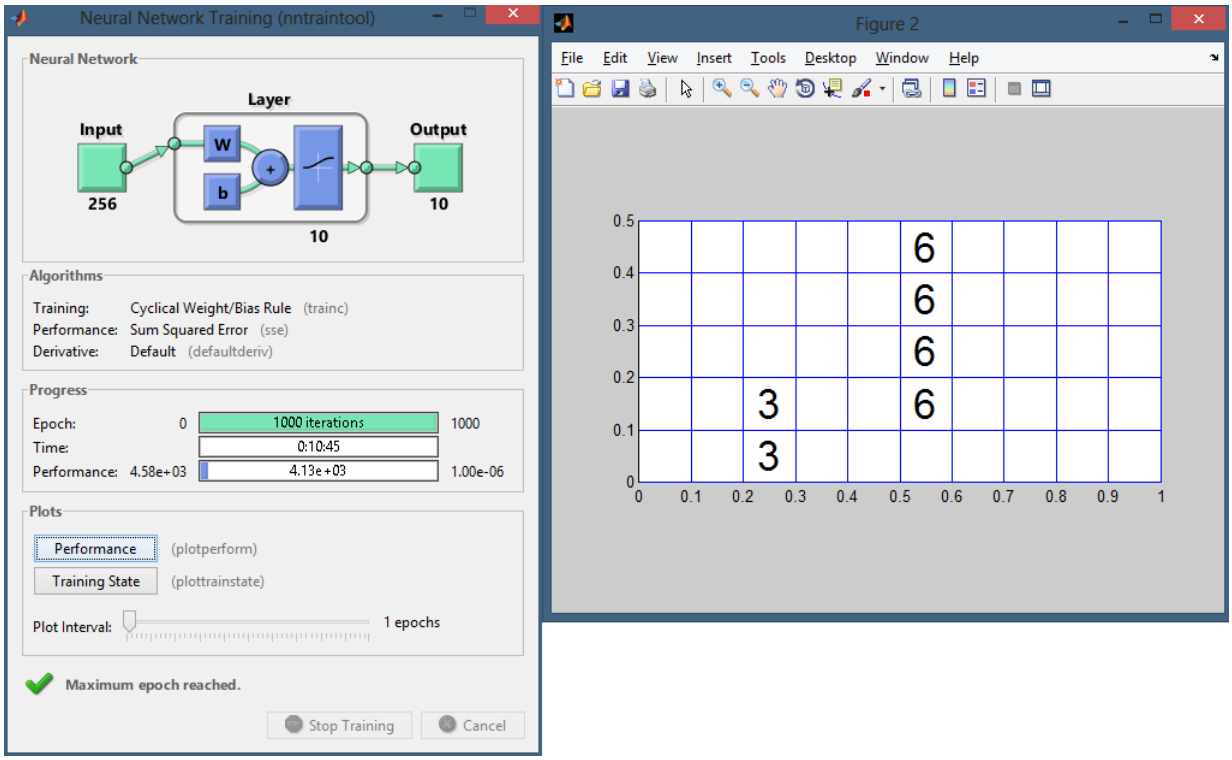

Figure 52: No Associative Memory, Sigmoidal with Gradient descent, 500 train input values, perfect Arial numerals.

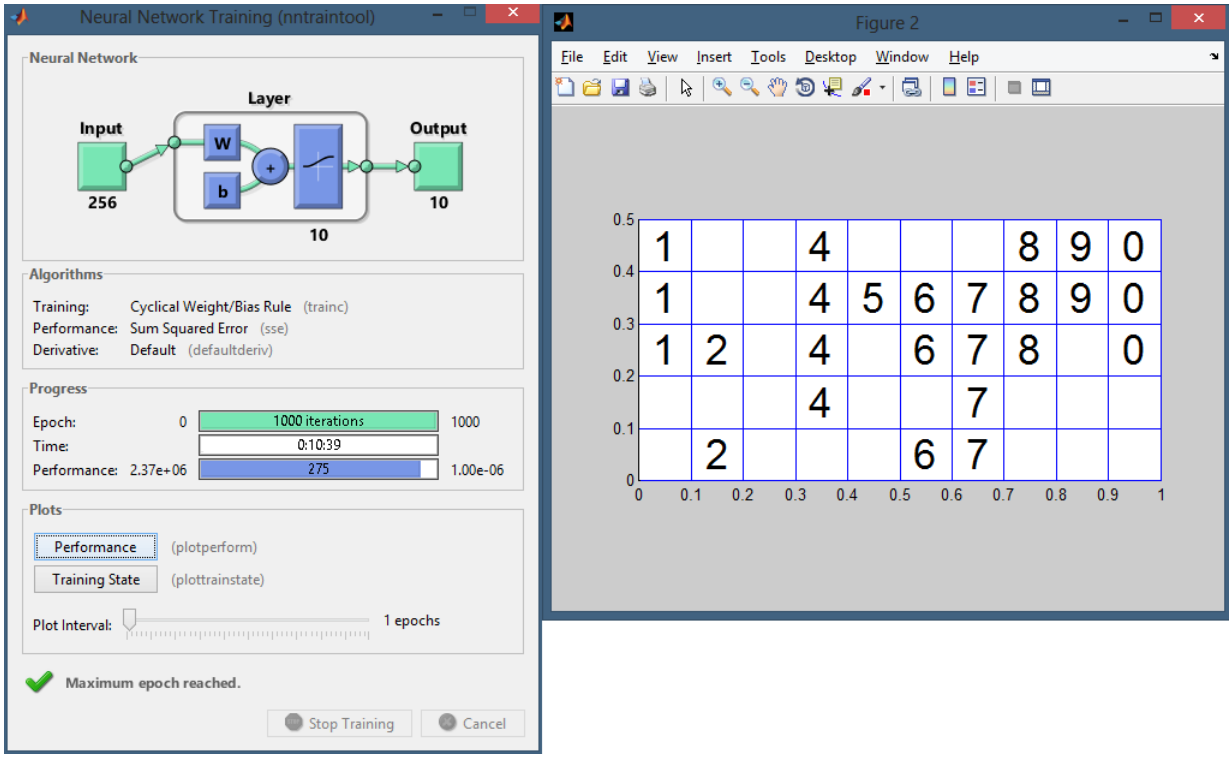

Figure 53: No Associative Memory, Linear with Gradient descent, 500 train input values, perfect Arial numerals.

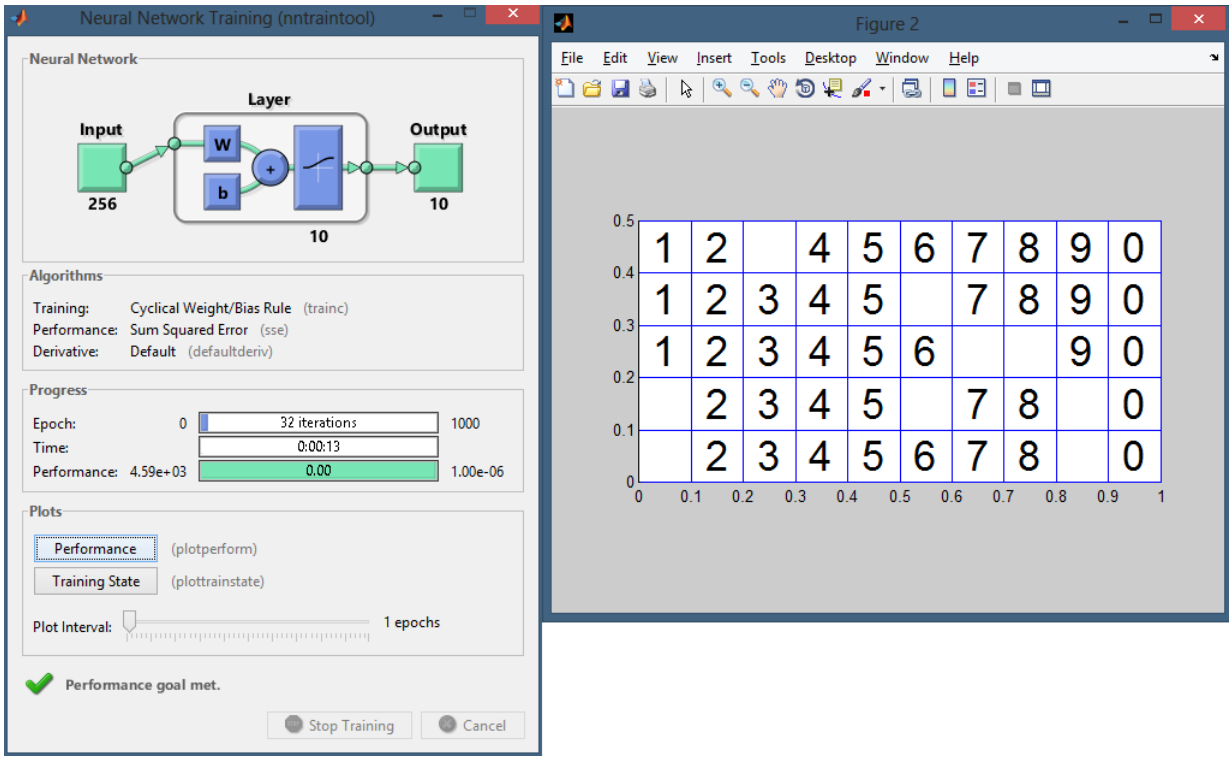

Figure 54: No Associative Memory, Hard-Limit with Perceptron, 500 train input values, perfect Arial numerals.

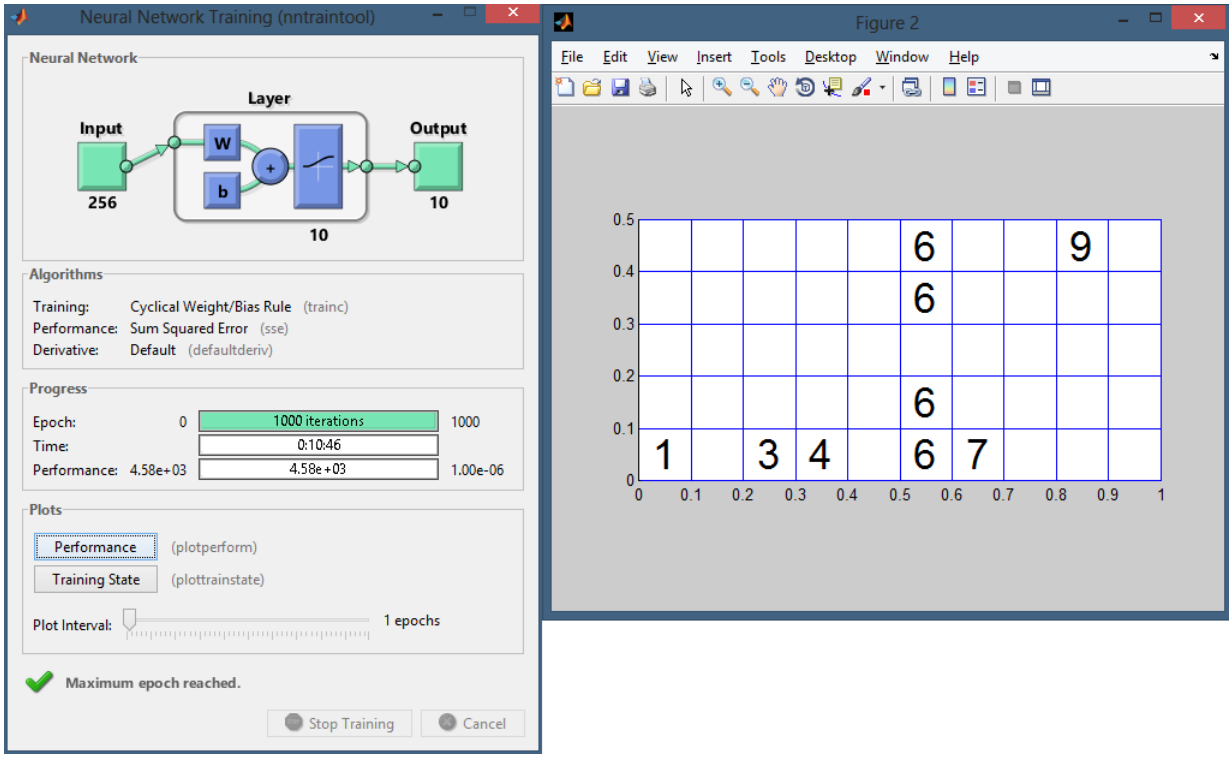

Figure 55: No Associative Memory, Sigmoidal with Gradient descent, 500 train input values, perfect Arial numerals.

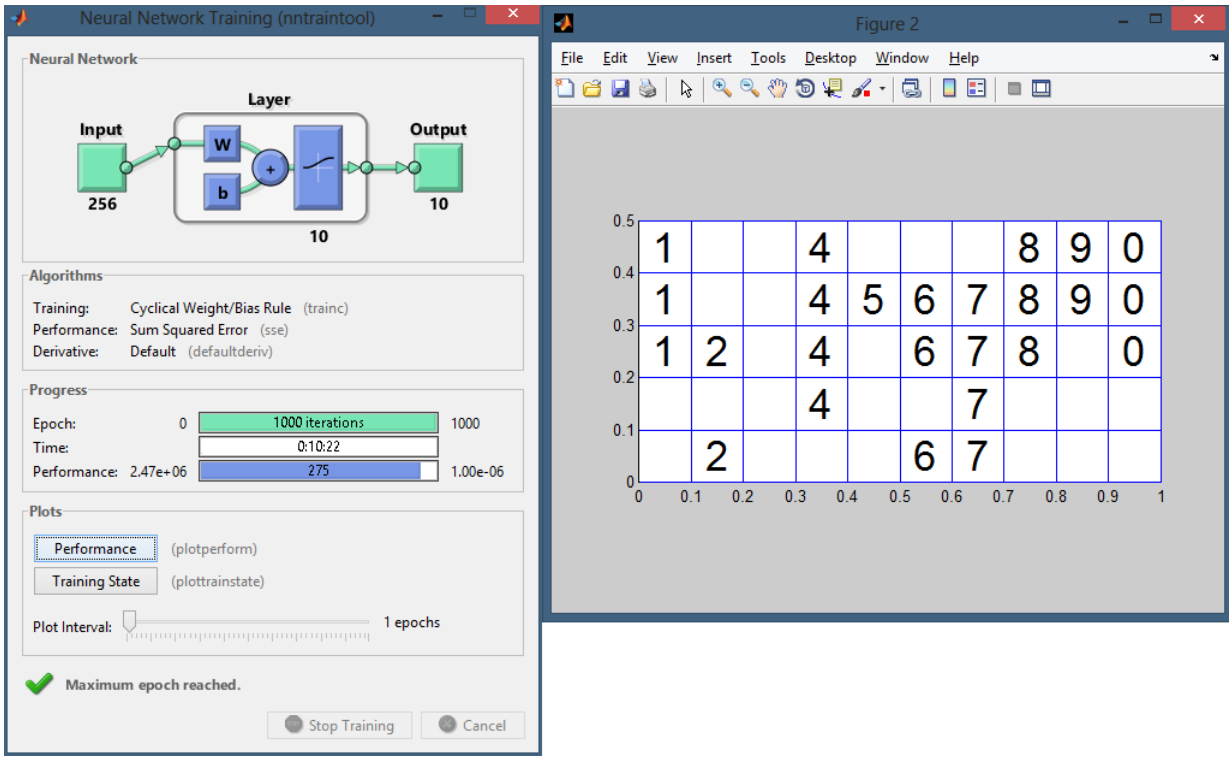

Figure 56: No Associative Memory, Linear with Gradient descent, 500 train input values, perfect Arial numerals.

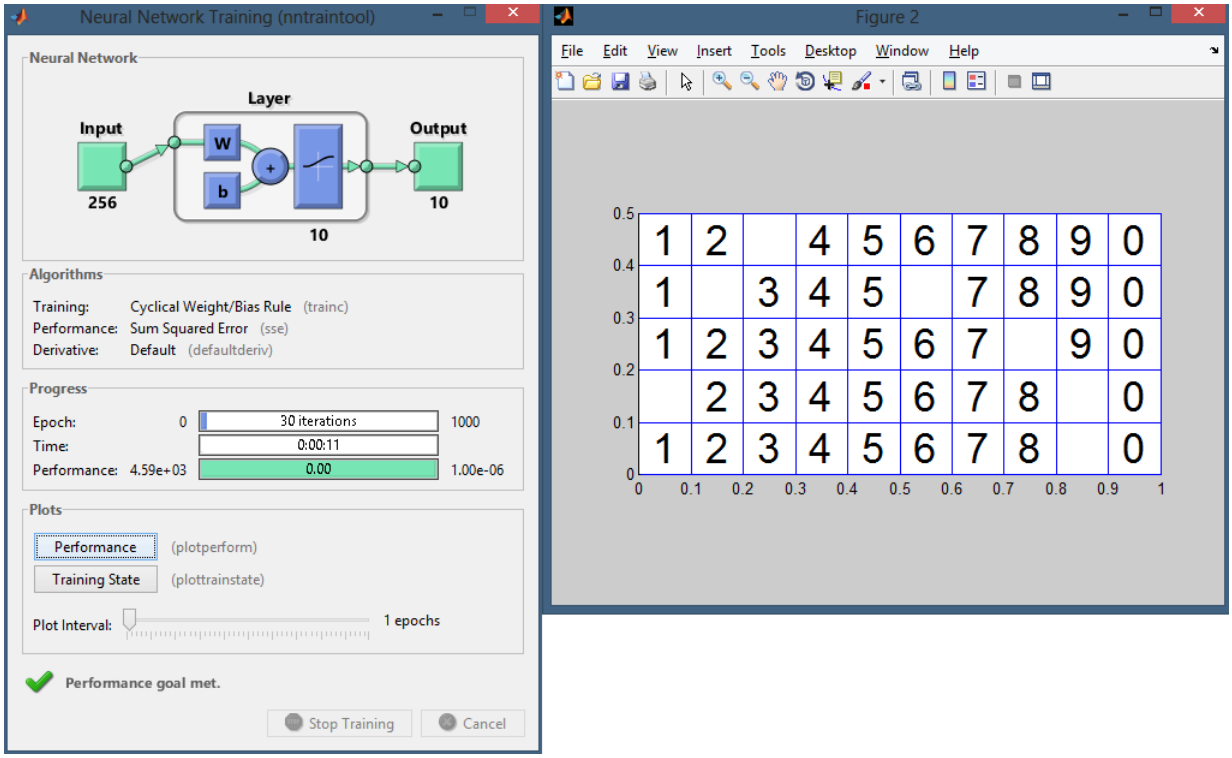

Figure 57: No Associative Memory, Hard-Limit with Perceptron, 500 train input values, perfect Arial numerals.

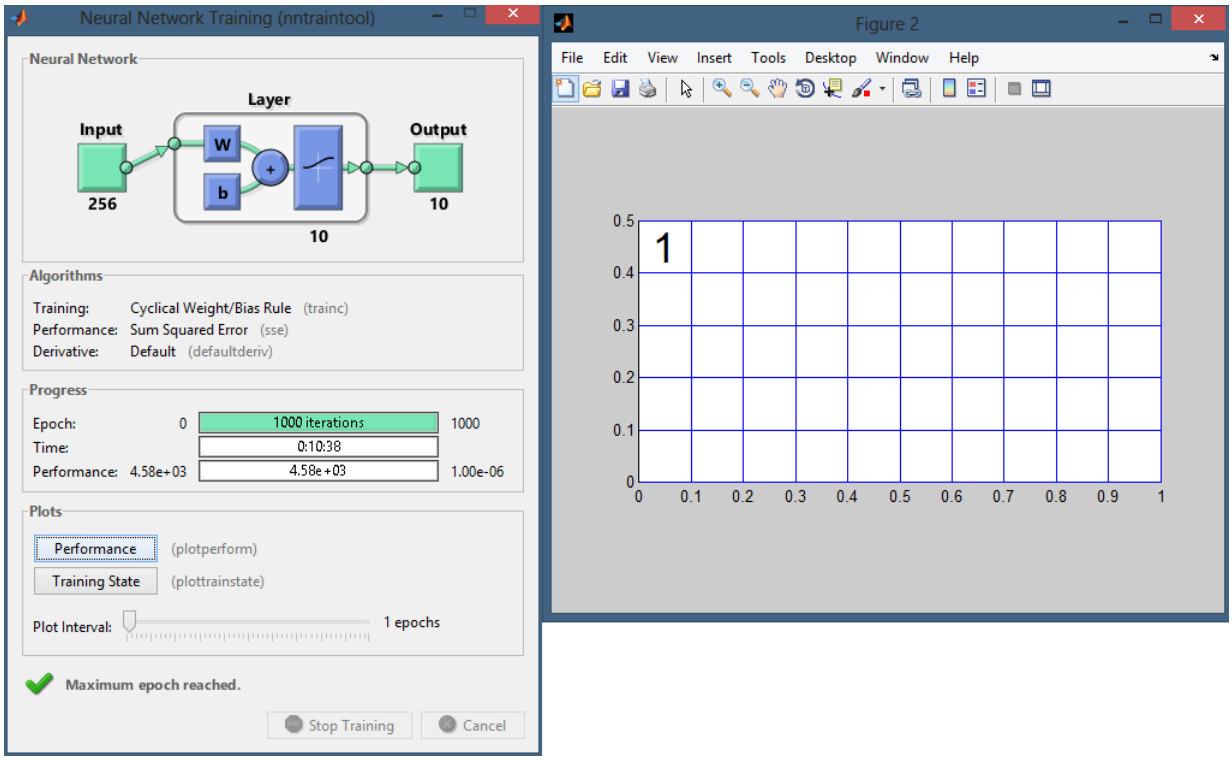

Figure 58: Associative Memory, using the transpose weighing method, Sigmoidal with Gradient descent, 500 train input values, perfect Arial numerals.

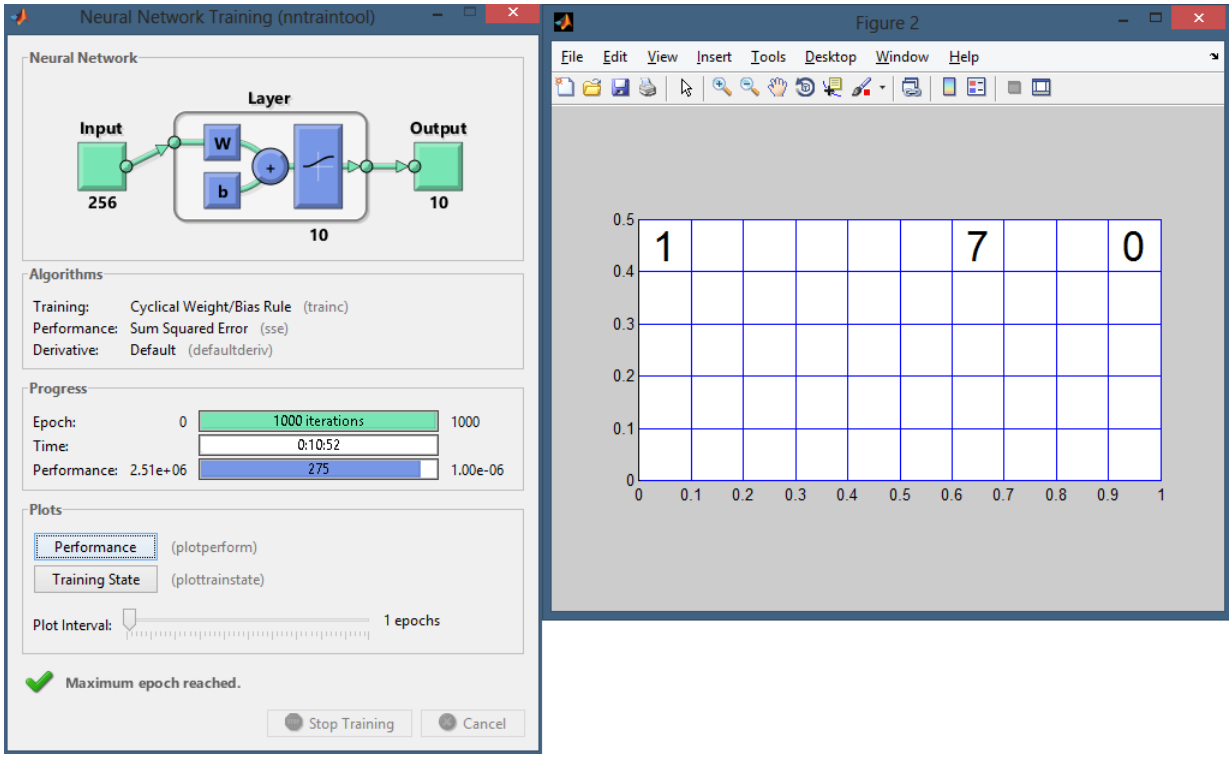

Figure 59: Associative Memory, using the transpose weighing method, Linear with Gradient descent, 500 train input values, perfect Arial numerals.

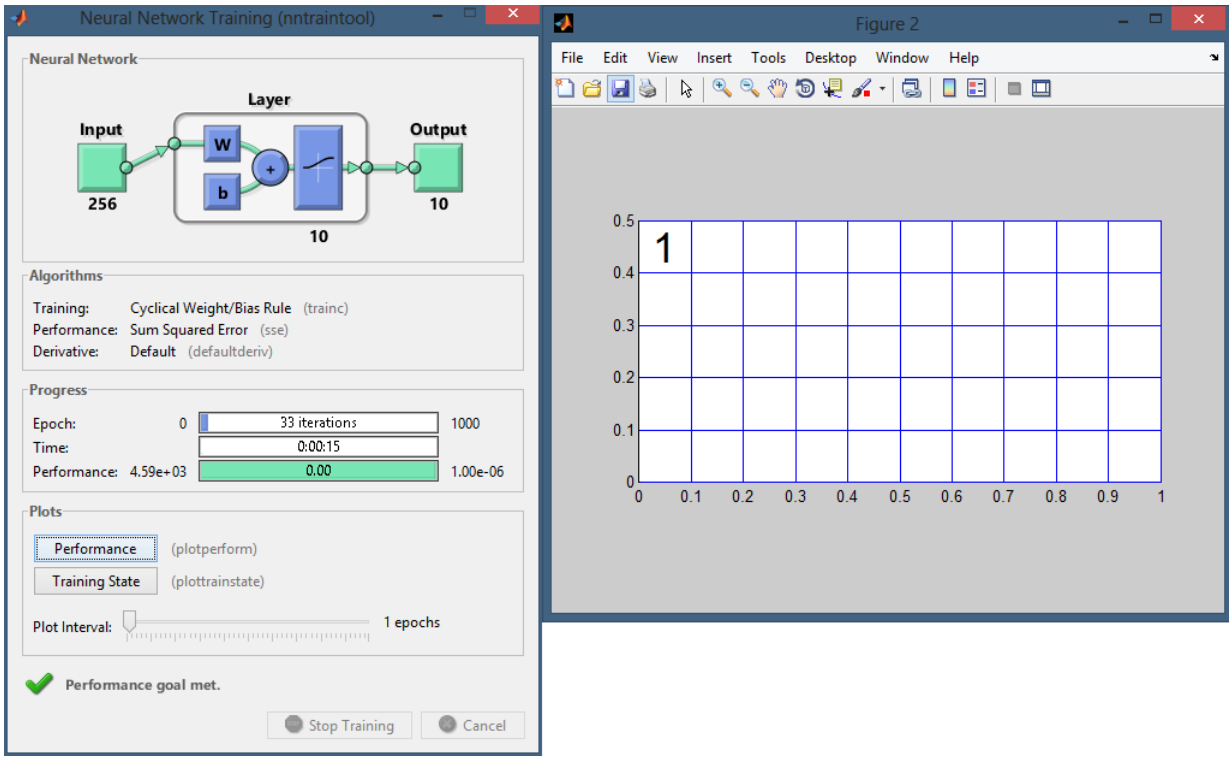

Figure 60: Associative Memory, using the transpose weighing method, Hard-Limit with Perceptron, 500 train input values, perfect Arial numerals.

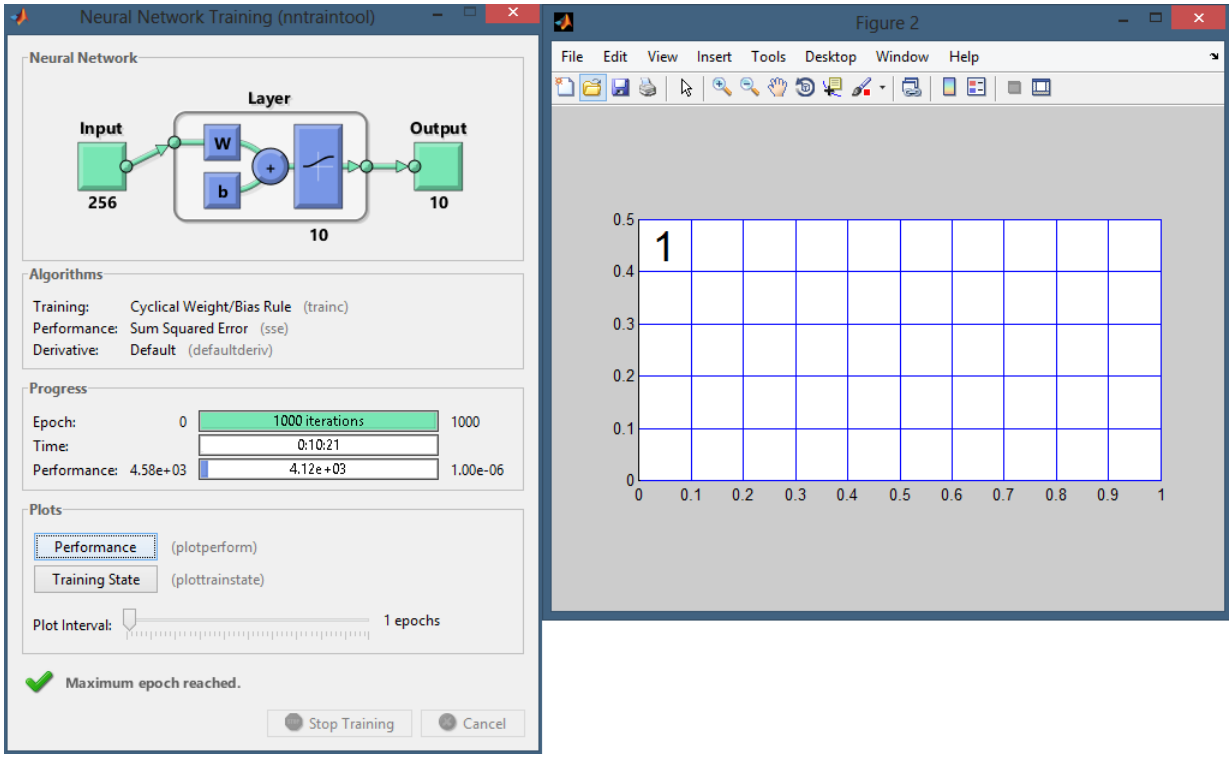

Figure 61: Associative Memory, using the Hebb's rule, Sigmoidal with Gradient descent, 500 train input values, perfect Arial numerals.

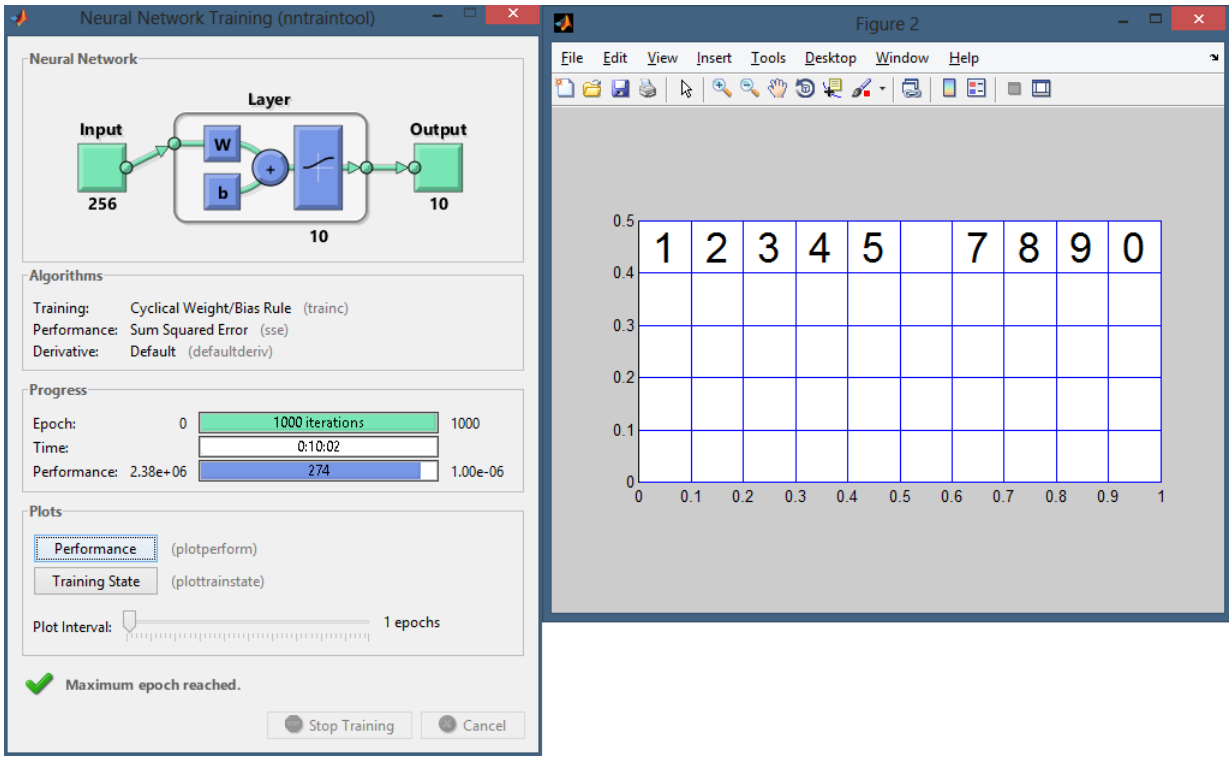

Figure 62: Associative Memory, using the Hebb's rule, Linear with Gradient descent, 500 train input values, perfect Arial numerals.

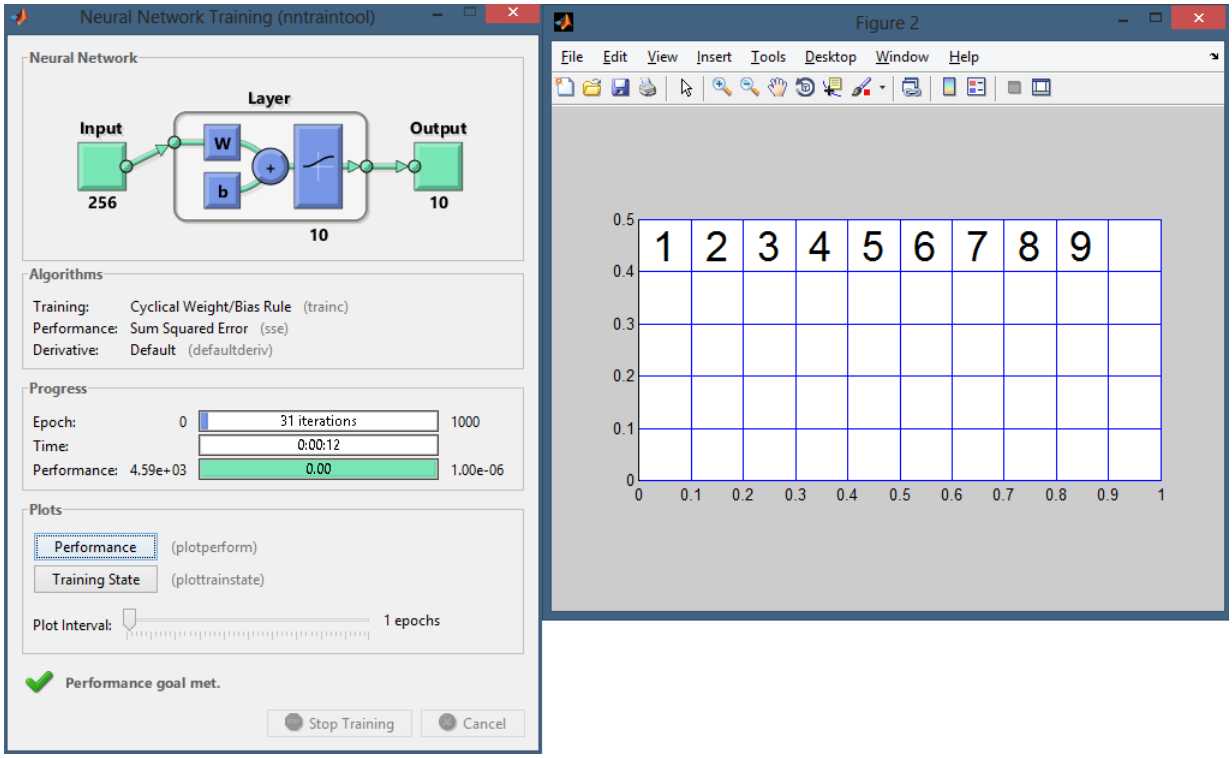

Figure 63: Associative Memory, using the Hebb's rule, Hard-Limit with Perceptron, 500 train input values, perfect Arial numerals.

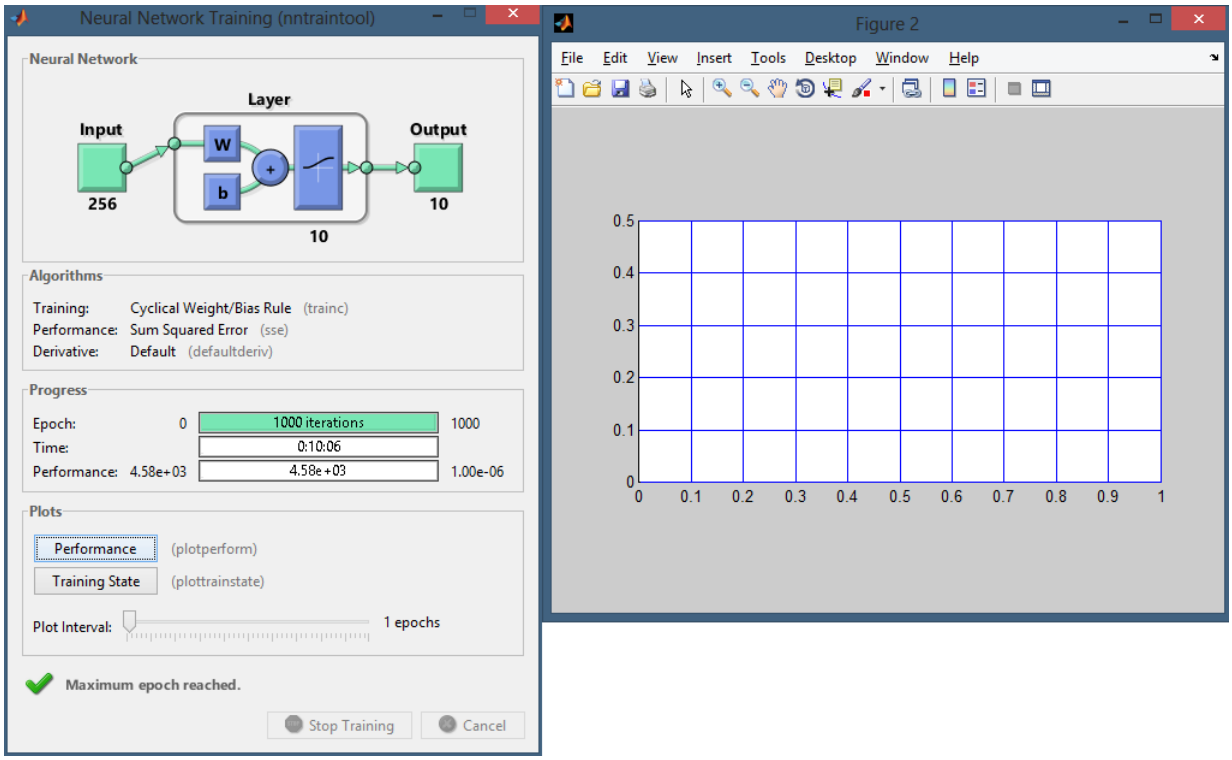

Figure 64: No Associative Memory, Sigmoidal with Gradient descent, 500 train input values, perfect Arial numerals.

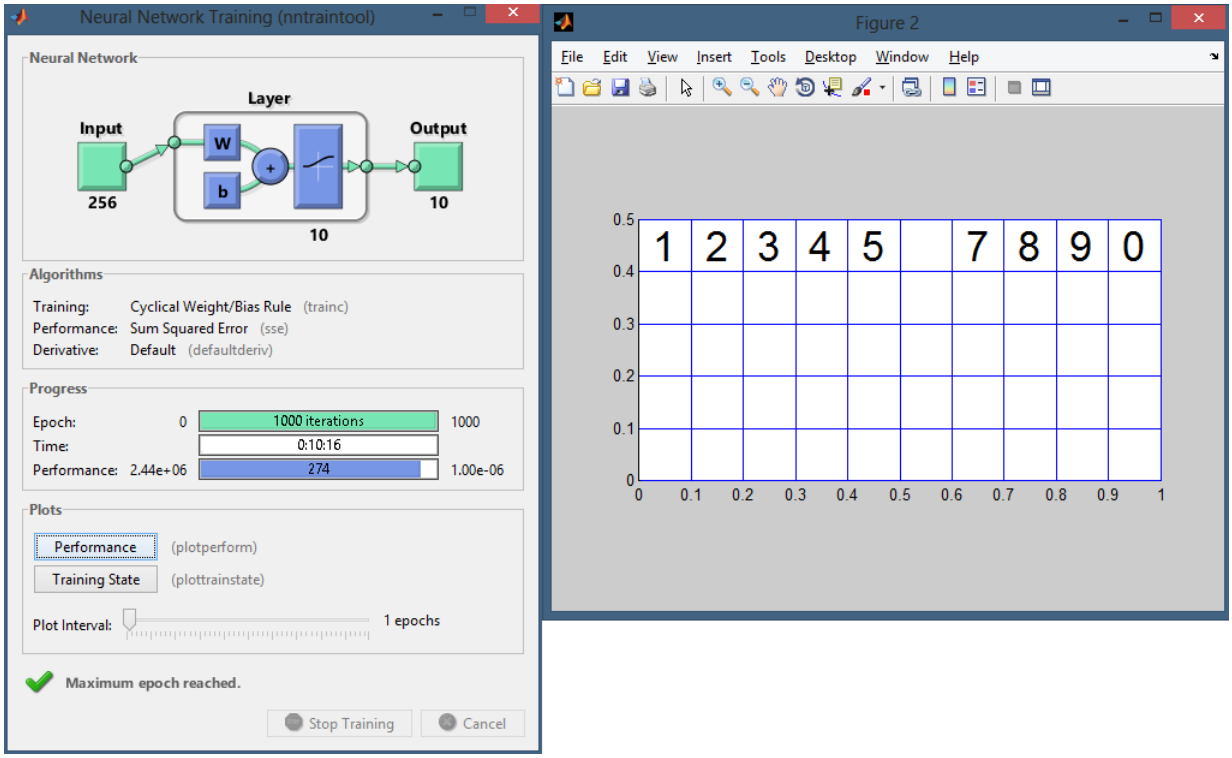

Figure 65: No Associative Memory, Linear with Gradient descent, 500 train input values, perfect Arial numerals.

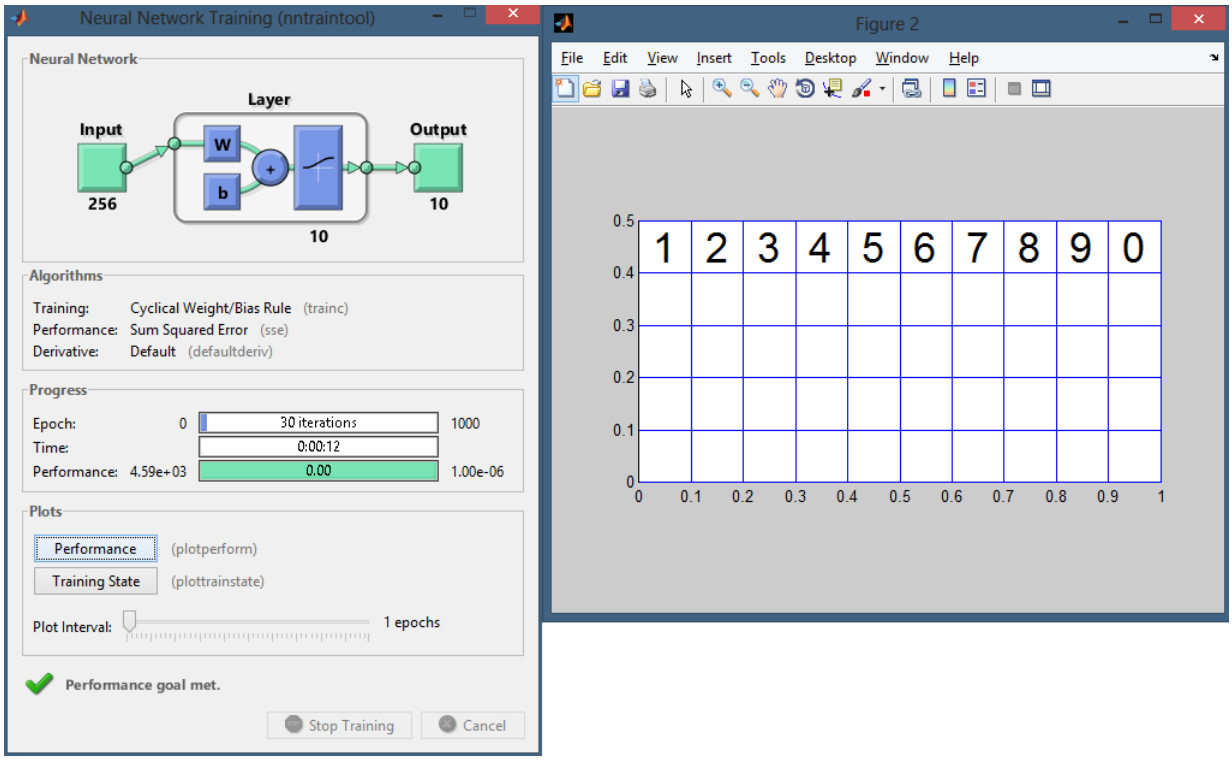

Figure 66: No Associative Memory, Hard-Limit with Perceptron, 500 train input values, perfect Arial numerals.

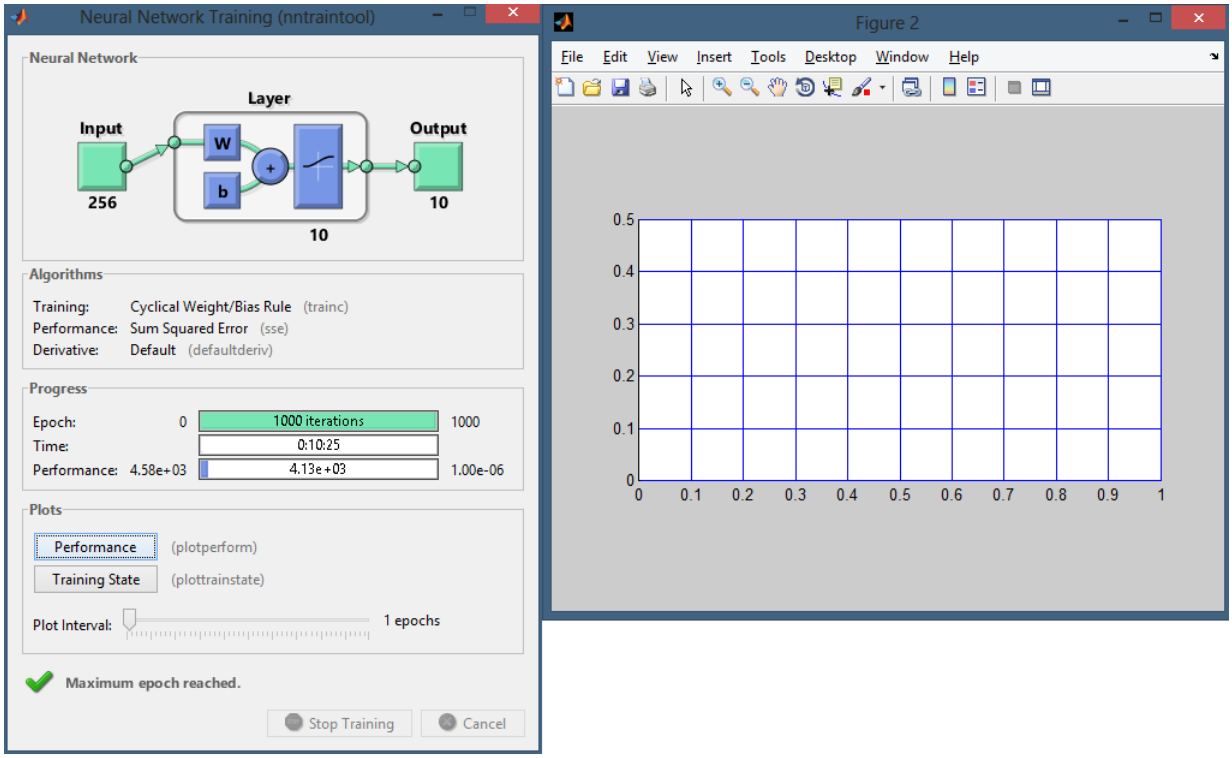

Figure 67: No Associative Memory, Sigmoidal with Gradient descent, 500 train input values, perfect Arial numerals.

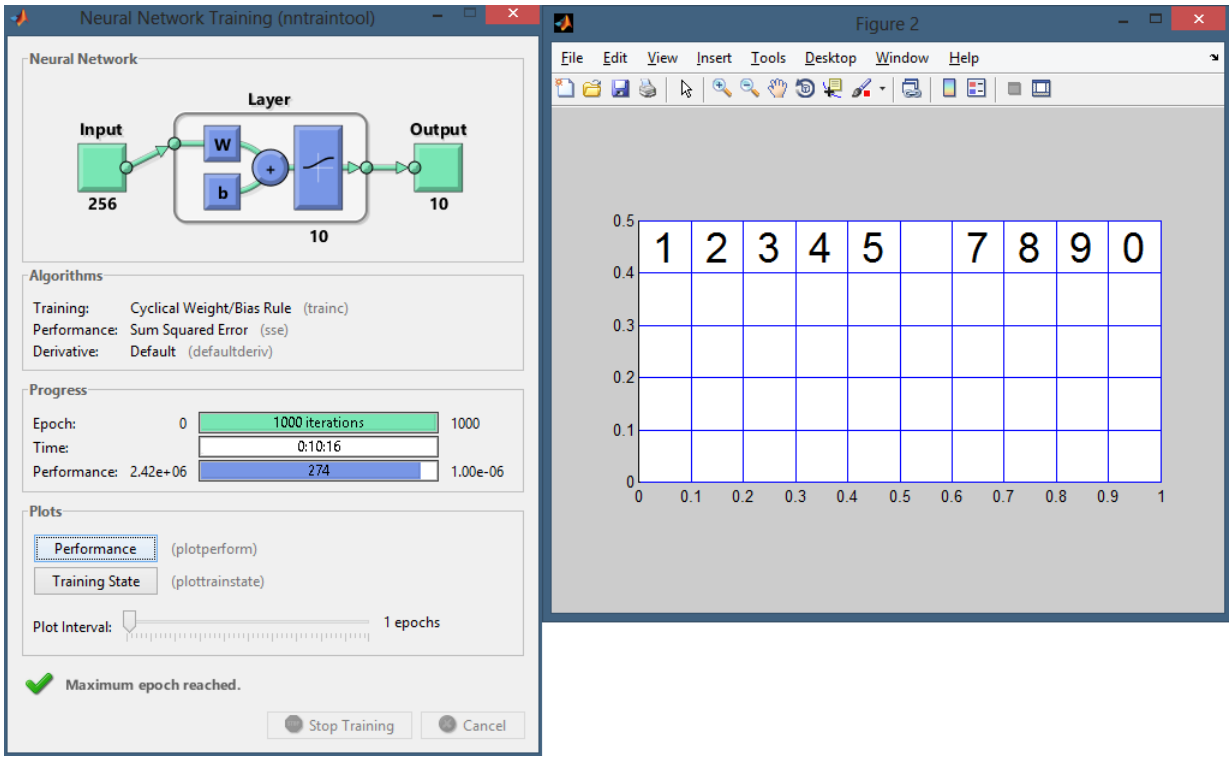

Figure 68: No Associative Memory, Linear with Gradient descent, 500 train input values, perfect Arial numerals.

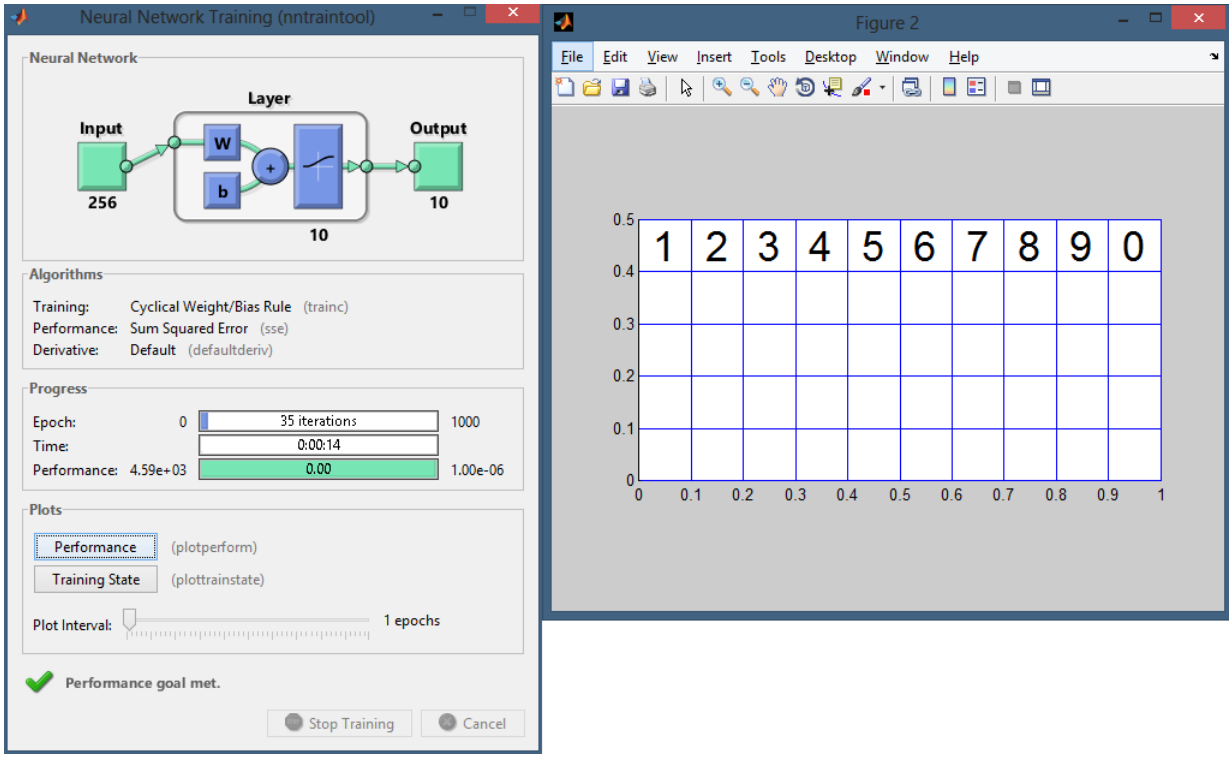

Figure 69: No Associative Memory, Hard-Limit with Perceptron, 500 train input values, perfect Arial numerals.