Project 1 - EEL 5813 - Neural Networks

Nestor Hernandez

May 31, 2017

1 Introduction

In this project, single-layer networks will discriminate 2 classes of 5×4 characters (one class at a time). One prototype of each class will be given to build a training set. Then test sets with increasing levels of noise will be used in testing the trained networks. Hebbian, Perceptron, and Adaline learning will be used for training weights in the single-layer network.

2 Training and Testing Sets.

The patterns used with the networks are 5×4 matrices of bipolar data ($+1$ or -1). Graphically, a value of $+1$ is an ON (dark pixel), while a -1 value is an OFF (blank) pixel. We transform the matrices to vectors by row-scanning left to right, starting at the top. In the networks trained, the bias term is the last element of the input and weight vectors.
The training and testing sets were generated following the guidelines in the project's handout in a spreadsheet-like software. There are 8 files in the .zip file attached: 4 files used to train and test single neural networks for the class "A" and another 4 to do so with class "E". The only difference between the files is the target of the patterns.

3 Single Layer Networks (Class "A")

In this part, we design, train and test a single-layer, single-output network to recognize the "A" class, separating its members from the rest of the patterns.

3.1 Hebbian Learning

Donald Hebb, a psychologist at McGill University, proposed in 1949 that learning takes place by modification of the synapse strengths (weights) in a manner such that if two interconnected neurons are both "on" at the same time, then the strength of the connection between those neurons (weight) should be increased. Neurons that fire together, wire together. In terms of the weight's update, a Hebbian network works as follows:

$$w_i^k = w_i^{k-1} + x_i y$$

where x_i is the $i-th$ element of the training pattern x and y is the target or label of x.

Table 3.1.1 displays the sets of weights obtained by using Hebbian learning after training. Also, Table 3.1.2 shows the results after applying the trained network to the three test sets designed. We see that the hit ratio is 100% for the three sets.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
-17	-13	-5	-13	-11	-11	19	1	21	-5
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}
-7	12	-3	17	5	-1	-15	-19	-23	3
b									
-15									

Table 3.1.1: Set of weights obtain after training with Hebbian learning

Pattern	**Actual**	**Test I**	**Test II**	**Test III**
A1	1	1	1	1
A2	1	1	1	1
A3	1	1	1	1
A4	1	1	1	1
A5	1	1	1	1
E1	-1	-1	-1	-1
E2	-1	-1	-1	-1
E3	-1	-1	-1	-1
E4	-1	-1	-1	-1
E5	-1	-1	-1	-1
I1	-1	-1	-1	-1
I2	-1	-1	-1	-1
I3	-1	-1	-1	-1
I4	-1	-1	-1	-1
I5	-1	-1	-1	-1
O1	-1	-1	-1	-1
O2	-1	-1	-1	-1
O3	-1	-1	-1	-1
O4	-1	-1	-1	-1
O5	-1	-1	-1	-1
U1	-1	-1	-1	-1
U2	-1	-1	-1	-1
U3	-1	-1	-1	-1
U4	-1	-1	-1	-1
U5	-1	-1	-1	-1
Error		0.00%	0.00%	0.00%

Table 3.1.2: Results after applying the trained Hebbian network to the test sets.

3.2 Perceptron Learning

Perceptron is a more evolved form of training wherein the activation for current weights is calculated and the weights are modified only if there is a deviation from the target. Training is modulated by a learning rate $\alpha \in [0, 1]$. The weights are computed as follows:

$$w_i^k = w_i^{k-1} + \alpha y x_i$$

where x_i is the $i-th$ element of the training pattern x and y is the target or label of x.

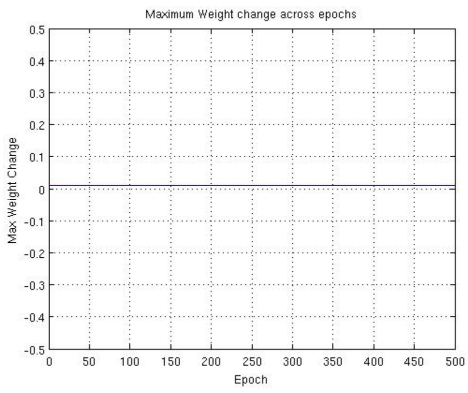
The algorithm stops if the maximum weight change is below a threshold or tolerance ϵ or the maximum number of epochs E is reached. This stopping condition is used in Adaline as well. We repeat the training process for three different combinations of α ϵ.

3.2.1 $\alpha = 0.01$, $\epsilon = 0.001$, $E = 500$

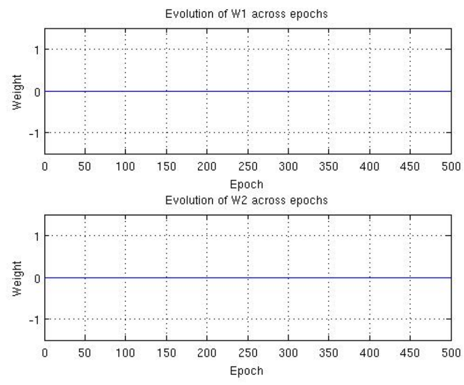
Table 3.2.1 displays the weights obtained by the perceptron network using $\alpha = 0.01$, $\epsilon = 0.001$, and $E = 500$. Figure 3.2.1 shows the maximum weight change across epochs and the evolution of $w_i, i = 1, \ldots, 5$ and the bias term b.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	0.01	0.01	0.03	0.01
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}
-0.01	0.01	-0.01	0.01	0.01	-0.01	-0.01	-0.03	-0.03	-0.01
b									
-0.03									

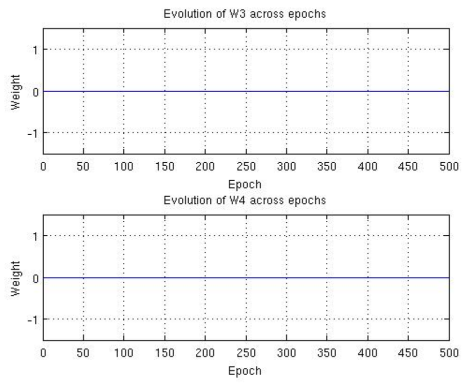
Table 3.2.1: Set of weights obtain after training with Perceptron learning.



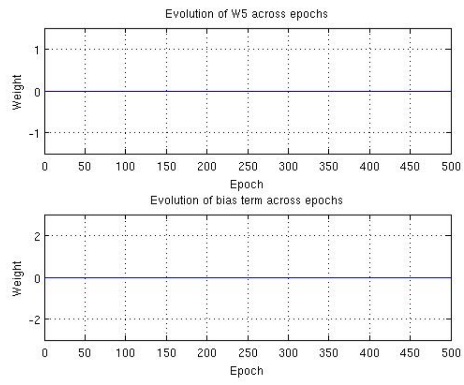
(a) Maximum weight change across epochs



(b) Evolution of w_1 and w_2 across epochs



(c) Evolution of w_3 and w_4 across epochs



(d) Evolution of w_5 and b across epochs

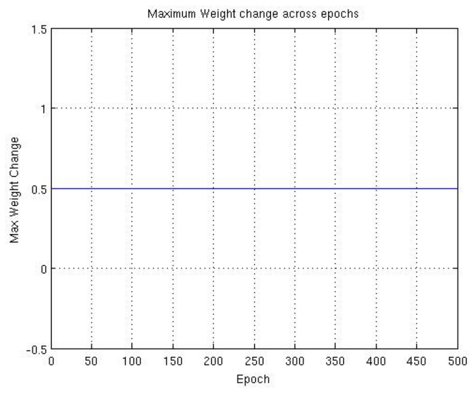
Figure 3.2.1: Results of Perceptron learning with $\alpha = 0.01$, $\epsilon = 0.001$ and $E = 500$

3.2.2 $\alpha = 0.5$, $\epsilon = 0.01$, $E = 500$

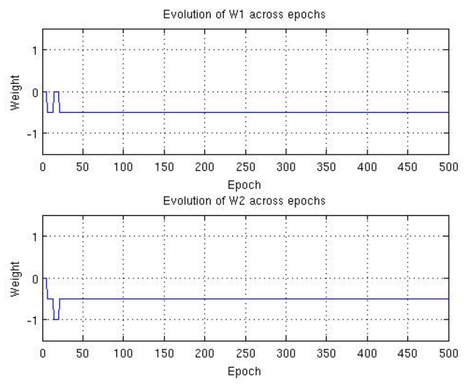
Table 3.2.2 displays the weights obtained by the perceptron network using $\alpha = 0.5$, $\epsilon = 0.01$, and $E = 500$. Figure 3.2.2 shows the maximum weight change across epochs and the evolution of $w_i, i = 1, \ldots, 5$ and the bias term b.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	0.5	0.5	1.5	0.5
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}
-0.5	0.5	-0.5	0.5	0.5	-0.5	-0.5	-1.5	-1.5	-0.5
b									
-1.5									

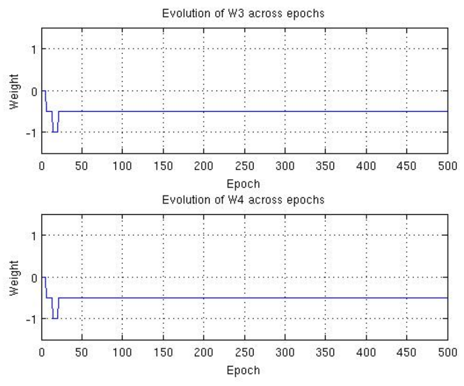
Table 3.2.2: Set of weights obtain after training with Perceptron learning.



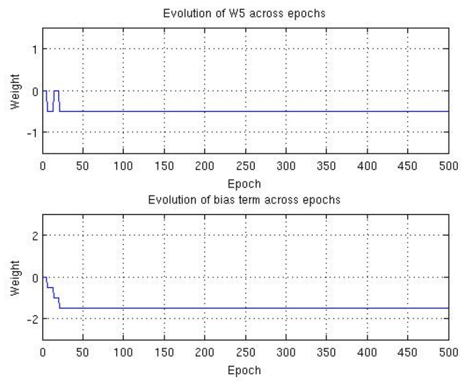
(a) Maximum weight change across epochs



(b) Evolution of w_1 and w_2 across epochs



(c) Evolution of w_3 and w_4 across epochs



(d) Evolution of w_5 and b across epochs

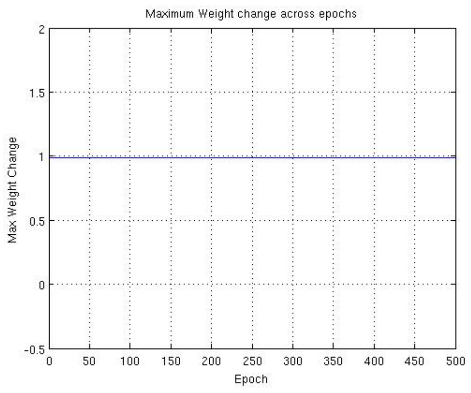
Figure 3.2.2: Results of Perceptron learning with $\alpha = 0.5$, $\epsilon = 0.01$ and $E = 500$

3.2.3 $\alpha = 0.99$, $\epsilon = 0.01$, $E = 500$

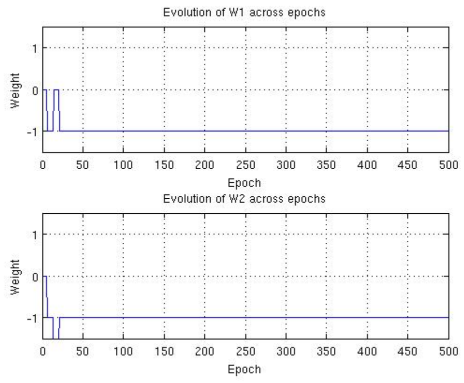
Table 3.2.3 displays the weights obtained by the perceptron network using $\alpha = 0.99$, $\epsilon = 0.01$, and $E = 500$. Figure 3.2.3 shows the maximum weight change across epochs and the evolution of $w_i, i = 1, \ldots, 5$ and the bias term b.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
-0.99	-0.99	-0.99	-0.99	-0.99	-0.99	0.99	0.99	2.97	0.99
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}
-0.99	0.99	-0.99	0.99	0.99	-0.99	-0.99	-2.97	-2.97	-0.99
b									
-2.97									

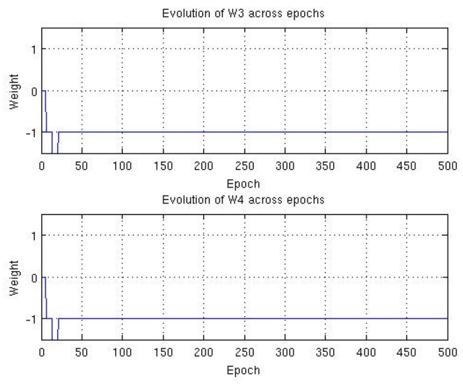
Table 3.2.3: Set of weights obtain after training with Perceptron learning.



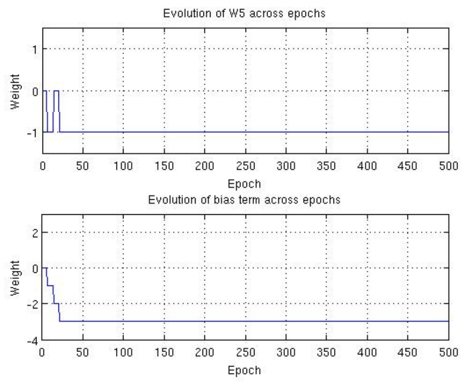
(a) Maximum weight change across epochs



(b) Evolution of w_1 and w_2 across epochs



(c) Evolution of w_3 and w_4 across epochs



(d) Evolution of w_5 and b across epochs

Figure 3.2.3: Results of Perceptron learning with $\alpha = 0.99$, $\epsilon = 0.01$ and $E = 500$

3.2.4 Testing

We have selected the weights obtained with $\alpha = 0.5$, $\epsilon = 0.01$, and $E = 500$ and applied the trained network to every test set. Table 3.2.4 displays the prediction for each pattern: the generalization error is 0% in all three test sets which means that the hit ratio is 100%.

Pattern	Actual	Test I	Test II	Test III
A1	1	1	1	1
A2	1	1	1	1
A3	1	1	1	1
A4	1	1	1	1
A5	1	1	1	1
E1	-1	-1	-1	-1
E2	-1	-1	-1	-1
E3	-1	-1	-1	-1
E4	-1	-1	-1	-1
E5	-1	-1	-1	-1
I1	-1	-1	-1	-1
I2	-1	-1	-1	-1
I3	-1	-1	-1	-1
I4	-1	-1	-1	-1
I5	-1	-1	-1	-1
O1	-1	-1	-1	-1
O2	-1	-1	-1	-1
O3	-1	-1	-1	-1
O4	-1	-1	-1	-1
O5	-1	-1	-1	-1
U1	-1	-1	-1	-1
U2	-1	-1	-1	-1
U3	-1	-1	-1	-1
U4	-1	-1	-1	-1
U5	-1	-1	-1	-1
Error		0.00%	0.00%	0.00%

Table 3.2.4: Results after applying the trained perceptron network to the test sets.

3.3 Adaline

A third training method for the simple one-layer, one-output feedforward network is known as the "Delta Rule" or Adaline. This learning method also modifies the weights according to the error found in each iteration, but it removes the nonlinear activation function for the output element during training.

In this way, the measurement of error is obtained as a continuous variable. Furthermore, the overall measure of performance for each epoch is the sum of squared errors found for all the patterns in that particular epoch.

The weights are computed as follows:

$$w_i^k = w_i^{k-1} + \alpha(t - y_{in})x_i$$

where x_i is the $i - th$ element of the training pattern x and t is the target or label of x and $y_{in} = w^T x$.

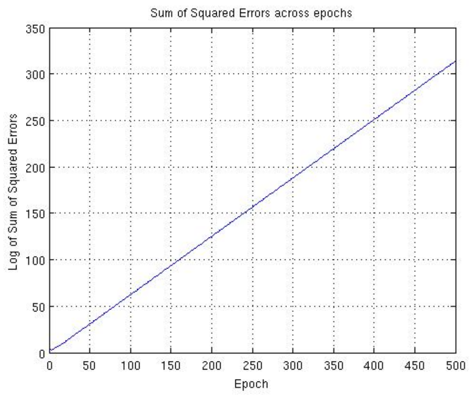
It has been suggested that for a single output neuron, good values for α are in $[\frac{1}{10n}, \frac{1}{n}]$ where n is the number of input neurons.

3.3.1 $\alpha = 0.1$, $\epsilon = 0.001$, $E = 500$

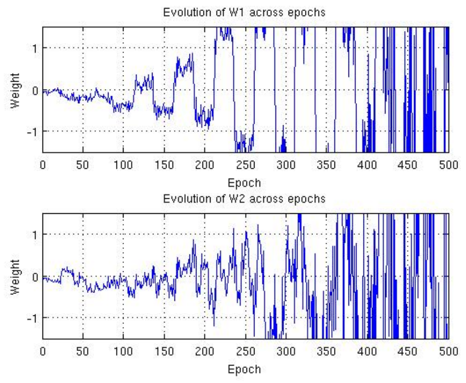
Table 3.3.1 displays the weights obtained by Adaline learning using $\alpha = 0.1$, $\epsilon = 0.001$, and $E = 500$. Note that $\alpha = 0.1$ does not converge in this case as can be seen from Figure 3.3.1 which shows the logarithm of the sum of squared errors across epochs and the evolution of $w_i, i = 1, \ldots, 5$ and the bias term b. Notice also that this value for α is not in the interval suggested $[\frac{1}{10n}, \frac{1}{n}]$.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
-0.1413	-0.2539	0.3512	0.6432	0.4082	0.3788	-0.4145	0.2755	-0.2234	0.3573
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}
-0.429	-0.24	1.2396	-0.5212	-0.1303	0.31	0.2387	0.2955	0.6803	0.3341
b									
0.3481									

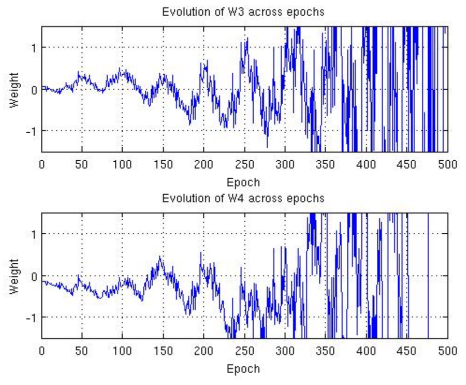
Table 3.3.1: Set of weights obtain after training with Adaline learning. Each w_i has to be multiplied by $1.0e{+}67$.



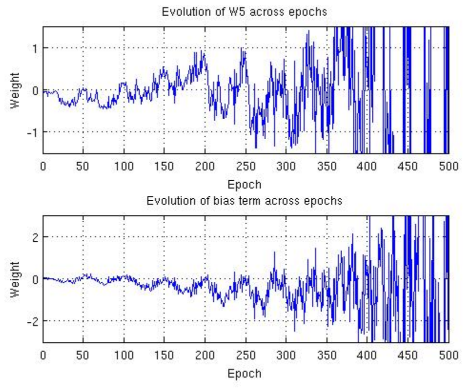
(a) Logarithm of the sum of squared errors across epochs



(b) Evolution of w_1 and w_2 across epochs



(c) Evolution of w_3 and w_4 across epochs



(d) Evolution of w_5 and b across epochs

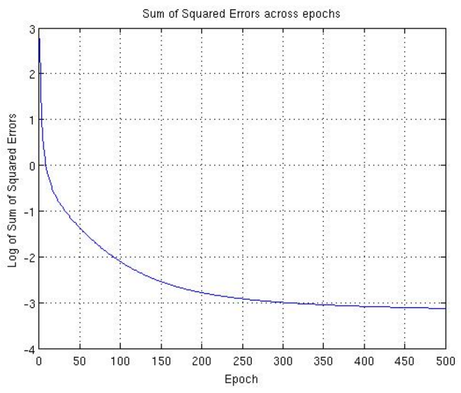
Figure 3.3.1: Results of Adaline learning with $\alpha = 0.1$, $\epsilon = 0.001$ and $E = 500$

3.3.2 $\alpha = 0.004$, $\epsilon = 0.001$, $E = 500$

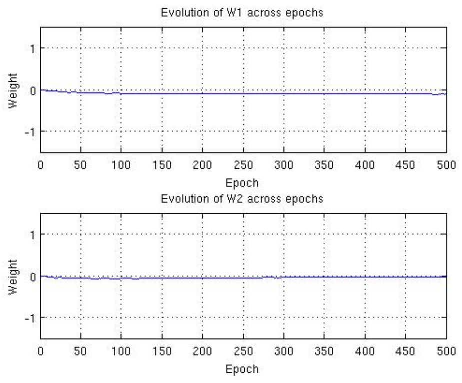
Following the rule of thumb to select α, we try $\alpha = 0.004 \in [\frac{1}{10n}, \frac{1}{n}]$ with $n = 25$. Table 3.3.2 displays the weights obtained by Adaline learning using $\epsilon = 0.001$ and $E = 500$. Figure 3.3.2 shows the logarithm of the sum of squared errors across epochs and the evolution of $w_i, i = 1, \dots, 5$ and the bias term b.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
-0.2839	0.148	-0.0419	-0.1597	-0.1256	-0.0264	0.0428	-0.0762	0.1086	0.1923
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}
-0.0231	0.1521	0.0562	-0.0184	0.0256	0.0647	-0.0043	-0.1067	-0.3353	0.1552
b									
-0.248									

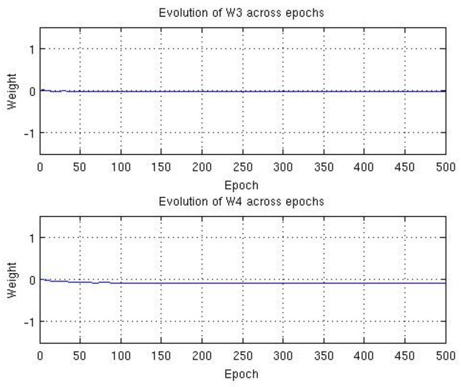
Table 3.3.2: Set of weights obtain after training with Adaline learning.



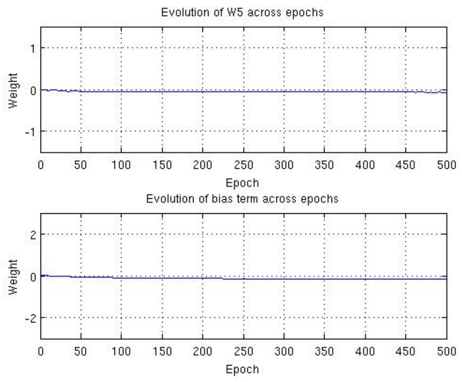
(a) Logarithm of the sum of squared errors across epochs



(b) Evolution of w_1 and w_2 across epochs



(c) Evolution of w_3 and w_4 across epochs



(d) Evolution of w_5 and b across epochs

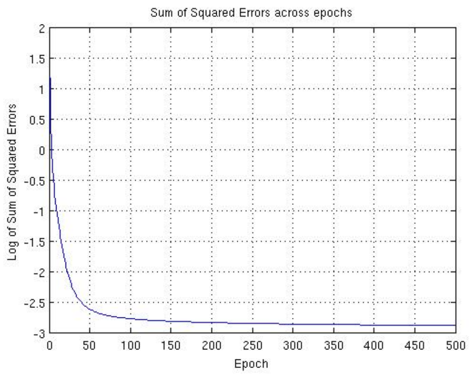
Figure 3.3.2: Results of Adaline learning with $\alpha = 0.004$, $\epsilon = 0.001$ and $E = 500$

3.3.3 $\alpha = 0.02$, $\epsilon = 0.001$, $E = 500$

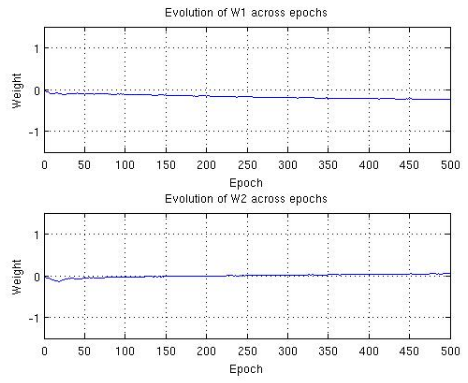
Table 3.3.3 displays the weights obtained by Adaline learning using $\alpha = 0.02$,$\epsilon = 0.001$ and $E = 500$. Figure 3.3.3 shows the logarithm of the sum of squared errors across epochs and the evolution of $w_i, i = 1, \ldots, 5$ and the bias term b.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
-0.2908	0.1925	-0.1213	-0.1214	-0.157	-0.0238	0.0404	-0.1138	0.1438	0.2094
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}
-0.0224	0.1261	0.0737	-0.0118	-0.0107	0.0628	0.0047	-0.1426	-0.3367	0.1861
b									
-0.2124									

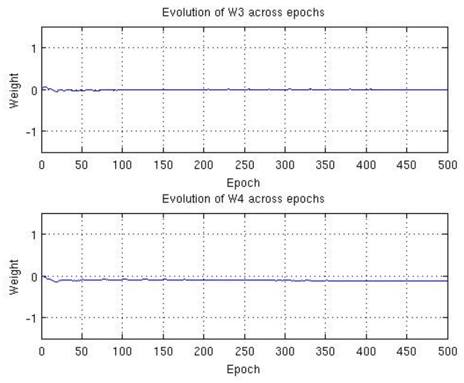
Table 3.3.3: Set of weights obtain after training with Adaline learning.



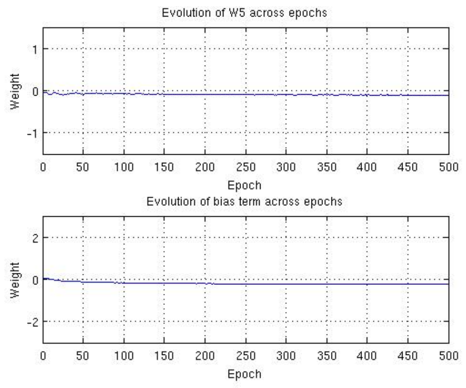
(a) Logarithm of the sum of squared errors across epochs



(b) Evolution of w_1 and w_2 across epochs



(c) Evolution of w_3 and w_4 across epochs



(d) Evolution of w_5 and b across epochs

Figure 3.3.3: Results of Adaline learning with $\alpha = 0.02$, $\epsilon = 0.001$ and $E = 500$

3.3.4 Testing

We have selected the weights obtained with $\alpha = 0.004$, $\epsilon = 0.001$, and $E = 500$ and applied the trained network to every test set. Table 3.3.4 displays the prediction for each pattern: the generalization error is 0% in all three test sets which means that the hit ratio is 100%.

Pattern	**Actual**	**Test I**	**Test II**	**Test III**
A1	1	1	1	1
A2	1	1	1	1
A3	1	1	1	1
A4	1	1	1	1
A5	1	1	1	1
E1	-1	-1	-1	-1
E2	-1	-1	-1	-1
E3	-1	-1	-1	-1
E4	-1	-1	-1	-1
E5	-1	-1	-1	-1
I1	-1	-1	-1	-1
I2	-1	-1	-1	-1
I3	-1	-1	-1	-1
I4	-1	-1	-1	-1
I5	-1	-1	-1	-1
O1	-1	-1	-1	-1
O2	-1	-1	-1	-1
O3	-1	-1	-1	-1
O4	-1	-1	-1	-1
O5	-1	-1	-1	-1
U1	-1	-1	-1	-1
U2	-1	-1	-1	-1
U3	-1	-1	-1	-1
U4	-1	-1	-1	-1
U5	-1	-1	-1	-1
Error		0.00%	0.00%	0.00%

Table 3.3.4: Results after applying the trained Adaline network to the test sets.

4 Single Layer Networks (Class "E")

4.1 Hebbian Learning

Table 4.1.1 displays the sets of weights obtained by using Hebbian learning after training. Also, Table 4.1.2 shows the results after applying the trained network to the three test sets designed. We see that the error is 24%, 20%, and 32% for Test I, II and III respectively.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
3	1	-5	3	5	5	5	5	3	-5
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}
-5	11	7	1	-13	-7	-5	-7	-7	-5
b									
-15									

Table 4.1.1: Set of weights obtain after training with Hebbian learning.

Pattern	**Actual**	**Test I**	**Test II**	**Test III**
A1	-1	-1	-1	-1
A2	-1	-1	-1	-1
A3	-1	1	1	1
A4	-1	-1	-1	-1
A5	-1	-1	-1	1
E1	1	-1	-1	-1
E2	1	-1	1	1
E3	1	-1	-1	-1
E4	1	-1	-1	-1
E5	1	-1	-1	-1
I1	-1	-1	-1	-1
I2	-1	-1	-1	-1
I3	-1	-1	-1	1
I4	-1	-1	-1	1
I5	-1	-1	-1	-1
O1	-1	-1	-1	-1
O2	-1	-1	-1	-1
O3	-1	-1	-1	-1
O4	-1	-1	-1	-1
O5	-1	-1	-1	-1
U1	-1	-1	-1	-1
U2	-1	-1	-1	-1
U3	-1	-1	-1	-1
U4	-1	-1	-1	-1
U5	-1	-1	-1	-1
Error		**24.00%**	**20.00%**	**32.00%**

Table 4.1.2: Results after applying the trained Hebbian network to the test sets.

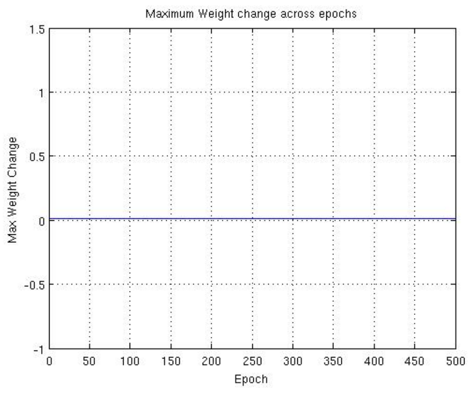
4.2 Perceptron Learning

4.2.1 $\alpha = 0.01$, $\epsilon = 0.001$, $E = 500$

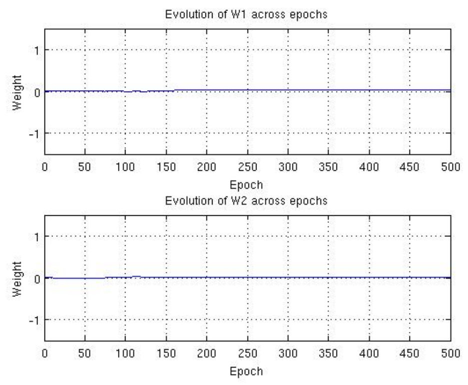
Table 4.2.1 displays the weights obtained by the perceptron network using $\alpha = 0.01$, $\epsilon = 0.001$, and $E = 500$. Figure 4.2.1 shows the maximum weight change across epochs and the evolution of $w_i, i = 1, \ldots, 5$ and the bias term b.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
0.03	0.01	-0.03	0.03	0.07	0.07	0.01	-0.01	0.01	-0.05
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}
0.01	0.03	0.05	0.01	-0.05	0.01	-0.01	-0.01	-0.01	0.01
b									
-0.07									

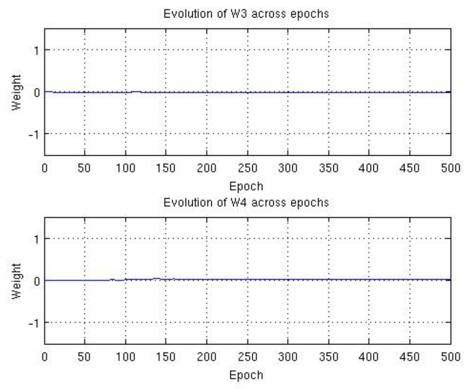
Table 4.2.1: Set of weights obtain after training with Perceptron learning.



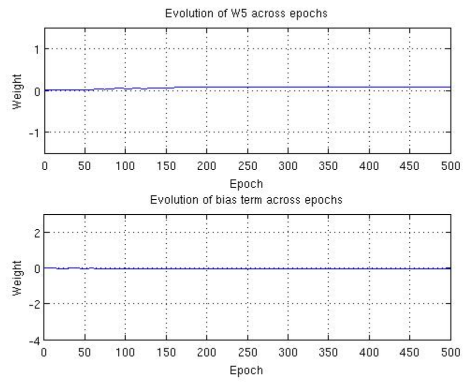
(a) Maximum weight change across epochs



(b) Evolution of w_1 and w_2 across epochs



(c) Evolution of w_3 and w_4 across epochs



(d) Evolution of w_5 and b across epochs

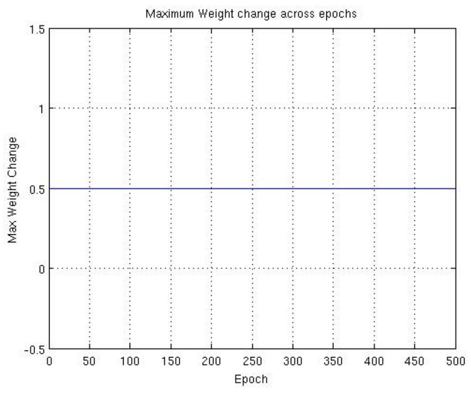
Figure 4.2.1: Results of Perceptron learning with $\alpha = 0.01$, $\epsilon = 0.001$ and $E = 500$

4.2.2 $\alpha = 0.5$, $\epsilon = 0.01$, $E = 500$

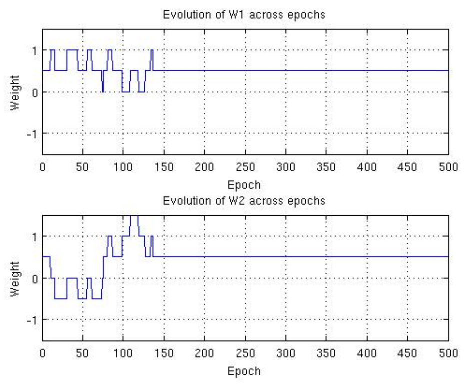
Table 4.2.2 displays the weights obtained by the perceptron network using $\alpha = 0.5$, $\epsilon = 0.01$, and $E = 500$. Figure 4.2.2 shows the maximum weight change across epochs and the evolution of $w_i, i = 1, \ldots, 5$ and the bias term b.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
0.5	0.5	-1.5	1.5	3.5	3.5	0.5	-0.5	0.5	-1.5
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}
0.5	1.5	1.5	0.5	-2.5	0.5	-0.5	-0.5	-0.5	0.5
b									
-3.5									

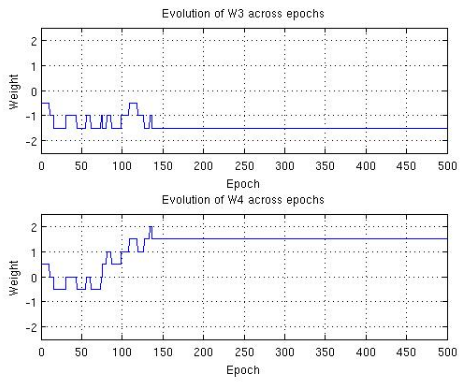
Table 4.2.2: Set of weights obtain after training with Perceptron learning.



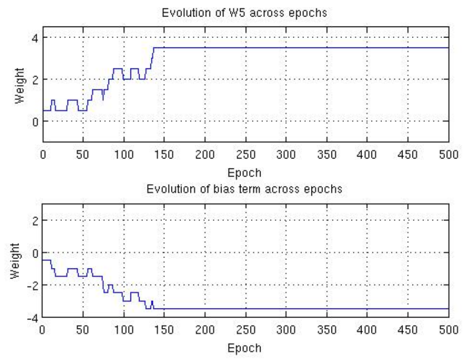
(a) Maximum weight change across epochs



(b) Evolution of w_1 and w_2 across epochs



(c) Evolution of w_3 and w_4 across epochs



(d) Evolution of w_5 and b across epochs

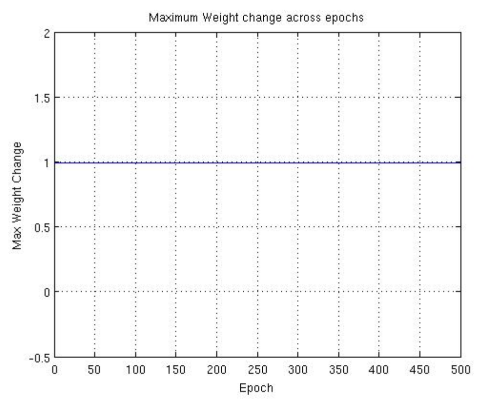
Figure 4.2.2: Results of Perceptron learning with $\alpha = 0.5$, $\epsilon = 0.01$ and $E = 500$

4.2.3 $\alpha = 0.99$, $\epsilon = 0.01$, $E = 500$

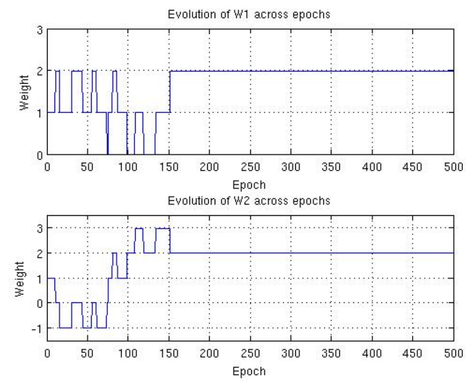
Table 4.2.3 displays the weights obtained by the perceptron network using $\alpha = 0.99$, $\epsilon = 0.01$, and $E = 500$. Figure 4.2.3 shows the maximum weight change across epochs and the evolution of $w_i, i = 1, \ldots, 5$ and the bias term b.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
1.98	1.98	-1.98	3.96	5.94	5.94	0	0	0	-3.96
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}
0	1.98	3.96	0	-5.94	0	0	0	0	0
b									
-5.94									

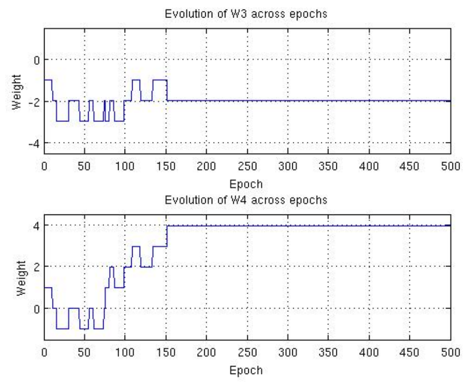
Table 4.2.3: Set of weights obtain after training with Perceptron learning.



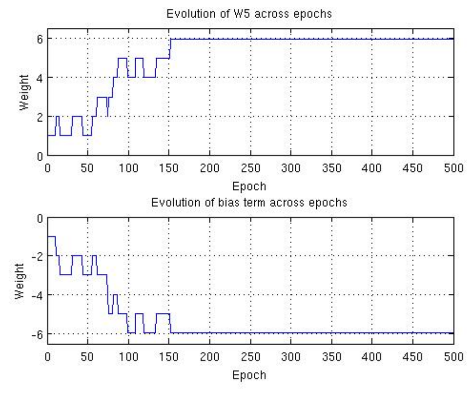
(a) Maximum weight change across epochs



(b) Evolution of w_1 and w_2 across epochs



(c) Evolution of w_3 and w_4 across epochs



(d) Evolution of w_5 and b across epochs

Figure 4.2.3: Results of Perceptron learning with $\alpha = 0.99$, $\epsilon = 0.01$ and $E = 500$

4.2.4 Testing

We have selected the weights obtained with $\alpha = 0.01$, $\epsilon = 0.001$, and $E = 500$ and applied the trained network to every test set. Table 4.2.4 displays the prediction for each pattern: the generalization error is 4% for Test I and 12% for Test II and III.

Pattern	**Actual**	**Test I**	**Test II**	**Test III**
A1	-1	-1	-1	-1
A2	-1	-1	-1	-1
A3	-1	-1	-1	-1
A4	-1	-1	-1	-1
A5	-1	-1	1	1
E1	1	1	1	1
E2	1	1	1	1
E3	1	1	1	1
E4	1	1	1	1
E5	1	1	1	1
I1	-1	-1	-1	-1
I2	-1	-1	-1	-1
I3	-1	-1	-1	-1
I4	-1	-1	-1	-1
I5	-1	-1	-1	-1
O1	-1	-1	-1	-1
O2	-1	-1	-1	-1
O3	-1	-1	-1	-1
O4	-1	-1	-1	-1
O5	-1	-1	1	-1
U1	-1	-1	-1	-1
U2	-1	1	1	1
U3	-1	-1	-1	1
U4	-1	-1	-1	-1
U5	-1	-1	-1	-1
Error		**4.00%**	**12.00%**	**12.00%**

Table 4.2.4: Results after applying the trained perceptron to the test sets.

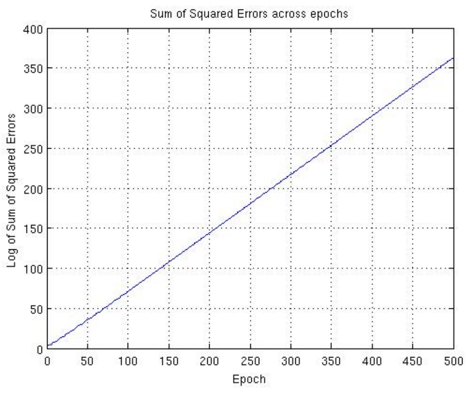
4.3 Adaline

4.3.1 $\alpha = 0.1$, $\epsilon = 0.001$, $E = 500$

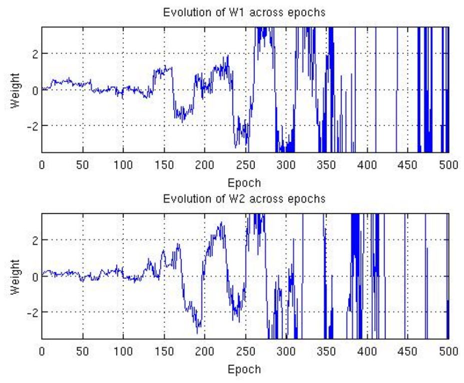
Table 4.3.1 displays the weights obtained by Adaline learning using $\alpha = 0.1$, $\epsilon = 0.001$, and $E = 500$. Note that $\alpha = 0.1$ does not converge in this case as can be seen from Figure 4.3.1 which shows the logarithm of the sum of squared errors across epochs and the evolution of $w_i, i = 1, \ldots, 5$ and the bias term b. Notice also that this value for α is not in the interval suggested $[\frac{1}{10n}, \frac{1}{n}]$.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
0.0405	-3.1359	0.4055	-3.7419	-0.2552	-0.2552	0.2586	-1.986	-0.4839	-1.1731
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}
1.986	4.2487	-1.2797	0.0769	0.194	3.8643	-1.2992	-3.8583	1.3451	1.986
b									
0.942									

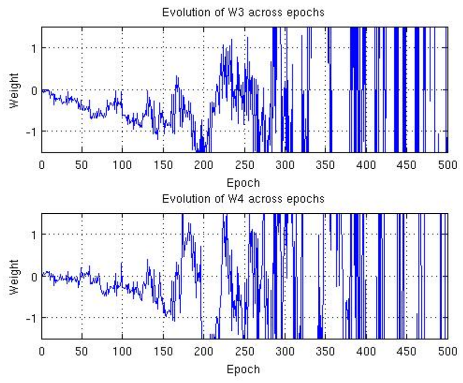
Table 4.3.1: Set of weights obtain after training with Adaline learning. Each w_i has to be multiplied by $1.0e{+}77$.



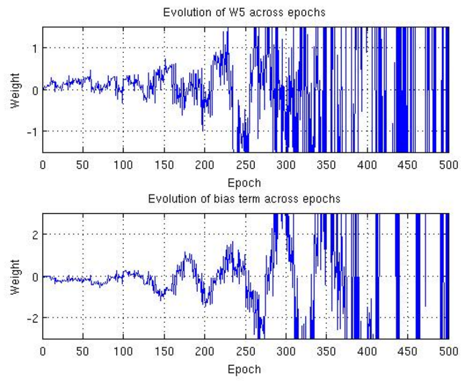
(a) Logarithm of the sum of squared errors across epochs



(b) Evolution of w_1 and w_2 across epochs



(c) Evolution of w_3 and w_4 across epochs



(d) Evolution of w_5 and b across epochs

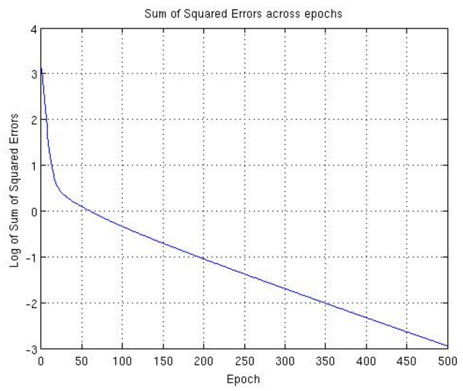
Figure 4.3.1: Results of Adaline learning with $\alpha = 0.1$, $\epsilon = 0.001$ and $E = 500$

4.3.2 $\alpha = 0.004$, $\epsilon = 0.001$, $E = 500$

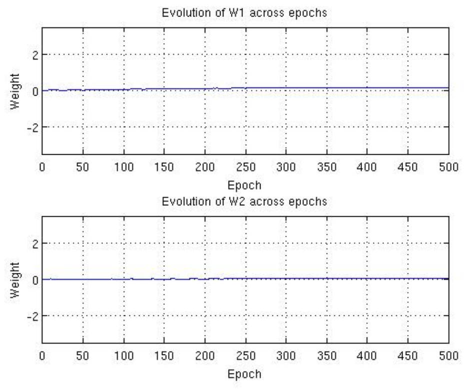
Following the rule of thumb to select α, we try $\alpha = 0.004 \in [\frac{1}{10n}, \frac{1}{n}]$ with $n = 25$. Table 4.3.2 displays the weights obtained by Adaline learning using $\epsilon = 0.001$ and $E = 500$. Figure 4.3.2 shows the logarithm of the sum of squared errors across epochs and the evolution of $w_i, i = 1, \ldots, 5$ and the bias term b.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
0.0164	-0.0065	-0.0301	0.0388	0.5515	0.5515	0.0068	0.0515	0.006	-0.8267
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}
-0.0515	0.035	0.0906	0.0073	-0.0551	0.0014	-0.1199	-0.058	-0.0589	-0.0515
b									
-0.7228									

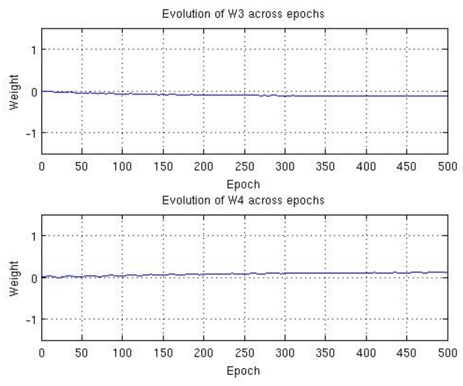
Table 4.3.2: Set of weights obtain after training with Adaline learning.



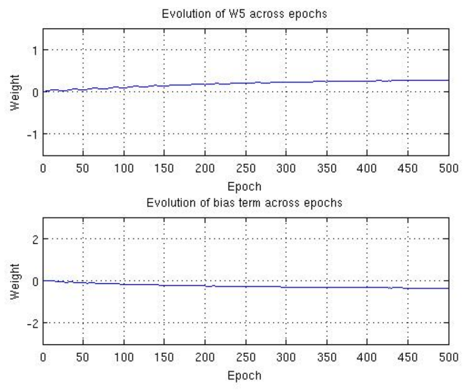
(a) Logarithm of the sum of squared errors across epochs



(b) Evolution of w_1 and w_2 across epochs



(c) Evolution of w_3 and w_4 across epochs



(d) Evolution of w_5 and b across epochs

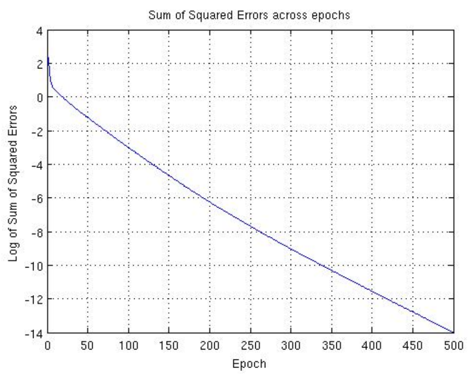
Figure 4.3.2: Results of Adaline learning with $\alpha = 0.004$, $\epsilon = 0.001$ and $E = 500$

4.3.3 $\alpha = 0.02$, $\epsilon = 0.001$, $E = 500$

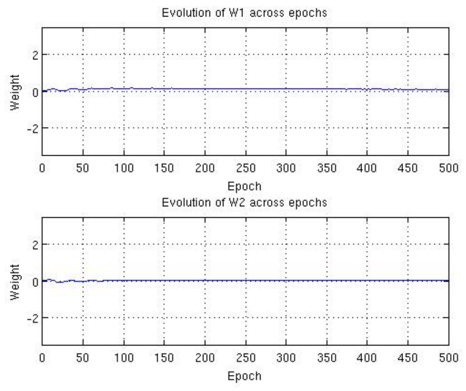
Table 4.3.3 displays the weights obtained by Adaline learning using $\alpha = 0.02$,$\epsilon = 0.001$ and $E = 500$. Figure 4.3.3 shows the logarithm of the sum of squared errors across epochs and the evolution of $w_i, i = 1, \dots, 5$ and the bias term b.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
0	-0.0001	0.0002	-0.0001	0.5884	0.5884	0.0001	0.059	0.0001	-0.9998
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{19}	w_{20}
-0.059	0	0.0002	0	0.0001	0	-0.1759	-0.0003	-0.0004	-0.059
b									
-0.8232									

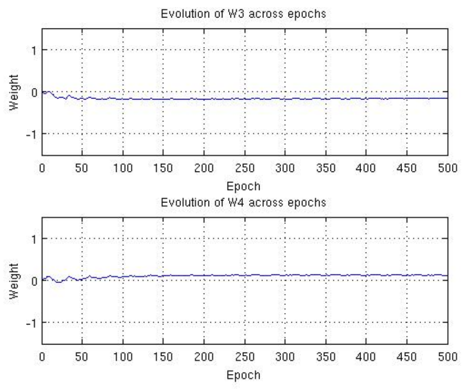
Table 4.3.3: Set of weights obtain after training with Adaline learning.



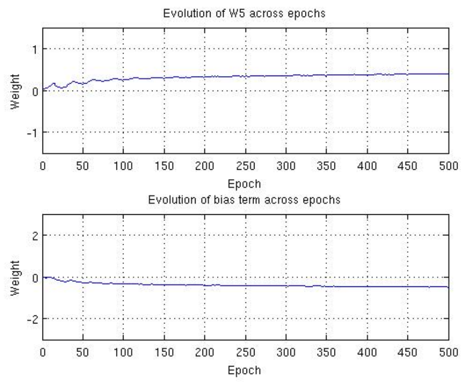
(a) Logarithm of the sum of squared errors across epochs



(b) Evolution of w_1 and w_2 across epochs



(c) Evolution of w_3 and w_4 across epochs



(d) Evolution of w_5 and b across epochs

Figure 4.3.3: Results of Adaline learning with $\alpha = 0.004$, $\epsilon = 0.001$ and $E = 500$

4.3.4 Testing

We have selected the weights obtained with $\alpha = 0.004$, $\epsilon = 0.001$, and $E = 500$ and applied the trained network to every test set. Table 4.3.4 displays the prediction for each pattern: the generalization error is 8%, 12% and 24% for Test I, II and III respectively. We can see that performance decrease when the data is noisy.

Pattern	Actual	Test I	Test II	Test III
A1	-1	-1	-1	-1
A2	-1	-1	-1	-1
A3	-1	-1	-1	-1
A4	-1	-1	-1	-1
A5	-1	1	1	1
E1	1	1	1	1
E2	1	1	1	1
E3	1	-1	-1	-1
E4	1	1	1	-1
E5	1	1	-1	-1
I1	-1	-1	-1	-1
I2	-1	-1	-1	-1
I3	-1	-1	-1	1
I4	-1	-1	-1	-1
I5	-1	-1	-1	-1
O1	-1	-1	-1	-1
O2	-1	-1	-1	-1
O3	-1	-1	-1	-1
O4	-1	-1	-1	-1
O5	-1	-1	-1	-1
U1	-1	-1	-1	-1
U2	-1	-1	-1	-1
U3	-1	-1	-1	1
U4	-1	-1	-1	-1
U5	-1	-1	-1	-1
Error		**8.00%**	**12.00%**	**24.00%**

Table 4.3.4: Classification of each instance using the trained Adaline network.

Conclusions

The three learning algorithms achieved perfect performance to classify character “A” regardless of the noise injected to the test sets. However, the learning algorithms did not perform as well when train to classify the class “E”. Table 4.3.5 shows that Adaline and Perceptron outdid Hebbian learning. Also, Perceptron performed better than Adaline in Test III and I.

Although Hebbian is easy to implement, it seems to be more sensible to noise. On the other hand, we also saw that the choice of the learning step α may be more critical to Adaline than to Perceptron. On the whole, Perceptron and Adaline show better performance and they can continue learning as long as the tolerance ϵ is not reached which is not possible with Hebbian.

Algorithm	Test I	Test II	Test III
Hebbian	24.00%	20.00%	32.00%
Perceptron	4.00%	12.00%	12.00%
Adaline	8.00%	12.00%	24.00%

Table 4.3.5: Performance of the algorithms for class “E” .

As for the differences between training for classes “A” and “E”, we posit that the algorithms learned better to classify “A” because of its distinctive features. Class “E” cannot even be easily recognize by humans as it is given in the encoding proposed for the project. It shares similar features with other letters such as “I”, “O”, and “U”.

5 Matlab Code

```
% activation function
% takes the net input and outputs 1 or -1.
function [y]=activation(y_net)
    if y_net>=0
        y=1;
    else
        y=-1;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Hebbian Learning.
% This script implements Hebbian learning.
% Training_E.csv, Test_I_E.csv, Test_II_E.csv, and Test_III_E.csv are the
% csv files containing the training and test sets respectively. The same
% applies for A.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear
Train_set=csvread('Training_E.csv');
[m,n]=size(Train_set);

w=zeros(21,1);
for i=1:n
    w=w+Train_set(1:21,i)*Train_set(22,i);
end

%Predictions

test_1=csvread('Test_I_E.csv');
T=test_1(1:21,:)'; %matrix containing test instances
pred=arrayfun(@activation,(T*w)); % multiply T by w to get
%the predictions for each instance.

test_2=csvread('Test_II_E.csv');
T=test_2(1:21,:)';
pred=arrayfun(@activation,(T*w));

test_3=csvread('Test_III_E.csv');
T=test_3(1:21,:)';
pred=arrayfun(@activation,(T*w));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This script implements the perceptron learning algorithm.
% The user should set alpha, tol (tolerance) and maxepoch.
% All the plots required in the project are programmed.
% Training_E.csv, Test_I_E.csv, Test_II_E.csv, and Test_III_E.csv are the
% csv files containing the training and test sets respectively. The same
% applies for A.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear
Train_set=csvread('Training_E.csv');
[m,n]=size(Train_set);
alpha=0.01;
w=zeros(21,1);
mx_wt=0;
```

```
tol=0.001;
maxepoch=500;
wc_v=[];
W=[];
miss=0;
for j=1:maxepoch

for i=1:n
     y_net=w'*Train_set(1:21,i);
     y=activation(y_net);

     if y~=Train_set(22,i)
          miss=miss+1;
          deltaw=alpha*Train_set(22,i)*Train_set(1:21,i);
          w=w+deltaw;
          mx_wp=max(abs(deltaw));

          if mx_wp>mx_wt
               mx_wt=mx_wp;
          end
     end
     ind=[1:5,21];
     W=[W,w(ind)];
end

wc_v=[wc_v,mx_wt];

if mx_wt<tol
     break
end

end


%plot to show the maximum weight change (absolute value) in each epoch.
plot(wc_v);
grid on
title('Maximum Weight change across epochs')
xlabel('Epoch')
ylabel('Max Weight Change')


%plots to show the evolution of weights.
figure(1);
subplot(2,1,1);
plot(W(1,:));
grid on
axis([0 500 0 3])
title('Evolution of W1 across epochs')
xlabel('Epoch')
ylabel('Weight')

subplot(2,1,2);
plot(W(2,:));
grid on
axis([0 500 -1.5 3.5])
title('Evolution of W2 across epochs')
```

```
 xlabel('Epoch')
 ylabel('Weight')

 figure(2);
 subplot(2,1,1);
 plot(W(3,:));
 grid on
 axis([0 500 -4.5 1.5])
 title('Evolution of W3 across epochs')
 xlabel('Epoch')
 ylabel('Weight')

 subplot(2,1,2);
 plot(W(4,:));
 grid on
 axis([0 500 -1.5 4.5])
 title('Evolution of W4 across epochs')
 xlabel('Epoch')
 ylabel('Weight')

 figure(3);
 subplot(2,1,1);
 plot(W(5,:));
 grid on
 axis([0 500 0 6.5])
 title('Evolution of W5 across epochs')
 xlabel('Epoch')
 ylabel('Weight')

 subplot(2,1,2);
 plot(W(6,:));
 grid on
 axis([0 500 -6.5 0])
 title('Evolution of bias term across epochs')
 xlabel('Epoch')
 ylabel('Weight')

%Predict using the tained network.

 test_1=csvread('Test_I_E.csv');
T=test_1(1:21,:)'; %matrix containing test instances
 pred=arrayfun(@activation,(T*w)); % multiply T by w
%to get the predictions for each instance.

 test_2=csvread('Test_II_E.csv');
T=test_2(1:21,:)'; %matrix containing test instances
 pred=arrayfun(@activation,(T*w)); % multiply T by w
%to get the predictions for each instance.

 test_3=csvread('Test_III_E.csv');
T=test_3(1:21,:)'; %matrix containing test instances
 pred=arrayfun(@activation,(T*w)); % multiply T by w
%to get the predictions for each instance.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This script implements the Delta Rule.
% The user should set alpha, tol (tolerance) and maxepoch.
% All the plots required in the project are programmed.
% Training_E.csv, Test_I_E.csv, Test_II_E.csv, and Test_III_E.csv are the
% csv files containing the training and test sets respectively. The same
% applies for A.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Adaline (Delta Rule) Learning
clear
Train_set=csvread('Training_E.csv');
[m,n]=size(Train_set);
alpha=0.004;
%w=0.3*ones(21,1);
w=zeros(21,1);
mx_wt=0;
tol=0.001;
maxepoch=500;
ss_v=[];
W=[];
miss=0;
for j=1:maxepoch
    ss=0;
    for i=1:n
        y_net=w'*Train_set(1:21,i);

        deltaw=alpha*(Train_set(22,i)-y_net)*Train_set(1:21,i);
        w=w+deltaw;
        mx_wp=max(abs(deltaw));

        if mx_wp>mx_wt
            mx_wt=mx_wp;
        end
        ss=ss+(y_net-Train_set(22,i))^2;

        ind=[1:5,21];
        W=[W,w(ind)];

    end

    ss_v=[ss_v,ss];

    if mx_wt<tol
        break
    end
end


plot(log(ss_v))
grid on
title('Sum of Squared Errors across epochs')
xlabel('Epoch')
ylabel('Log of Sum of Squared Errors')


%plots to show the evolution of weights.
figure(1);
subplot(2,1,1);
plot(W(1,:));
```

```
 grid on
 axis([0 500 -3.5 3.5])
 title('Evolution of W1 across epochs')
 xlabel('Epoch')
 ylabel('Weight')

 subplot(2,1,2);
 plot(W(2,:));
 grid on
 axis([0 500 -3.5 3.5])
 title('Evolution of W2 across epochs')
 xlabel('Epoch')
 ylabel('Weight')

 figure(2);
 subplot(2,1,1);
 plot(W(3,:));
 grid on
 axis([0 500 -1.5 1.5])
 title('Evolution of W3 across epochs')
 xlabel('Epoch')
 ylabel('Weight')

 subplot(2,1,2);
 plot(W(4,:));
 grid on
 axis([0 500 -1.5 1.5])
 title('Evolution of W4 across epochs')
 xlabel('Epoch')
 ylabel('Weight')

 figure(3);
 subplot(2,1,1);
 plot(W(5,:));
 grid on
 axis([0 500 -1.5 1.5])
 title('Evolution of W5 across epochs')
 xlabel('Epoch')
 ylabel('Weight')

 subplot(2,1,2);
 plot(W(6,:));
 grid on
 axis([0 500 -3 3])
 title('Evolution of bias term across epochs')
 xlabel('Epoch')
 ylabel('Weight')

%Predict using the trained weights on the test sets.

 test_1=csvread('Test_I_E.csv');
T=test_1(1:21,:)'; %matrix containing test instances
 pred=arrayfun(@activation,(T*w)); % multiply T by w
%to get the predictions for each instance.

 test_2=csvread('Test_II_E.csv');
T=test_2(1:21,:)'; %matrix containing test instances
```

```
 pred=arrayfun(@activation,(T*w)); % multiply T by w to
%get the predictions for each instance.


 test_3=csvread('Test_III_E.csv');
T=test_3(1:21,:)'; %matrix containing test instances
 pred=arrayfun(@activation,(T*w)); % multiply T by w
%to get the predictions for each instance.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```