Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas

upiita-ipn

Neurofuzzy Systems

Practice 5 NN
Adaline vs Perceptron

Elizarraras Llanos Angel Gustavo

2MM12

Professor: Ph.D. Tovar Corona Blanca

Deliver date: 06/05/2020

Semester 20/2

Index

Introduction

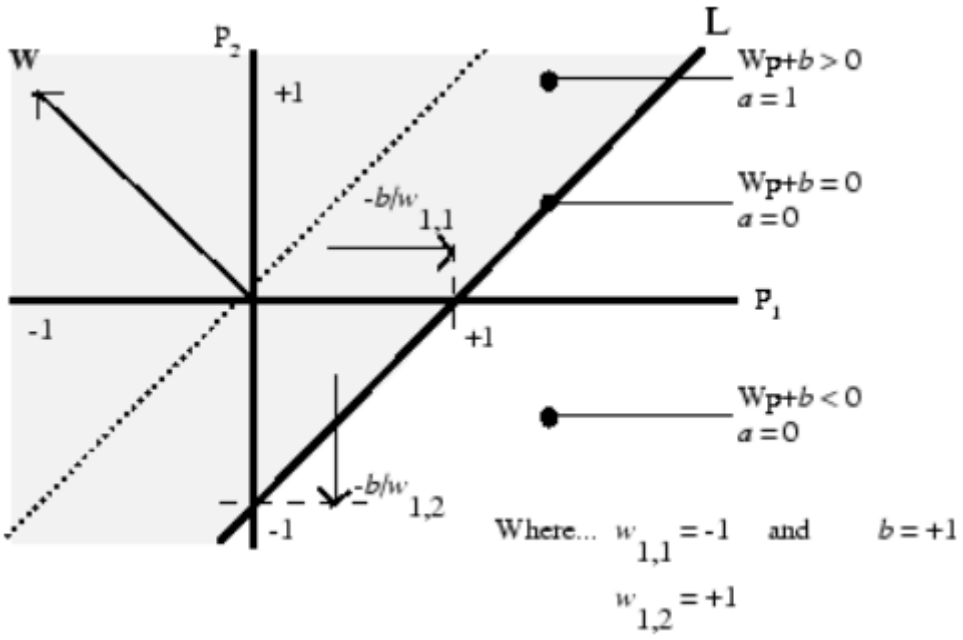
Perceptron

The perceptron generated great interest due to its ability to generalize from its training vectors and learn from initially randomly distributed connections. Perceptron's are especially suited for simple problems in pattern classification. They are fast and reliable networks for the problems they can solve.

The way a neural network learns how to classify patterns is a mathematical operation which depends of an activation function (hard-limit in this practice):

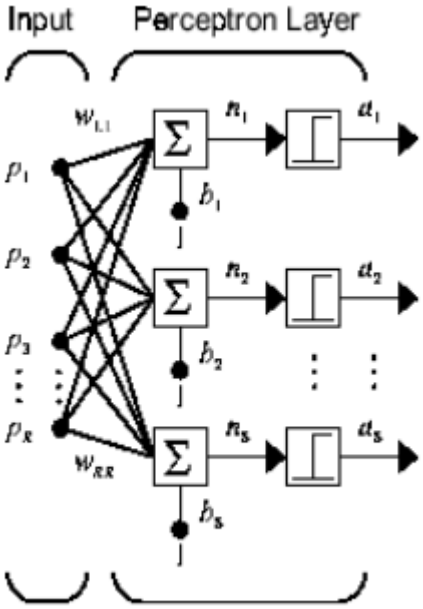
$$a = hardlim(Wp + b)$$

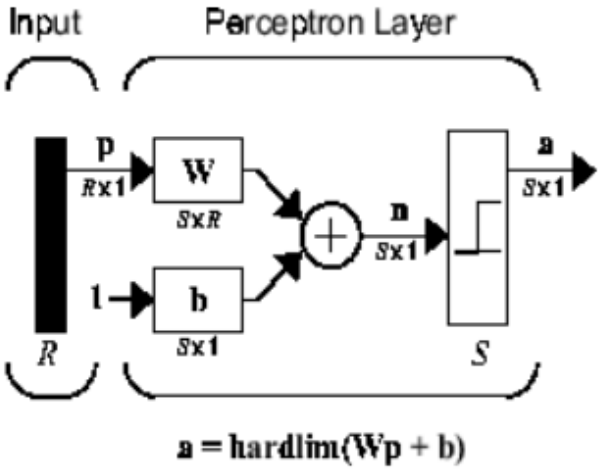
The hard-limit transfer function gives a perceptron the ability to classify input vectors by dividing the input space into two regions.



Each external input is weighted with an appropriate weight $w1j$, and the sum of the weighted inputs is sent to the hard-limit transfer function, which also has an input of 1 transmitted to it through the bias.

The perceptron neuron produces a 1 if the net input into the transfer function is equal to or greater than 0; otherwise it produces a 0. The architecture is shown in the images below.





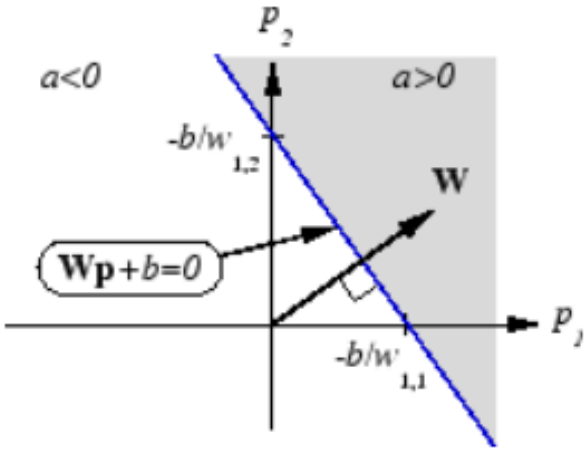
Where

R = number of elements in input

S = number of neurons in layer

Adaline

Like the perceptron, the ADALINE has a *decision boundary* that is determined by the input vectors for which the net input *n* is zero. For *n* = 0 the equation ***Wp + b = 0*** specifies such a decision boundary, as the image shown below:



Input vectors in the upper right gray area lead to an output greater than 0. Input vectors in the lower left white area lead to an output less than 0. Thus, the ADALINE can be used to classify objects into two categories. However, Adaline can classify objects in this way only when the objects are linearly separable.

Difference

The main difference between the two, is that a *Perceptron* takes that binary response (like a classification result) and computes an error used to update the weights, whereas an *Adaline* uses a continuous response value to update the weights (so before the binarized output is produced).

The fact that the Adaline does this, allows its updates to be more representative of the actual error, before it is thresholder, which in turn allows a model to converge more quickly.

Objective

To compare the behavior of Adaline and Perceptron

Problem explaining

We need to differentiate between rabbits and bears toys in a factory with four different neural networks with one Perceptron and three Adaline approaches. In addition, we will observe the error difference between all of them by changing the alpha values in Adaline training and compare they behavior.

Development

We propose 2 classes, one for each animal, then, to do it, the proposal was to difference the toys by propose the weight of the toy and the length of the ears, first and second element of P respectively, also, for the targets, rabbits are 0 for Perceptron and -1 for Adaline, and bears are 1 in both cases.

$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\4\end{bmatrix}, t_1 = 0\right\}\left\{\mathbf{p}_2 = \begin{bmatrix}1\\5\end{bmatrix}, t_2 = 0\right\}\left\{\mathbf{p}_3 = \begin{bmatrix}2\\4\end{bmatrix}, t_3 = 0\right\}\left\{\mathbf{p}_4 = \begin{bmatrix}2\\5\end{bmatrix}, t_4 = 0\right\}$$

$$\left\{\mathbf{p}_5 = \begin{bmatrix}3\\1\end{bmatrix}, t_5 = 1\right\}\left\{\mathbf{p}_6 = \begin{bmatrix}3\\2\end{bmatrix}, t_6 = 1\right\}\left\{\mathbf{p}_7 = \begin{bmatrix}4\\1\end{bmatrix}, t_7 = 1\right\}\left\{\mathbf{p}_8 = \begin{bmatrix}4\\2\end{bmatrix}, t_8 = 1\right\}$$

To observe how the neural networks error, behave, in the Adaline calculation of alpha the next calculations were made:

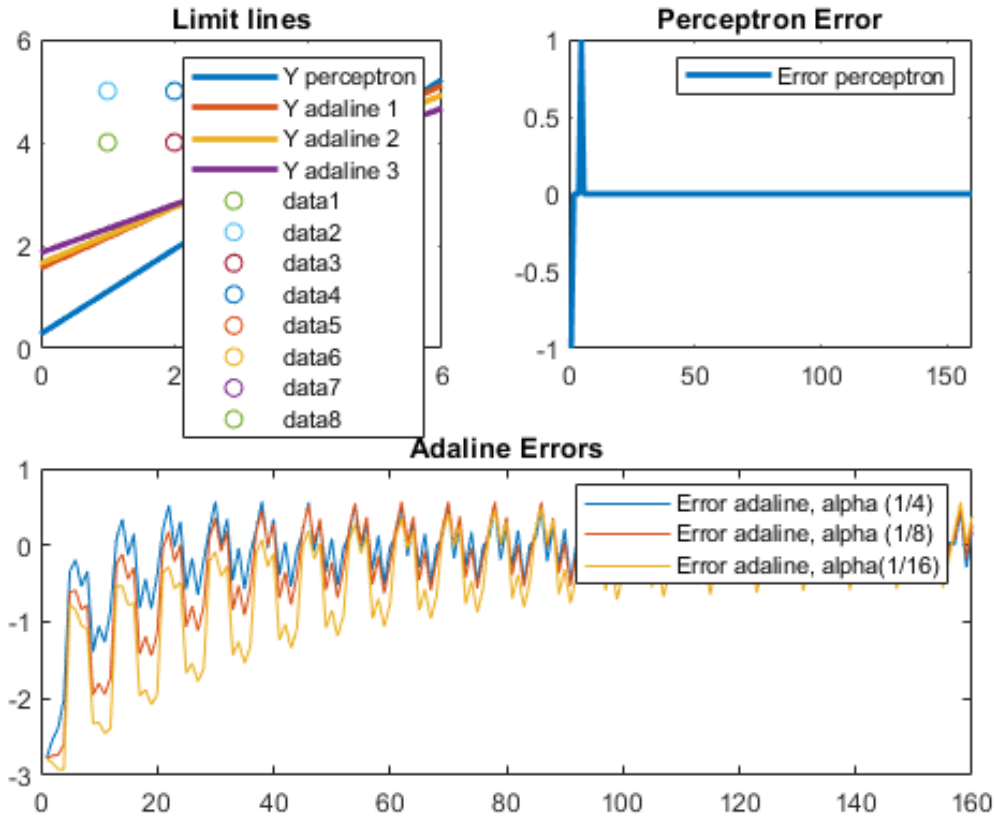
$$\alpha_1 = \frac{1}{4 * \lambda_{max}}, \quad \alpha_2 = \frac{1}{8 * \lambda_{max}}, \quad \alpha_3 = \frac{1}{16 * \lambda_{max}}$$

Results

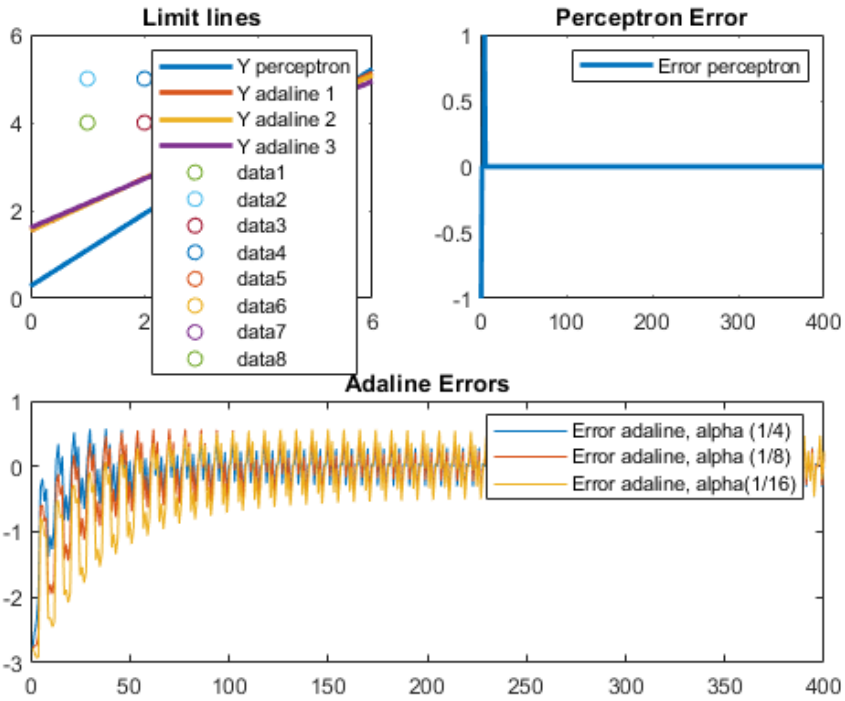
In all the graphs the bear “points” are hidden by the legend of the lines, but they are correctly separated.

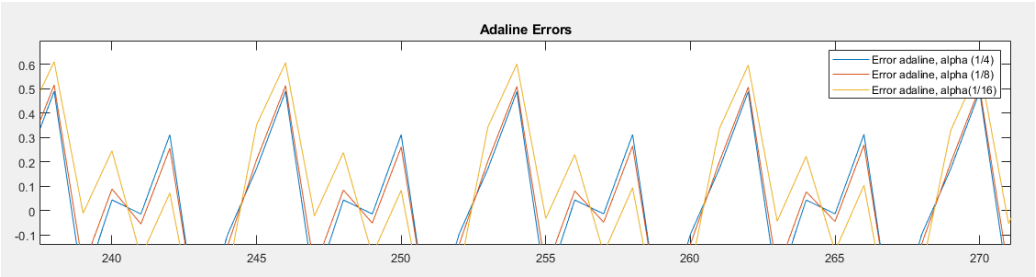
20 epochs

We can observe that both neural networks, Adaline and Perceptron, do their job, Adaline separate the toys almost in the middle. Regarding the errors, perceptron one, only after a few epochs its error is zero. In the other hand, in Adaline we can see that the error oscillates near the zero; the best Adaline is when $\alpha = 1 / (8*\lambda)$, because the error graph oscillates in smaller errors, but there isn’t too much difference when $\alpha = 1 / (4*\lambda)$

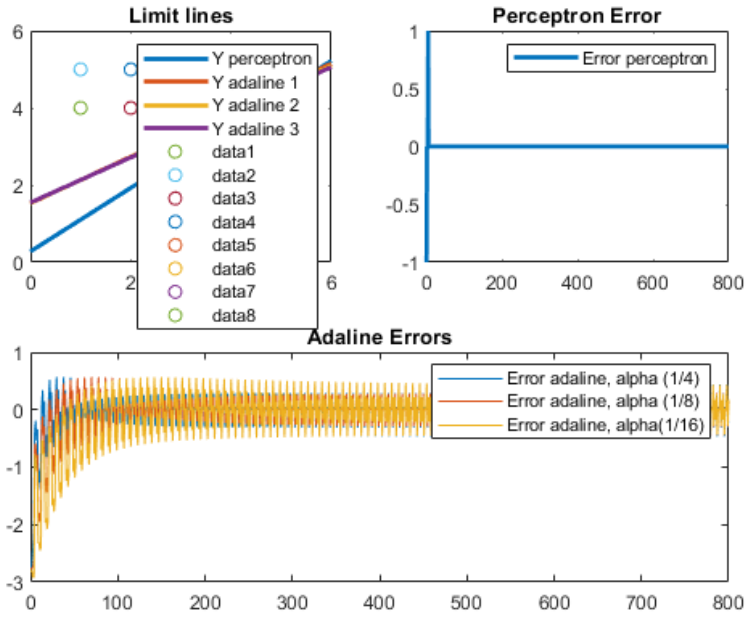


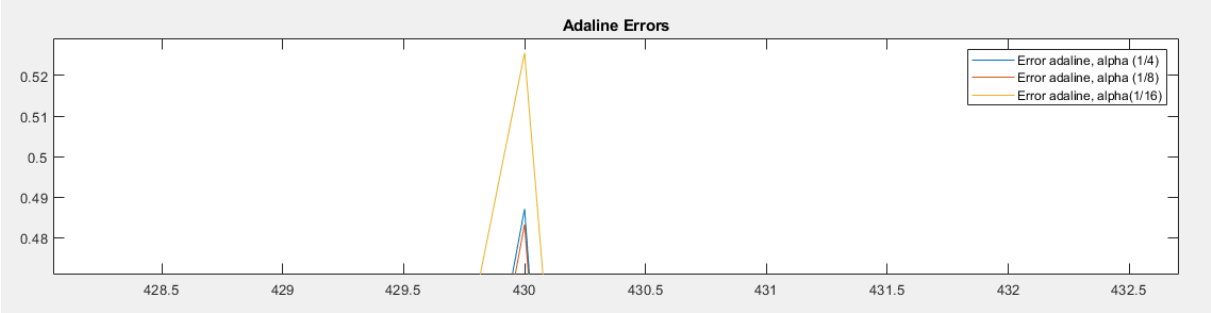
50 epochs





100 epochs



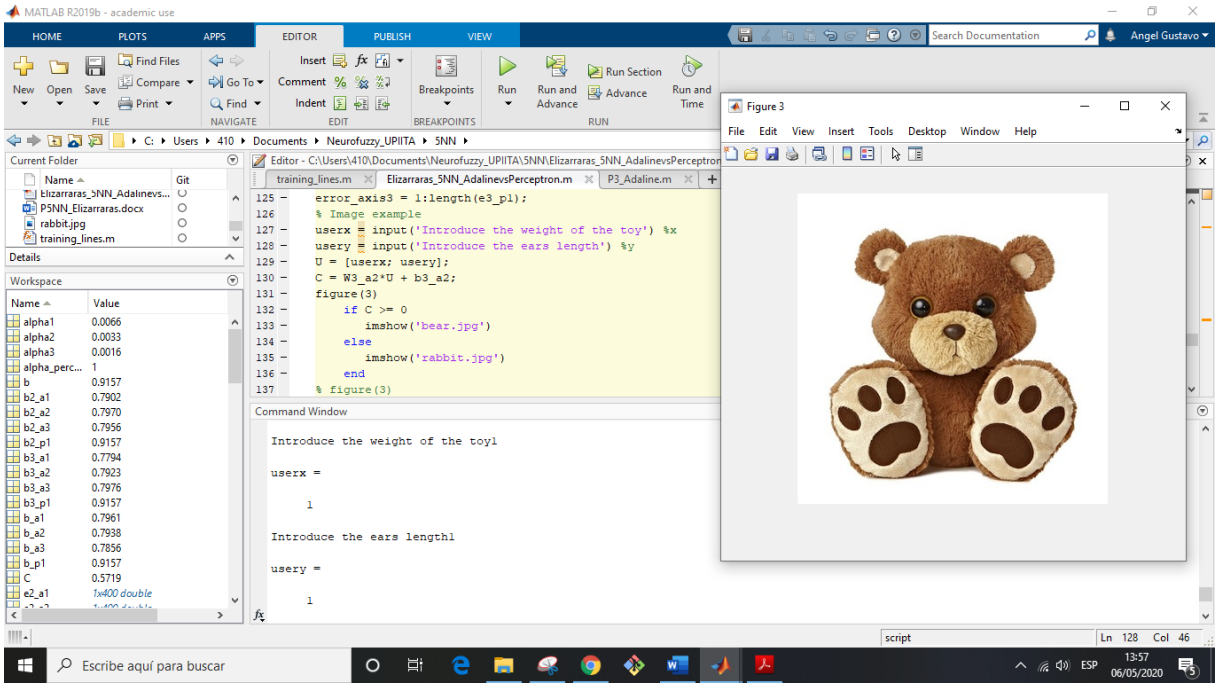


The results of these last 4 graphs are very similar, and we can say that the best sorting network is Adaline, when alpha is α = 1 / (4*λ) or α = 1 / (8*λ), there isn't a notorious difference.

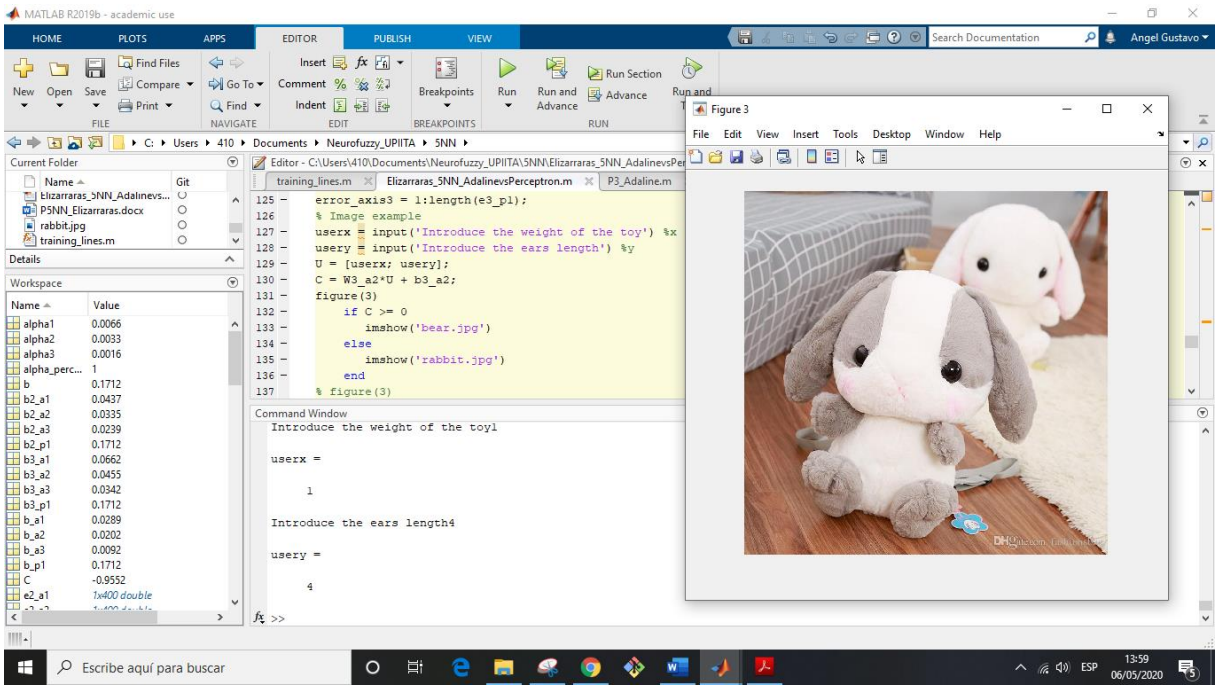
User input

The decided network for testing is Adaline when α = 1 / (8*λ) with 100 epochs.

The first one was a weight = 1 (small), and ears = 1 (small), so it would be a tiny bear.



The second one was a weight = 1 (small), and ears = 4 (large), so it would be a rabbit.



Conclusions

By the end of this practice I observed carefully the differences between Adaline and perceptron, they are very similar, but Adaline is better, because even that there is always an error in real applications it would be more accurate the sorting, if the decision boundary line separate the classes in the middle. Also, I implemented a function for the training in order to make shorter the code and faster the code running.

References

- https://datascience.stackexchange.com/questions/36368/what-is-the-difference-between-perceptron-and-adaline
- https://www.mathworks.com/help/deeplearning/ug/perceptron-neural-networks.html
- https://www.mathworks.com/help/deeplearning/ug/adaptive-neural-network-filters.html#bss4gnx-2

Annex

Generating the variables for the neuron

```
epochs1 = 20;%I did not ask to the user because if i do, i can't publish with matlab.
epochs2 = 50;%I did not ask to the user because if i do, i can't publish with matlab.
epochs3 = 100;%I did not ask to the user because if i do, i can't publish with matlab.
% alpha = 0.1;
%Rabbits
P1 = [1  4];
P2 = [1 5];
P3 = [2 4];
P4 = [2 5];
%Bears
P5 = [3 1];
P6 = [3 2];
P7 = [4 1];
P8 = [4 2];
t_P = [0 0 0 0 1 1 1 1];%Perceptron targets
t_A = [-1 -1 -1 -1 1 1 1 1];%Adaline targets
P = [P1; P2; P3; P4; P5; P6; P7; P8];%Points
num_patterns = 8;
N=1;%Number of neurons
W = rand(N,2)%random weigth
b = rand(N,1)%random bias
X = 0:0.1:6;
R = zeros(2,2);
```

Calculating Alphas

```
    for i = 1:num_patterns
        R = R + (1/num_patterns).*P(i,:)*P(i,:)';
    end
lamda = eig(R);
Lamdamax = max(lamda);
alpha_perceptron = 1;
alpha1 = 1/(4*Lamdamax);
alpha2 = 1/(8*Lamdamax);
alpha3 = 1/(16*Lamdamax);
```

Calling the function training and plotting them

20 epochs

```
[e_p1, W_p1, b_p1, Y_p1,Yw_p1] = training_lines(alpha_perceptron, epochs1, num_patterns, P, W,
b, t_P);
[e_a1, W_a1, b_a1, Y_a1,Yw_a1] = training_lines(alpha1, epochs1, num_patterns, P, W, b, t_A);
[e_a2, W_a2, b_a2, Y_a2,Yw_a2] = training_lines(alpha2, epochs1, num_patterns, P, W, b, t_A);
[e_a3, W_a3, b_a3, Y_a3,Yw_a3] = training_lines(alpha3, epochs1, num_patterns, P, W, b, t_A);
error_axis = 1:length(e_p1);
figure(1)
%Limit lines
subplot(2,2,1)
plot(X, Y_p1, X, Y_a1, X, Y_a2, X, Y_a3, 'Linewidth', 2)
title("Limit lines")
legend({'Y perceptron','Y adaline 1','Y adaline 2','Y adaline 3'},'Location','northeast')
hold on
```

```
scatter(1,4)
hold on
scatter(1,5)
hold on
scatter(2,4)
hold on
scatter(2,5)
hold on
scatter(3,1)
hold on
scatter(3,2)
hold on
scatter(4,1)
hold on
scatter(4,2)
hold on
%Perceptron error
subplot(2,2,2)
plot(error_axis, e_p1, 'Linewidth', 2)
title("Perceptron Error")
legend({'Error perceptron'},'Location','northeast')
%Adaline errors
subplot(2,2,[3 4])
plot(error_axis, e_a1, error_axis, e_a2, error_axis, e_a3)
title("Adaline Errors")
legend({'Error adaline, alpha (1/4)','Error adaline, alpha (1/8)','Error adaline,
alpha(1/16)'},'Location','northeast')
```

50 epochs

```
[e2_p1, W2_p1, b2_p1, Y2_p1,Yw2_p1] = training_lines(alpha_perceptron, epochs2, num_patterns,
P, W, b, t_P);
[e2_a1, W2_a1, b2_a1, Y2_a1,Yw2_a1] = training_lines(alpha1, epochs2, num_patterns, P, W, b,
t_A);
[e2_a2, W2_a2, b2_a2, Y2_a2,Yw2_a2] = training_lines(alpha2, epochs2, num_patterns, P, W, b,
t_A);
[e2_a3, W2_a3, b2_a3, Y2_a3,Yw2_a3] = training_lines(alpha3, epochs2, num_patterns, P, W, b,
t_A);
error_axis2 = 1:length(e2_p1);
figure(2)
%Limit lines
subplot(2,2,1)
plot(X, Y2_p1, X, Y2_a1, X, Y2_a2, X, Y2_a3, 'Linewidth', 2)
title("Limit lines")
legend({'Y perceptron','Y adaline 1','Y adaline 2','Y adaline 3'},'Location','northeast')
hold on
scatter(1,4)
hold on
scatter(1,5)
hold on
scatter(2,4)
hold on
scatter(2,5)
hold on
scatter(3,1)
hold on
scatter(3,2)
hold on
scatter(4,1)
```

```
hold on
scatter(4,2)
hold on
%Perceptron error
subplot(2,2,2)
plot(error_axis2, e2_p1, 'Linewidth', 2)
title("Perceptron Error")
legend({'Error perceptron'},'Location','northeast')
%Adaline errors
subplot(2,2,[3 4])
plot(error_axis2, e2_a1, error_axis2, e2_a2, error_axis2, e2_a3)
title("Adaline Errors")
legend({'Error adaline, alpha (1/4)','Error adaline, alpha (1/8)','Error adaline,
alpha(1/16)'},'Location','northeast')
```

100 epochs

```
[e3_p1, W3_p1, b3_p1, Y3_p1,Yw3_p1] = training_lines(alpha_perceptron, epochs3, num_patterns,
P, W, b, t_P);
[e3_a1, W3_a1, b3_a1, Y3_a1,Yw3_a1] = training_lines(alpha1, epochs3, num_patterns, P, W, b,
t_A);
[e3_a2, W3_a2, b3_a2, Y3_a2,Yw3_a2] = training_lines(alpha2, epochs3, num_patterns, P, W, b,
t_A);
[e3_a3, W3_a3, b3_a3, Y3_a3,Yw3_a3] = training_lines(alpha3, epochs3, num_patterns, P, W, b,
t_A);
error_axis3 = 1:length(e3_p1);
figure(3)
%Limit lines
subplot(2,2,1)
plot(X, Y3_p1, X, Y3_a1, X, Y3_a2, X, Y3_a3, 'Linewidth', 2)
title("Limit lines")
legend({'Y perceptron','Y adaline 1','Y adaline 2','Y adaline 3'},'Location','northeast')
hold on
scatter(1,4)
hold on
scatter(1,5)
hold on
scatter(2,4)
hold on
scatter(2,5)
hold on
scatter(3,1)
hold on
scatter(3,2)
hold on
scatter(4,1)
hold on
scatter(4,2)
hold on
%Perceptron error
subplot(2,2,2)
plot(error_axis3, e3_p1, 'Linewidth', 2)
title("Perceptron Error")
legend({'Error perceptron'},'Location','northeast')
%Adaline errors
subplot(2,2,[3 4])
plot(error_axis3, e3_a1, error_axis3, e3_a2, error_axis3, e3_a3)
title("Adaline Errors")
```

```
legend({'Error adaline, alpha (1/4)','Error adaline, alpha (1/8)','Error adaline,
alpha(1/16)'},'Location','northeast')
```

Training function

```
function [err, W, b, Y, Yw] = training_lines(alpha, epochs, num_patterns, P, W, b, target)
g = zeros(1,8);
err = [];
X = 0:0.1:6;
% Training
if alpha == 1
    for i = 1:epochs
        for j = 1:num_patterns

        a = hardlim(W*P(j,:)' + b);
        e = target(j) - a;
        x = alpha*e*P(j,:);
        W = W + x;
        b = b + alpha*e;
        g(j) = e;
        end
        err = [err,g];
    end
else
      for i = 1:epochs
        for j = 1:num_patterns

        a = W*P(j,:)' + b;
        e = target(j) - a;
        x = alpha*e*P(j,:);
        W = W + x;
        b = b + alpha*e;
        g(j) = e;
        end
        err = [err,g];
      end
end

% Getting the limit line
xpoint = -b / W(1);
ypoint = -b / W(2);
slope = -ypoint/xpoint;
Y = slope*X + ypoint;
mw = -1 / slope;
Yw = mw*X;
end
```

Show image

```
% Image example
userx = input('Introduce the weight of the toy') %x
usery = input('Introduce the ears length') %y
U = [userx; usery];
C = W3_a2*U + b3_a2;
figure(3)
    if C >= 0
       imshow('bear.jpg')
    else
       imshow('rabbit.jpg')
    end
```