Perceptron Learning rule

By: Group 12

CSS 485

Professor Michael Stiber, Ph D.

Team Member: Jun H Park, Thomas Hedrick

TABLE OF CONTENTS

Contents

STATE OF WORK

The following table contains contents of the homework with contributor to that contents.

Table 0: Contribution of Work

Contents	Contributor(s)
E 4.1 Draw a diagram of the single-neuron perceptron you would use to solve the problem and draw the graph with data points to identify if the problem is solvable or not.	Jun & Thomas
E 4.4 Start from Weight [0 0] and bias 0, solve the classification problem by using given input patterns with output target.	Jun & Thomas
E 4.11 .i && .ii Just like E 4.4, it is classification problem with given input patterns and target output 0 and 1 to train the network to get reasonable weight and bias	Jun & Thomas

INTRODUCTION

The purpose of this group project is to learn and apply the concept of perceptron learning rule. By solving problems in the book (E4.1, E4.4, E4.11. i, and E4.11. ii), we were able to apply our knowledge about weight vector, bias, the decision boundary, and perceptron learning rule especially error with finding new weight and bias.

INTRODUCTION FOR E4.1

Problem E4.1 requires representing correct diagram of the single-neuron perceptron, which requires understands the differences between input patterns and inputs. Then, the problem E4.1 requires represent the graph of the data points and verify if the problem is solvable or not. To accomplish the following two problems, group 18 had processed the following method in Method For 4.1 section below.

METHOD 4.1

The first approach that we took was analyzing the number of inputs and outputs. Since the given output was 1 and 0, we decided to use hardlim as our activation function for the single perceptron equation, which is represented in Figure 1.0.

After we came up with the basic single neuron perceptron equation, we decided to draw a 2D graph and plot the target outputs corresponding to its inputs, which is represented in Figure 1.1 located in the Results for 4.1 section.

Based on the graph we had in Figure 1.1, we came up with a decision boundary, weight vector, and a bias by using the following steps

Step 1: draw a reasonable decision boundary based on the targets in graph.

Step 2: Choose a weight vector

- In this case, we had chosen [-1 -1].

Step 3: Find a bias

- First, we had picked a point on the decision boundary, [0.5 0]
- Then apply weight vector with point we had chose
  - $W^T P + b \Rightarrow [-1\ -1] * [0.5; 0] + b = 0$
  - $b = 0.5$

Result For 4.1

This section covers the overall results of problem 4.1 based on suggested method in method 4.1 section.

Figure 1.0: Diagram of Single Neuron Perceptron

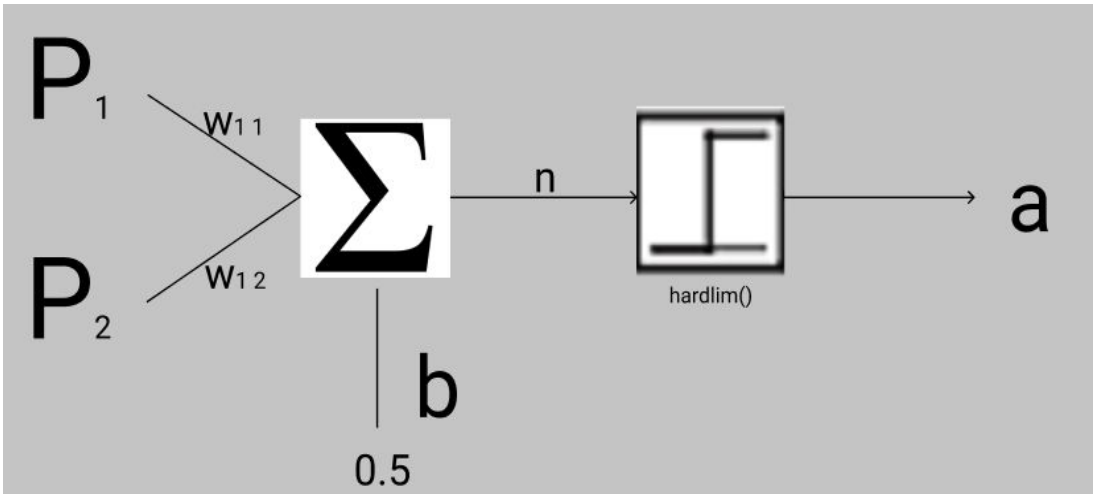

Figure 1.0. Representation of Single Neuron Perceptron, which uses hardlim as activate function. Takes two inputs (2 x 1) and conduct the inner product with weight vector then add bias which produce summation or n. Then use n value as input of activate function to produce an output a.

There were 2 inputs required

Figure 1.1: Visualization of Target Points

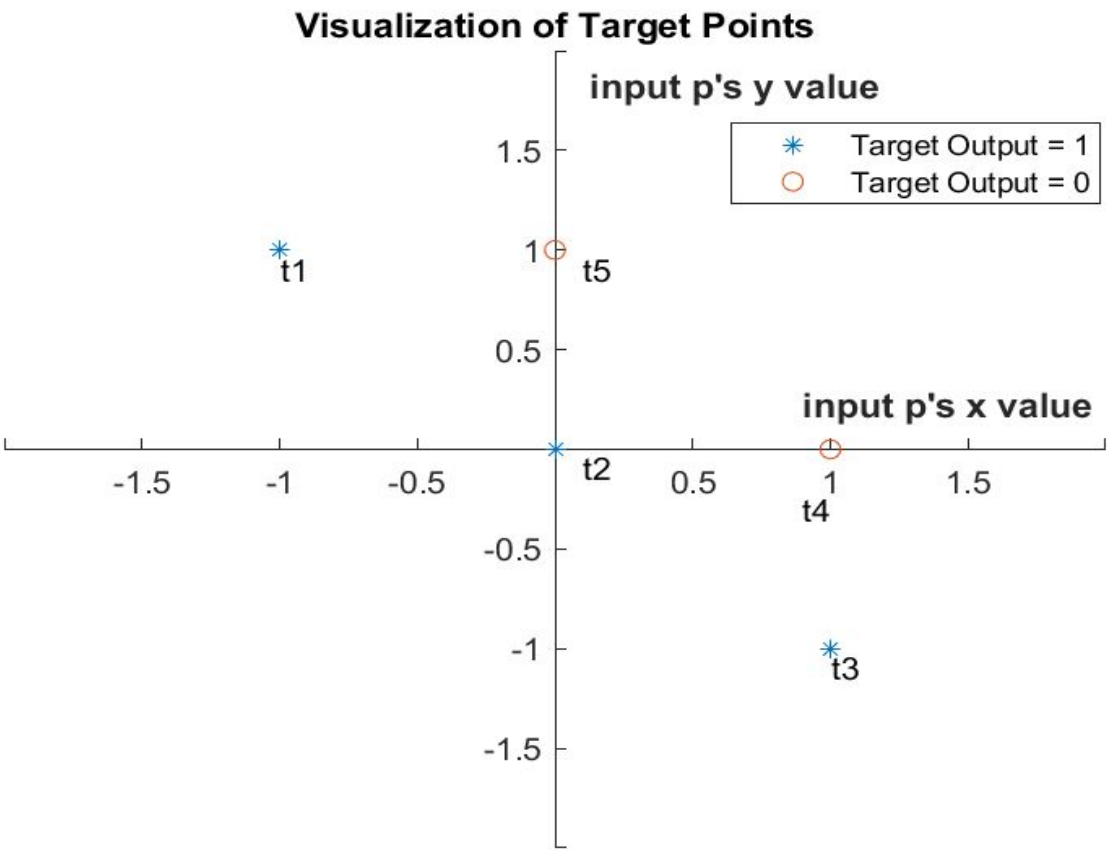

Figure 1.1. Visualization of target points, which is representing the inputs of input patterns labeled according to their targets.

This graph is solvable. Clearly, there is a decision boundary that exists, for instance, we can draw a line x = 0.5 and y = 0.5 as shown in Figure 1.2 below.

Figure 1.2: Visualization of Target Points with Decision Boundary

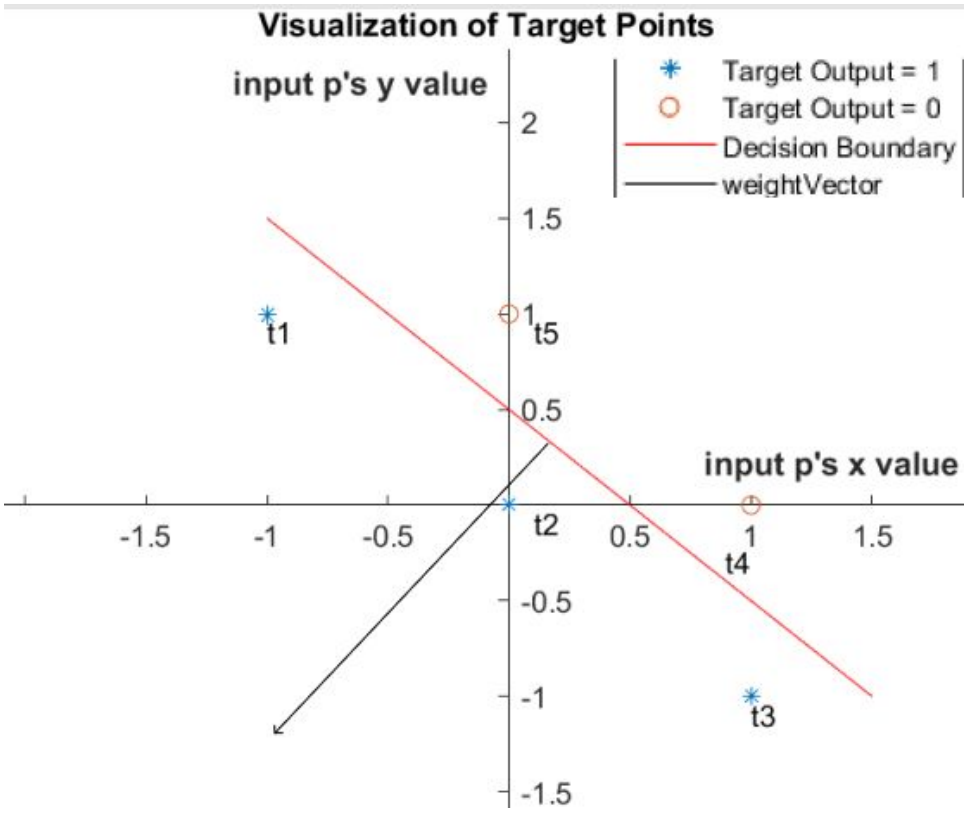

Figure 1.2. Visualization of target points, which is representing the inputs of input patterns labeled according to their targets with reasonable decision boundary to classify target outputs with weight vector point toward side of t1, t2, and t3.

To verify our steps in 4.1 method we used Matlabs, which represented in Matlab Code 1.0 below.

Matlab Code 1.0: Single Neuron Perceptron

```
format compact
p1Input = [-1;1]
p2Input = [0; 0]
p3Input = [1;-1]
p4Input = [1;0]
p5Input = [0;1]

myWeight = [-1 -1]
bias = 0.5

summation = n(p1Input, myWeight, bias)
hardlimFunc(summation)
```

```
function axion = n(input_, weight_, bias)
    axion = weight_ * input_ + bias
end

function myFunc = hardlimFunc(input_)
   myFunc = hardlim(input_)
end
```

Result of the Matlab code

Input	Output
[-1 ; 1]	1
[0 ; 0]	1
[1 ; -1]	1
[1 ; 0]	0
[0 ; 1]	0

Therefore, the problem is solvable because the outputs using Matlabs match the outputs that were given in the question.

Introduction For E4.4

Problem E4.4 requires applying the perceptron rule by using given weight vector [0 0] with bias 0 and train by using the given inputs in E4.2 to find reasonable weight vector and bias. The goal for the problem is to find weight vector and bias and classify given inputs which suggested in E4.2. iii. To accomplish the following two problems, group 12 had processed the following method in Method For 4.4 section below.

Method 4.4

The first approach we did was to visualize ourselves if the problem is solvable with a decision boundary, which is represented in Figure 2.0 located at Result section.

After we confirmed that the problem is solvable, we started to set up the possible equations we needed, which are

- a = hardlim(wp + b)
- Error = target - a
- $weight^{New} = weight^{Old} + Error * P$
- $Bias^{New} = Bias^{Old} + Error$

Using these equations above, we trained the weight and bias so that all results have the correct target outputs for the input patterns, which is represented in table 1.0: Iteration of the Training for Weight Vector and Bias.

Lastly, we had used the weight vector and bias that we calculated based on training representing in table 1.0 to get target output for unknown target outputs for inputs p_5 , p_6, p_7, and p_8.

Result for 4.4

This section covers overall results of E4.4, which led to E4.2 problem ii and iii based on the method group 12 suggested in above method section for 4.4.

Figure 2.0: Graph Representation of Targets Corresponding to Inputs

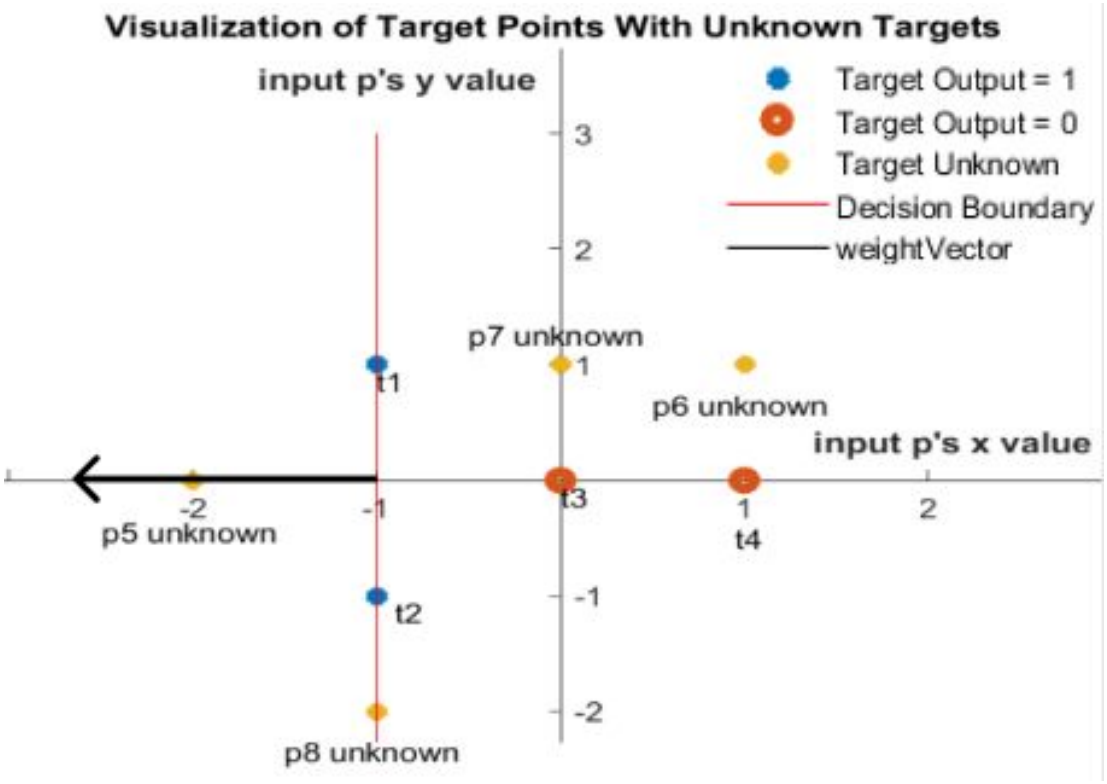

Figure 2.0. represents the given input patterns. The points with labeled t are represent inputs given targets and labeled with p are unknown target. The visualization graph also represents our decision boundary which is the red line with weight vector.

Table: 1.0: Iteration of the Training for Weight Vector and Bias

input	Target	a	weight	bias	error
P1 [-1 1]	1	1	[0 0]	0	N/A
P2 [-1 -1]	1	1	[0 0]	0	N/A
P3 [0 0]	0	1	[0 0]	-1	-1
P4 [1 0]	0	0	[0 0]	-1	N/A
P1 [-1 1]	1	0	[-1 1]	0	1
P2 [-1 -1]	1	1	[-1 1]	0	N/A
P3 [0 0]	0	1	[-1 1]	-1	-1
P4 [1 0]	0	0	[-1 1]	-1	N/A
P1 [-1 1]	1	1	[-1 1]	-1	N/A
P2 [-1 -1]	1	0	[-2 0]	0	1

P3 [0 0]	0	1	[-2 0]	-1	-1
P4 [1 0]	0	0	[-2 0]	-1	N/A
P1 [-1 1]	1	1	[-2 0]	-1	N/A
P2 [-1 -1]	1	1	[-2 0]	-1	N/A
P3 [0 0]	0	0	[-2 0]	-1	N/A
P4 [1 0]	0	0	[-2 0]	-1	N/A

Table: 1.0. represents the iteration of the training inputs and its corresponding target output based on new weights and new bias. Table: 1.0. contains 6 columns. The column "input" represents the inputs, which is representing given input vectors in input patterns used in training algorithm. The column "Target" represents the target output of the given input. The column "a" represents output value of our training algorithm. The column "weight" represents new weight based on the equation group 12 suggested in method 4.4 section. The column "bias" represents the values of new bias based on the equation group 12 suggested in method 4.4 section. Last, "error" column represents if there are error exist by comparing target to a.

Final results of training

Weight: [-2 0]
Bias: -1

2) Solve E.4.2iii

Weight: [-2 0]
Bias: -1

Formula: hardlim(W * p + b)

Given input patterns

P_5= [-2 ; 0]
hardlim([-2 0] * [-2; 0] + (-1)) = 1

P_6= [1 ; 1]
hardlim([-2 0] * [1 ;1] + (-1)) = 0

P_7= [0 ; 1]
hardlim([-2 0] * [0; 1] + (-1)) = 0

P_8= [-1; -2]
hardlim([-2 0] * [0; 1] + (-1)) = 1

Introduction For E4.11

Problem E4.11 requires of classification of the rabbit and bear which represent rabbit as 0 and bear as 1. The problem suggested to start the problem by finding out the reasonable weight vector with bias by training based on the input patterns and target outputs. Then test out the inputs by using weight vector and bias, if that give target outputs. To accomplish the problem, the following method in Methods for 4.11. i and 4.11. ii had used.

Methods for 4.11.i and 4.11.ii

First, we had decided to visualize the target outputs corresponding to inputs pattern on the 2D graph to confirm if the problem is solvable, which is represented in figure 3.0 in Result for 4.11 and 4.11.ii section.

After we had confirmed that the problem is solvable, we decided to use hardlim as activation function because of its target outputs. Then we converted the equations we had in 4.4 Method section into matlab, which can be seen in Matlab 2.0 in Result for 4.11. i and 4.11.ii section. Last, we apply the logic of iteration to train the weight vector and bias.

Once we got the trained weight vector and bias, we tested with given inputs to verify its results of matlab result are match with target outputs, which is represented in Table 2.0.

Result for 4.11 .i and 4.11. ii

This section covers overall results of E4.11. i and 4.11 .ii , based on the method group 12 suggested in above method section for 4.11.

Figure 3.0: Visualization of Target Points

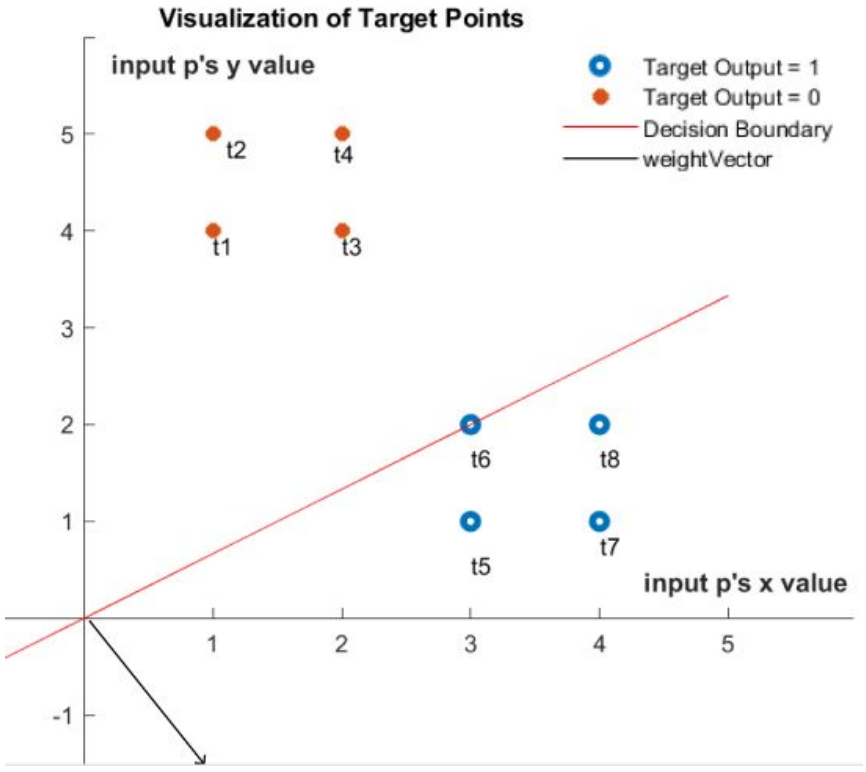

Figure 3.0. represents the given input patterns. The points with labeled t are represent inputs given targets. There are two types of target value exist in this graph, which are 0 and 1 and those target values are separated by decision boundary.

Matlab 2.0: Perceptron Training

```
format compact

p1 = [1; 4];
p2 = [1; 5];
p3 = [2; 4];
p4 = [2 ; 5];
p5 = [3; 1];
p6 = [3; 2];
p7 = [4; 1];
p8 = [4; 2];
inputs = [p1 p2 p3 p4 p5 p6 p7 p8];
targets = [0 0 0 0 1 1 1 1];

weight = [0 0];
bias = 0;
% ouptut/store for weight and bias
[weight, bias] = perceptronTraining(inputs, targets, weight, bias)

% function perceptronTraining returns two values (trained weights and bias)
function [weight, bias] = perceptronTraining(inputs_, targets_, weight_, bias_)
   % if target and axion is not equal
   % train the weight and bias until target and axion is equal
   % weight and bias can be updated by following formula
   % w(new) = w(old) + ep
   % b(new) = b(old) + e
   while 1
      allCorrect = 1;
      for i = 1:length(inputs_)
        a = hardlim(weight_ * inputs_(:, i)+ bias_);
        if targets_(i) ~= a
           error = targets_(i) - a;
           weight_ = weight_ +  error * inputs_(:, i)';
           bias_ = bias_ + error;
           allCorrect = 0;
        end
      end
      if allCorrect
        break;
      end
   end
   weight = weight_;
   bias = bias_;
end
```

Matlab 2.1: Test the resulting weight and bias values

```
format compact

p1 = [1; 4];
p2 = [1; 5];
p3 = [2; 4];
p4 = [2 ; 5];
p5 = [3; 1];
p6 = [3; 2];
p7 = [4; 1];
p8 = [4; 2];
inputs = [p1 p2 p3 p4 p5 p6 p7 p8];
targets = [0 0 0 0 1 1 1 1];
% Before Trained
weight = [0 0];
bias = 0;
% Trained ouptut for weight and bias
[weight, bias] = perceptronTraining(inputs, targets, weight, bias)
% 4.11.ii
for i = 1:length(inputs)
   target = hardlim(weight * inputs(:, i)+ bias)
end
% end of 4.11.ii
```

Table 2.0: Matlab Result using Perceptron Trained Weight and Bias

Input	**Target**	**Matlab target result**
P_1 = [1; 4]	0	0
P_2 = [1; 5]	0	0
P_3 = [2; 4]	0	0
P_4 = [2; 5]	0	0
P_5 = [3; 1]	1	1
P_6 = [3; 2]	1	1
P_7 = [4; 1]	1	1
P_8 = [4; 2]	1	1

Table 2.0. represents overall output of group 12's training algorithm. The table column, "input" represents the test inputs, "Target" represents the target output of the inputs, and "Matlab target result" represents the output by using trained weight and bias with given inputs.

Conclusion

Starting with 4.1 we learned that inputs and outputs can be drawn to show how a weight and bias can be chosen by using decision boundaries. Then in 4.4 we learned that this could be done the same way by using the perceptron learning rule that decides the weight and bias for us. This however was very time consuming to do by hand since it has to iterate many times until the weight and bias worked for all the inputs. This leads to 4.11 where we translate the perceptron learning rule to code to make it much faster and able to handle many more inputs than we could have done trying to do it by hand.