Perceptron

The perceptron is simplest neural network based algorithm.
It only has one input layer and one output layer.

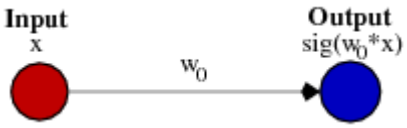
About the transfer function

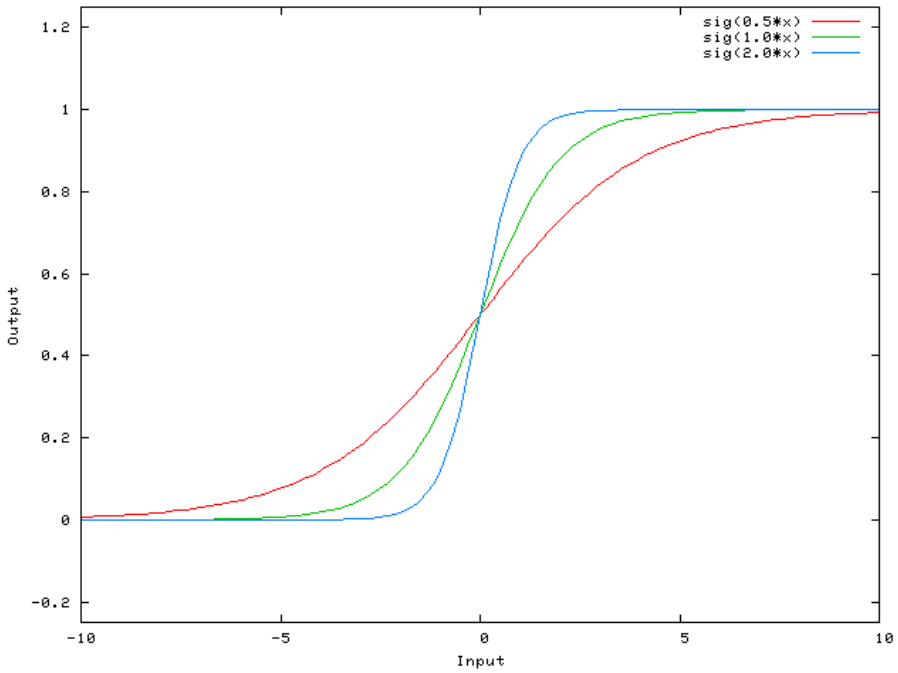
Sigmoid is used for classification due to his nice derivative. For regression, other functions are used. However, it can also be used for the simple perceptron the step function which changes its value quickly. Nonetheless, since the step is discontinuous, it cannot be differentiate, in contrast with sigmoid.

Why a bias is needed?

The easiest way to understand why we need a bias is because, if we don't have it, the perceptron cannot learn if it has a non-zero input and a zero output or viceversa. This is because when the sum is computed, if all inputs are zero the result will be zero and the output will be zero as well. Hence, there is no way the weights can learn something, because the result will be always zero.

A more mathematical explanation is that the steepness of the transfer function (sigmoid) can be modified easily without any bias. In the following example, we have a one-to-one network.

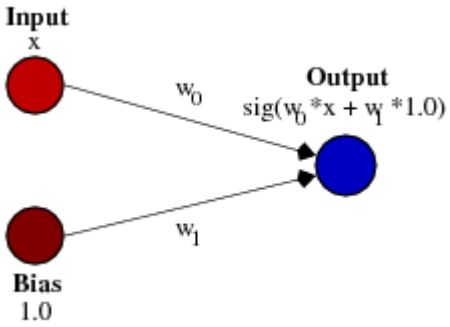


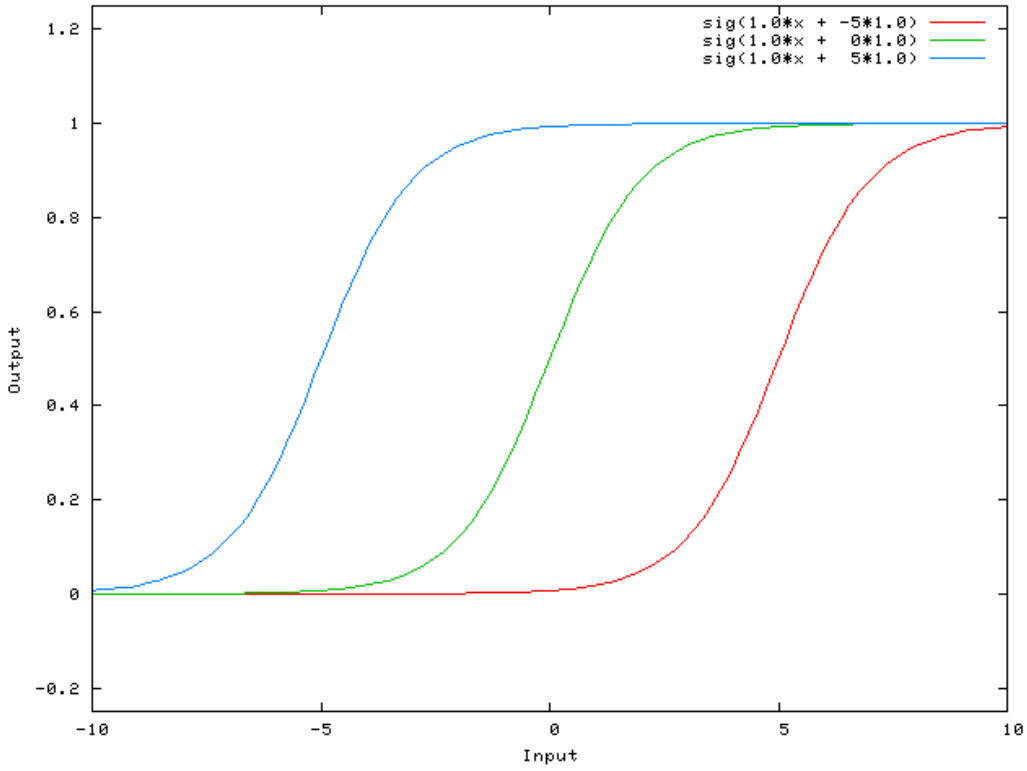


When we have a x dependent argument, the steepness of the sigmoid function can be modified.

However, there is no way we can go anyplace further than 0 when the input is 0.

For this, we need a bias.





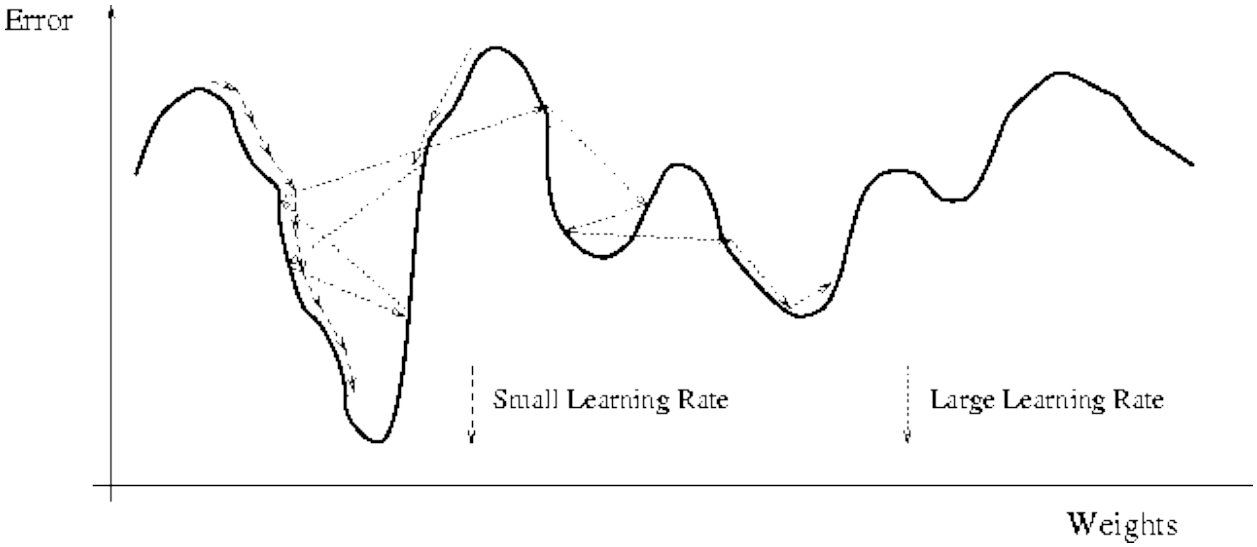
Now, the sigmoid curve can be shifted and we can, for example, get an output of 0 when the input is 2. Usually, bias is -1 or 1.

The AND example code shown later was used to try to understand whether -1 or 1 it is better. Since the [0 0 → 0] exists in the AND example (and therefore, it really does not matter that at 0 0 the algorithm learns something different than zero), the NAND logic gate was used. In this case, [0 0 → 1] so it makes more sense to use a positive bias because for input 0, we get a positive result. The amount of iterations has been measured and averaged, and in case of positive bias we get a slightly better result than with -1.

Why deciding the proper learning rate is important?

Having a small learning rate may slow down considerably the speed of the learning algorithm and it will need more iterations and consequently more time and resources.

Having a big learning rate may make the algorithm jump from the minimum (as we can see in the picture) and end up climbing the mountain.

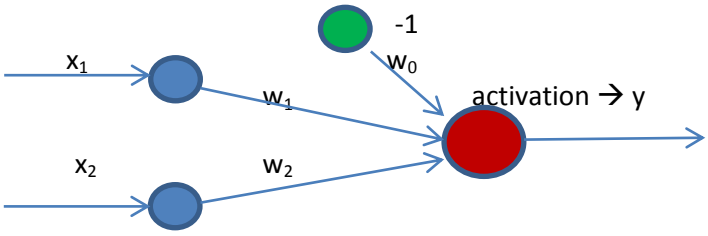


Usually, the learning rate is $0.1 < \eta < 0.4$ since the weights are small numbers, usually initialized randomly between 0 and 1.

Example 1: AND

In this case we have a very simple diagram with only two neurons as input and one as output. Additionally, we have a bias connected to the output.

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1



To solve this problem, we can iterate as many times as needed over our training set until we get no errors.

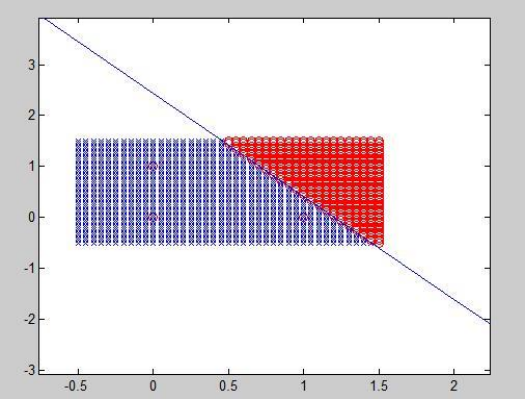
$$\sum_{i=0}^{N} w_i \cdot x_i$$

To learn, we apply this:

$$w_i = w_i + \eta(t_i - y_i)x_i$$

After we finish iterating with no errors, we can build up the decision boundary using this formula:

$$y = \frac{-w_1 x + w_0}{w_2} \longrightarrow w_2 y + w_1 x = w_0$$

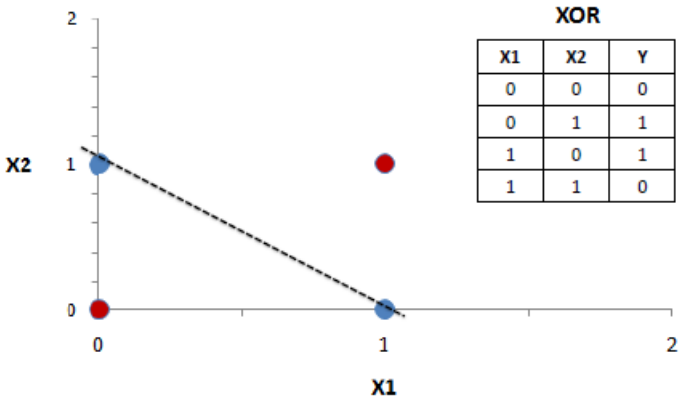


If we think about that, all elements at one side of the previous formula belong to a class whereas at the other side they belong to the other class.

In this picture we can see an AND example where all points in a mesh have been drawn, as well as the decision boundary. Code available in the Apendix I

Example 2: XOR

This is the best example to see how Perceptron fails at classifying XOR since it is not possible to separate both classes using only one line, in contrast with other logic functions.



XOR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

If we try to use the perceptron here, it will endlessly try to converge.

Example 3: Evolution of the decision boundary

Adding new data:

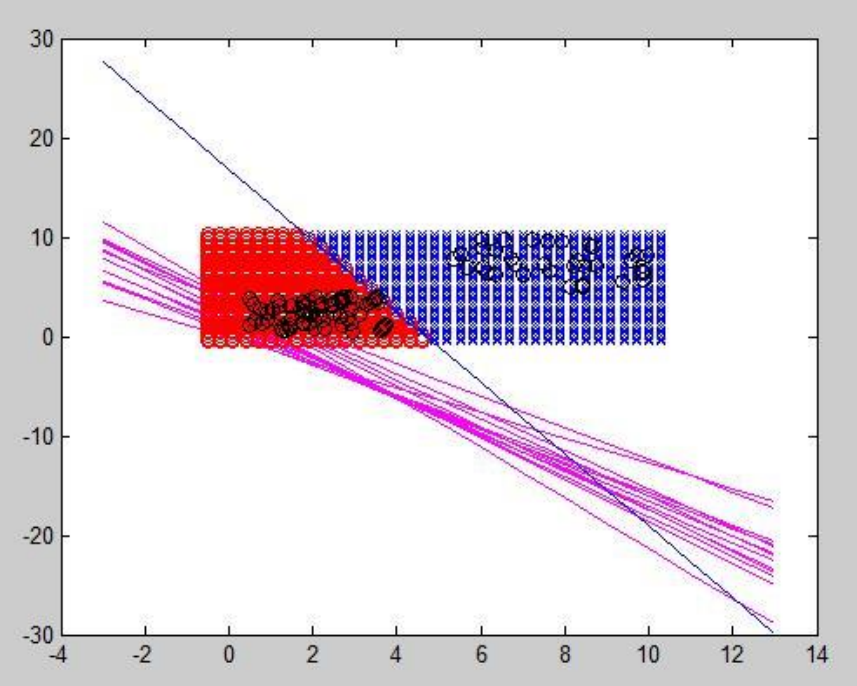
```
data = 0.5 + (4-0.5).*rand(66,2);
data = [data ones(66,1)];

data2 = 5 + (10-5).*rand(34,2);
data2 = [data2 zeros(34,1)];

data = [data;data2]
```

And plotting it in each iteration:

```
x = -3:13;
```

```
y = (-weights(2)*x+weights(1))/weights(3);
plot(x,y,'m');
```

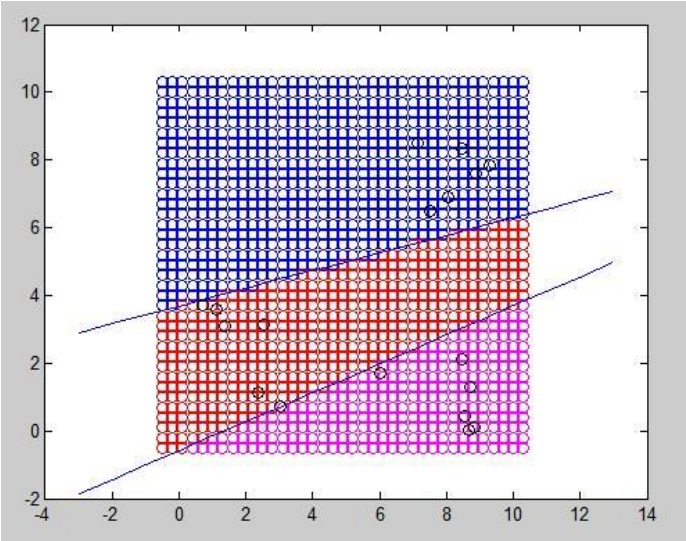
We can see the evolution of the decision boundary:



By the way, in this example, we can see that boundaries are not located at the same distance. That is because we have different amounts of samples. Using the same amount, the distance between the closest elements is the same.

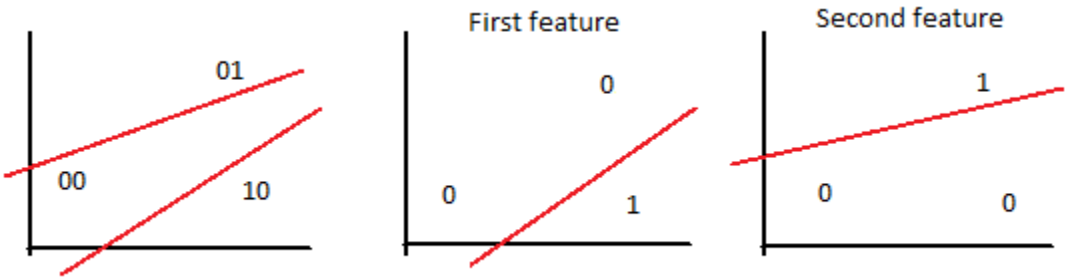
Example 4: Distinguishing more than 2 classes

One may think that we can classify 3 or 4 classes using 2 output neurons. This is because we can have a target vector so that we can encode 2^N different classes. For example, we have classes A, B, C and D, and two neurons, so when output is 00 it belongs to A, when it is 01 it belongs to B and so on. However, this does not always work, because each output neuron and the weights which come from the inputs can separate only one feature.

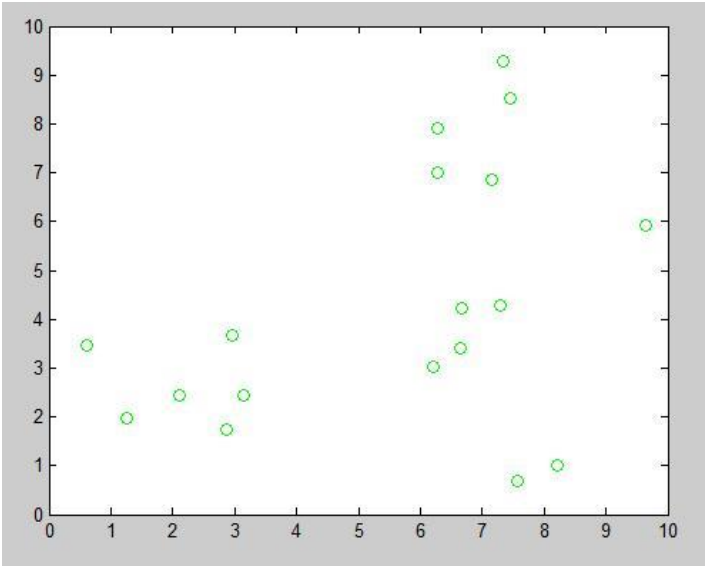


As we can see here, we have 3 different elements which can be classified perfectly. However, the perceptron which only has 2 output neurons works as follows: it distinguishes one feature from another.

In this example, we have classes B (blue), R (red) and M (magenta) which respectively corresponds to 01, 00 and 10.



This is how the algorithm plots each line. Therefore, we can say that we will have the same amount of lines than output neurons, whose characteristics lie upon the weights connected to the inputs.



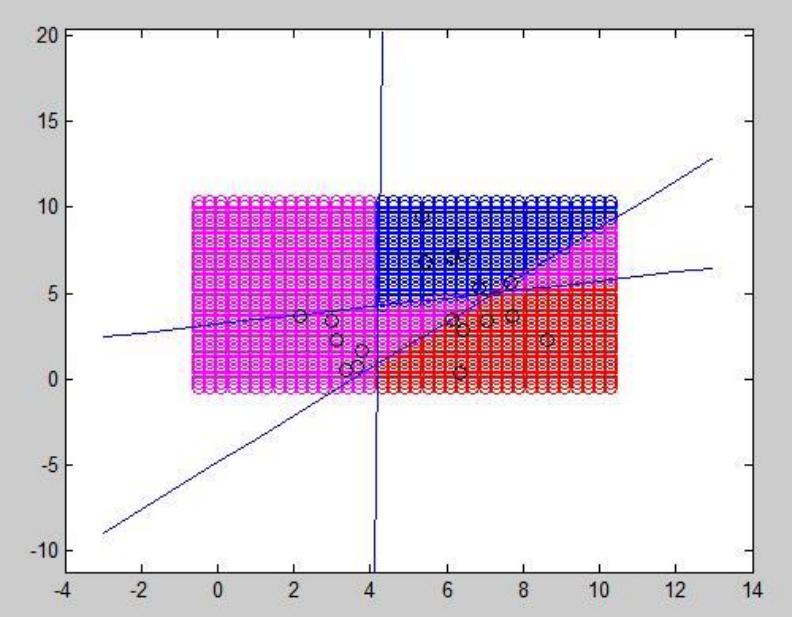
And here we can see an example when it is not possible the classification and therefore, the algorithm will endlessly work.

Each class has 6 samples, and we can see that it's not possible to separate M class from the rest using a straight line.

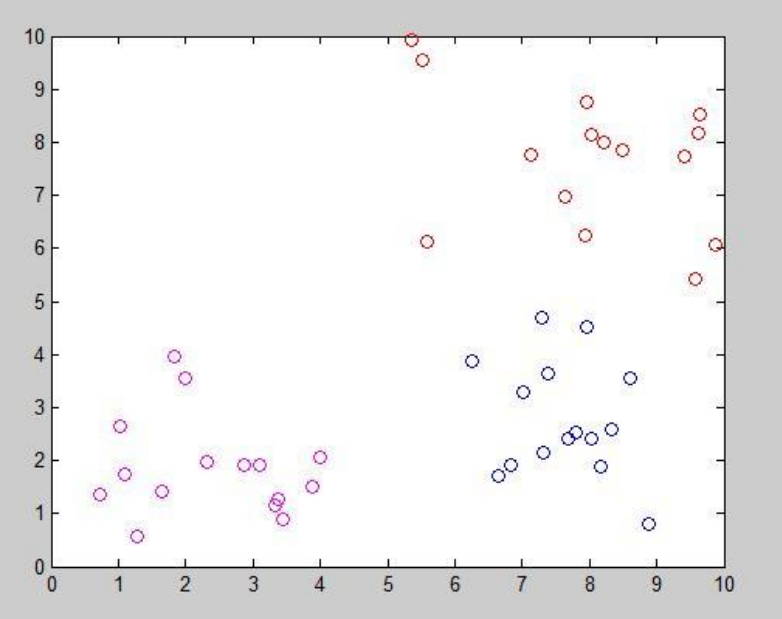
We can see the code in the second appendix.

What if we use N output neurons?

You also may think that using N output neurons can be a good idea, and code our classes B, R, M as (0,0,1), (0,1,0) and (1,0,0) so that each has a unique characteristic that the algorithm can distinguish from the others.



And in practice, it may work sometimes. Here we can see that each line uniquely separate each group (nonetheless it has an odd behavior since we can see those magenta sections where makes no sense to classify magenta elements).



However, in cases such this, it is not possible to separate with a straight line all classes uniquely, specifically the blue one. In this case, the algorithm will be running endlessly.

So, this algorithm will have higher chances to work with few elements in each class. Code is in appendix three.

Appendix 1: AND example

```
function [ output ] = activationstep( input, type )
    if strcmp(type,'step')
       if(input>0)
           output=1;
       else
           output=0;
       end
    elseif strcmp(type,'sigmoid')
        output=1/(1+exp(-input));
    end
end
```

```
clear;clc;

% AND
data = [0 0 0; 0 1 0; 1 0 0; 1 1 1];
%data = [-1 -1 -1; -1 1 -1; 1 -1 -1; 1 1 1];
% OR
%data = [0 0 0; 0 1 1; 1 0 1; 1 1 1];
% XOR
%data = [0 0 0; 0 1 1; 1 0 1; 1 1 0];


% Input = bias + the rest without last row (targets)
input = [-1*ones(size(data,1),1),data(:,1:end-1)];
target = data(:,end);

%Learning rate
eta = 0.5;
% Amount of features
inputNeurons = size(data,2)-1;
% Amount of inputs for training (4 cases)
inputCases = size(data,1);

disp('First weights');
weights = rand(inputNeurons+1,1)';
disp(weights);

% Plot elements
plot(input(:,2),input(:,3),'ro')
hold on;

% Training until it has no errors
repeat=1;
counter=1;
while repeat==1
    repeat=0;
    fprintf('Counter: %i.\n',counter)
    for j=1:inputCases
        y=activation(input(j,:)*weights','step');
```

```
        if target(j)~=y
        %if abs(target(j)-y)>0.01 We have to use this with sigmoid since
        %it's regression

           fprintf('Target: %i, y: %i, j: %i\n',target(j),y,j);

           weights = weights + eta*(target(j)-y)*input(j,:);
           repeat=1;
        end

    end
    counter=counter+1;

end
disp('Final weights');
disp(weights);

% Ploting the mesh
for i=-0.5:0.05:1.5
   for j=-0.5:0.05:1.5
        % Change this if using +1 as bias
        res=activation(-weights(1)+weights(2)*i+weights(3)*j,'step');
        if res==1
        %if res>0.5 % regresion
            plot(i,j,'ro');
        else
            plot(i,j,'bx');
        end
   end
end

x = -3:3;
% Decision boundary
% Change this if using +1 as bias
y = (-weights(2)*x+weights(1))/weights(3);
plot(x,y);
hold off;
```

Appendix 2: 3 classes with two output neurons

```
clear;clc;

eachGroup=6;

data = 0.5 + (4-0.5).*rand(eachGroup,2);
data = [data zeros(eachGroup,2)];

data2 = 5 + (10-5).*rand(eachGroup,2);
data2 = [data2 zeros(eachGroup,1) ones(eachGroup,1)];

data31 = 6 + (9-6).*rand(eachGroup,1);
data32 = 0 + (5-0).*rand(eachGroup,1);
data3 = [data31 data32 ones(eachGroup,1) zeros(eachGroup,1)];

data = [data;data2;data3];



% Input = bias + the rest without last row (targets)
input = [-1*ones(size(data,1),1),data(:,1:end-2)]
% Last two columns are now target
target = data(:,end-1:end)

%Learning rate
eta = 0.5;
% Amount of features. -2 last rows
inputNeurons = size(data,2)-2;
% Amount of inputs for training (4 cases)
inputCases = size(data,1);


disp('First weights');
weights1 = rand(inputNeurons+1,1)';
weights2 = rand(inputNeurons+1,1)';
disp(weights1);
disp(weights2);

% Plot elements
plot(input(:,2),input(:,3),'go')
hold on;

% Training 10 times
repeat=1;
counter=1;
while repeat==1
    repeat=0;
    fprintf('Counter: %i.\n',counter)
    for j=1:inputCases
        y1=activation(input(j,:)*weights1','step');
        y2=activation(input(j,:)*weights2','step');
        if target(j,1)~=y1
```

```
        %if abs(target(j)-y1)>0.5 % We have to use this with sigmoid since
        %it's regression
           fprintf('Target: %i, y: %i, j: %i\n',target(j,1),y1,j);
           weights1 = weights1 + eta*(target(j,1)-y1)*input(j,:);
           repeat=1;
        end

        if target(j,2)~=y2
        %if abs(target(j)-y2)>0.5 % We have to use this with sigmoid since
        %it's regression
           fprintf('Target: %i, y: %i, j: %i\n',target(j,2),y2,j);
           weights2 = weights2 + eta*(target(j,2)-y2)*input(j,:);
           repeat=1;
        end
    end
    counter=counter+1;


end
disp('Final weights');
disp(weights1);
disp(weights2);


% Ploting the mesh
for i=-0.5:0.3:10.5
   for j=-0.5:0.3:10.5
       % Change this if using +1 as bias
        res1=activation(-weights1(1)+weights1(2)*i+weights1(3)*j,'step');
        res2=activation(-weights2(1)+weights2(2)*i+weights2(3)*j,'step');

        if res1==0 && res2==0
        %if res>0.5 % regresion
            plot(i,j,'ro');
        elseif res1==0 && res2==1
            plot(i,j,'bo');
        else
            plot(i,j,'mo');
        end
   end
end

x = -3:13;
% Decision boundary
% Change this if using +1 as bias
y = (-weights1(2)*x+weights1(1))/weights1(3);
plot(x,y);
y = (-weights2(2)*x+weights2(1))/weights2(3);
plot(x,y);
plot(input(:,2),input(:,3),'ko')
hold off;
```

Appendix 3: N output neurons

```
clear;clc;

eachGroup=15;
totalClasses=3;

data = 0.5 + (4-0.5).*rand(eachGroup,2);
data = [data zeros(eachGroup,2) ones(eachGroup,1)];

data2 = 5 + (10-5).*rand(eachGroup,2);
data2 = [data2 zeros(eachGroup,1) ones(eachGroup,1) zeros(eachGroup,1)];

data31 = 6 + (9-6).*rand(eachGroup,1);
data32 = 0 + (5-0).*rand(eachGroup,1);
data3 = [data31 data32 ones(eachGroup,1) zeros(eachGroup,1)
zeros(eachGroup,1)];

data = [data;data2;data3];

% Input = bias + the rest without last row (targets)
input = [-1*ones(size(data,1),1),data(:,1:end-totalClasses)]
% Last N columns are now target
target = data(:,end-(totalClasses-1):end)

%Learning rate
eta = 0.5;
% Amount of features. -N last rows
inputNeurons = size(data,2)-totalClasses;
% Amount of inputs for training (4 cases)
inputCases = size(data,1);


disp('First weights');
weights = rand(inputNeurons+1,totalClasses)';
disp(weights);

% Plot elements
plot(input(1:eachGroup,2),input(1:eachGroup,3),'mo')
hold on;
plot(input(eachGroup+1:eachGroup*2,2),input(eachGroup+1:eachGroup*2,3),'ro')
plot(input(eachGroup*2+1:eachGroup*3,2),input(eachGroup*2+1:eachGroup*3,3),'b
o')

% Training 10 times
repeat=1;
counter=1;
while repeat==1
    repeat=0;
    fprintf('Counter: %i.\n',counter)

    for j=1:inputCases
```

```
        for k=1:totalClasses
            y=activation(input(j,:)*weights(k,:)','step');

            if target(j,k)~=y
            %if abs(target(j)-y1)>0.5 % We have to use this with sigmoid
since
            %it's regression
               fprintf('Target: %i, y: %i, j: %i\n',target(j,k),y,j);
               weights(k,:) = weights(k,:) + eta*(target(j,k)-y)*input(j,:);
               repeat=1;
            end

        end
    end
    counter=counter+1;

end
disp('Final weights');
disp(weights);


% Ploting the mesh
for i=-0.5:0.3:10.5
   for j=-0.5:0.3:10.5
       % Change this if using +1 as bias
       for k=1:totalClasses
            res(k)=activation(-
weights(k,1)+weights(k,2)*i+weights(k,3)*j,'step');
       end
       index=find(res); %it finds the index of non-zero values

       %if we change totalClasses, we have to change this to make it work
        if index==1
            plot(i,j,'ro');
        elseif index==2
            plot(i,j,'bo');
        else
            plot(i,j,'mo');
        end
   end
end

x = -3:13;
% Decision boundary
% Change this if using +1 as bias
for i=1:totalClasses
    y = (-weights(i,2)*x+weights(i,1))/weights(i,3);
    plot(x,y);
end

plot(input(:,2),input(:,3),'ko')
hold off;
```