Practice with Perceptrons

Exercise 1

Here you will study the behaviour of the perceptron learning rule in two different classification problems. The file lab2.mat contains four matrices, as follows:

Matrix	Comment
p1	Input vectors for the first problem
t1	Target outputs (classes) for the first problem
p2	Input vectors for the second problem
t2	Target outputs (classes) for the second problem

Note that there is only one set of data (inputs + target output) for each problem because we are only going to train the perceptron. Of course, in a real world problem, you would want to use at least two data sets — one for training, and one for testing. If you look at t1 and t2, you will see that in each problem there are two classes.

Plot the training examples using different colours or symbols for each class, e.g.,

```
>> plot (p1(1,1:50), p1(2,1:50), 'b+', p1(1,51:100), p1(2,51:100), 'ro')
```

The function newp from neural network toolbox requires an argument which specifies the expected range for each of the inputs in the input vector. The function minmax can be used for this purpose, as follows:

```
>> mm = minmax(p1);
>> net = newp(mm,1);
```

The above steps will automatically work out the correct range for each input, and then create a new perceptron with one neuron (and therefore one output).

Now train the perceptron using the provided m-function trainp_sh as follows:

```
>> net = trainp_sh(net, p1, t1, 1);
```

The last argument to trainp_sh, the number of epochs (training steps) should be set to 1. This function will also update the graphical display, showing the decision line as well as the training examples.

For each of the two classification problems, plot the training examples and create a new perceptron, as described above. Then, do successive calls to trainp_sh and study the graph (resize the Matlab window so that you can watch the graph at the same time).

Once you get the general idea of what's happening, you can set the number of epochs to a higher number so that things go a little faster.

Consider the following question (for both data sets):

[1] Is the perceptron able to classify all training vectors correctly after training? If not, why?

Exercise 2.

Design a perceptron with one output to decide if a digit represented by binary pattern is even or odd.

The 10 digits from 0 to 9 with the corresponding binary pattern is shown as the vectors below:

digit	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}
0	1	1	1	1	0	1	1	0	1	1	0	1	1	1	1
1	1	1	0	0	1	0	0	1	1	0	1	0	1	1	1
2	1	1	1	1	0	1	0	1	1	1	1	0	1	1	1
3	1	1	1	1	1	1	0	1	1	0	0	1	1	1	1
4	0	1	0	1	1	0	1	1	0	1	1	1	0	1	0
5	1	1	1	1	1	0	0	1	1	0	0	1	1	1	1
6	0	1	1	1	1	0	1	1	1	1	0	1	1	1	1
7	1	1	1	1	0	1	0	1	0	0	1	0	0	1	0
8	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1
9	1	1	1	1	0	1	1	1	1	0	1	1	1	1	0

(1) Use the vectors given in above table as the training data, together with their corresponding targets (0 for odd and 1 for even), train a perceptron by applying the Matlab Neural Network Toolbox.

(2) Note the performance of training error within 10 epochs;

(3) After the training complete, test the following two patterns to see if the perceptron's outputs are correct.

```
ptest=[1 1 1 1 0 1 1 0 1 1 0 1 1 0 1;
       1 1 0 0 1 0 0 1 1 0 1 0 1 1 1]';
```