

[Studenti](#)[Ricerca](#)[Ateneo](#)[Servizi online](#)[Intranet](#)[Aulaweb](#)

[Dibris](#) ► [My courses](#) ► [Robotics Engineering \(Scuola Politecnica\)](#) ► [Anno Accademico 2013/14](#) ► [65862-1314](#) ► [Lab assignments](#) ► [Lab assignment 1: one-layer perceptrons](#)  
You are logged in as **Muhammad Farhan Ahmed** ([Logout](#))

## Navigation

[Dibris](#)[My home](#)[Dibris](#)[My profile](#)[Current course](#)[65862-1314](#)[Participants](#)[Badges](#)[General](#)[Activity dates and topics](#)[Notes and slides](#)[Lab assignments](#)[Lab submission guidelines](#)[Lab assignment 1: one-layer perceptrons](#)[Experiment 2: Adaline and non-linearly separable problems](#)[Feedback on](#)

## Lab assignment 1: one-layer perceptrons




- Task 1: Simple 2D perceptron for binary problems, manual design
- Task 2: Simple perceptron for Iris data, manual design
- Task 3: Perceptron array for simple image recognition and experiments in training
- Task 4 (optional): Improvements

Describe everything in a report. One figure is required (a graph), others can be added if useful.

### Task 1: Simple 2D perceptron for binary problems, manual design

Write suitable data files (plain-text files in a format suitable for reading by a program) containing the following two-dimensional data sets:

DATASET 1			DATASET 2		
point	class		point	class	
.12	.1	1	.27	.78	1
.1	.31	1	.1	.31	1
.6	.23	1	.6	.43	1
.4	.5	1	.4	.5	1
.61	.7	-1	.61	.7	-1
.84	.4	-1	.84	.4	-1
.2	.87	-1	.2	.87	-1

[course workload](#) [Lab assignment 3: Prototype-based classifiers](#) [Lab assignment 4: Training multi-layer networks](#) [Lab assignment 5: Unsupervised learning](#)[Lab resources](#)[General resources](#)[My courses](#)[Courses](#)

## Administration

[Course administration](#)[My profile settings](#)

(1) For each data set, plot it to view it. Manually find a solution, a separating hyperplane, then write a program that implements a perceptron with the weights corresponding to that solution, and run it on the data. Report on whether it works, and, if not, why.

(2) Test each perceptron with weights affected by random perturbations, and check whether it still works:

A random perturbation is a random vector with a given norm  $p$ , that is summed to the weight vector. You obtain it by creating a completely random vector, making it unit norm, then multiplying it by  $p$ .

Increase the norm from a minimum to a maximum, with a step size that is not too large. Guideline: maximum not larger than 10% the norm of  $\mathbf{w}$ , and 10 steps (so step size = maximum/10).

For each value of  $p$ , do some statistics: create a number of perturbations, apply them, run the perceptron on the data, and evaluate the number of errors. Guideline: use a fairly high number of random trials, e.g., 100.

For each data set, plot a graph (Excel, Matlab, hand... is fine) of the **average number of errors** vs **perturbation size**  $p$ . Do the results differ for different data sets? How? Is there a specific feature of these sets that can justify different behaviors?

### Task 2: Simple perceptron for Iris data, manual design

Download the [simplified Iris data set](#) (two classes, 2 and 3 merged). Design and implement a perceptron that solves the problem. Test it and describe. You may use the graphs on page 16 of the course notes as a guidance.

### Task 3: Perceptron array and experiments in training

Implement the perceptron learning rule in a **general-purpose program module** (e.g., a callable function). Provide:

- Multiple loops over the training set, until convergence is attained
- A regular stopping criterion that recognizes when convergence has been reached
- An emergency stopping criterion that avoids infinite looping in case of non-convergent data sets

Don't hardwire constant values: put them in variables, so they can be changed in one place (where they are assigned).

Also implement a **main program or script** that does the housekeeping tasks: loads the data, calls the functions, performs the tests, prints the results...

Tests:

(1) Apply the perceptron training algorithm to simple 2-dimensional datasets, for instance those from Task 1. Show the progress of learning by **recording the weights at each iteration** and then by **drawing the corresponding separating surfaces** (lines) to see how it proceeds.

Does modifying the learning rate change anything? What does it change?

(2) Download the [original Iris data set](#) (three classes). Build three training sets:

1. one with class 1 as 1 and 2 and 3 as -1;
2. one with class 2 as 1 and 1 and 3 as -1;
3. one with class 3 as 1 and 1 and 2 as -1.

Train three perceptrons by running the perceptron learning algorithm, each on one training set to recognize one class. Note that classes 2 and 3 are not linearly separable, so the emergency stopping criterion is required.

Perform training with different initial weights and note the average number of iterations before convergence (when there is convergence).

Test some of the trained perceptrons, using the training set itself as a test set, and describe the results. Especially of interest are the performances of non-convergent perceptrons, stopped forcefully.


#### **Task 4 (optional): Improvements**

Can you think of some ways to improve the perceptron learning rule? For instance, to return a suboptimal solution in non-linearly-separable cases; to return the most robust solution rather than the first found.

How would you proceed to overcome the limitation to linearly separable cases?

UPLOAD BELOW YOUR CODE, DATA, AND REPORT

#### **Submission status**

Submission status	Submitted for grading
Grading status	Not graded
Due date	Sunday, 20 October 2013, 11:55 PM
Time remaining	Assignment was submitted 1 min 59 secs late
Last modified	Sunday, 20 October 2013, 11:56 PM
File submissions	 <a href="#">Ahmed_lab@1.rar</a>