Further Practice with Perceptrons

Demo: classifying 3D patterns into 2 classes

The following Matlab program sets up a perceptron network with 3 input units and 1 output unit to perform a 2-way classification:

initPerceptron.m	(initialization)
plotPerceptronInput.m	(plot the input patterns)
plotDecisionSurf.m	(plot the classification boundary)
trainPerceptron.m	(train perceptron with implemented learning algorithm)
runPerceptron.m	(run the perceptron training and other related m-files)

1) The m-file script 'initPerceptron.m' initializes the weights and the training patterns;
2) The m-file script 'runPerceptron.m' calls the function 'trainPerceptron' and plots the input patterns and the classification boundary (hyperplane) after 1 learning epoch (use all the data once);
3) The patterns displayed as asterisks have a target output of +1; the patterns displayed as circles have a target output of -1.

Exercises:

1. Run the example

Read the code and understand how the weights and states are initialized and how the weights are updated. Then do the following:

1) Execute the script 'initPerceptron', which initializes the weights and training patterns;
2) Run the script 'runPerceptron' (it is suggested to adjust the positions of Matlab figure window and command windows so they are both visible);
3) Repeatedly run the script 'runPerceptron' until the network has converged (weight no longer changes). You can drag the figure to adjust the viewing angle. The classification boundary can be observed in the figure and the numerical values of the weights is printed out in command window after each 'runPerceptron' call. It may take between 5 and 20 learning epochs (run 'runPerceptron' for 5-20 times) before the learning converges.

Repeat this entire process for several trials: for each trial, re-initialize the perceptron ('initPerceptron') and re-train ('runPerceptron') until convergence. Think about the reason of different convergence time for each trial.

2. Increase the classification difficulty

You can change the variance of the random Gaussian function used to generate different training patterns. Specifically, change 'var' in 'initPerceptron' from 0.1 to 0.3. Re-run your Perceptron a few more times on these data.

1) Does the Perceptron converge more quickly or more slowly on these data?
2) After each time you re-initialize the perceptron and data set, does it always converge, or sometimes fail to converge?
3) How can you explain the convergence or lack of convergence by looking at the training patterns?