Neural Networks and Computational Intelligence 2017/18
Practical Assignment I: Rosenblatt Perceptron

The topic of this assignment is the Rosenblatt perceptron algorithm. We apply it to randomized data and try to observe our theoretical findings (capacity of a hyperplane) in computer experiments. The code (or large parts thereof) can be re-used in the next assignment(s).

---

1) Rosenblatt Perceptron Algorithm

For this systematic study of linear separability, write a program (Matlab recommended, but not obligatory) which can be used to

a) ... generate artificial data sets containing P randomly generated N-dimensional feature vectors and binary labels: $\mathbb{D} = \{\boldsymbol{\xi}^\mu, S^\mu\}_{\mu=1}^P$. Here, the $\boldsymbol{\xi}^\mu \in \mathbb{R}^N$ are vectors of independent random components ξ_j^μ with mean *zero* and variance *one*. You can use, for instance, the Matlab command `randn` to generate vectors of Gaussian components $\xi_j^\mu \sim \mathcal{N}(0,1)$. The labels in $\mathbb{D}$ should be independently drawn $S^\mu = \pm 1$ with equal probability 1/2.

b) ... implement sequential perceptron training by cyclic presentation of the P examples. At *time step* $t = 1, 2, \ldots$ present example $\mu(t) = 1, 2, \ldots, P, 1, 2, \ldots$.
This should be realized by using nested loops where the inner one runs from 1 to P and the outer loop counts the number n of *sweeps* or *epochs* through the data set $\mathbb{D}$. Limit the number of *sweeps* to $n \leq n_{max}$, i.e. the total number of single learning steps will be at most $n_{max}P$.

c) ... run the Rosenblatt algorithm for a given data set $\mathbb{D}$:

$$\boldsymbol{w}(t+1) = \begin{cases} \boldsymbol{w}(t) + \frac{1}{N}\,\boldsymbol{\xi}^{\mu(t)}\,S^{\mu(t)} & \text{if } E^{\mu(t)} \leq 0 \\ \boldsymbol{w}(t) & \text{else} \end{cases}$$

where $E^{\mu(t)} = \boldsymbol{w}(t) \cdot \boldsymbol{\xi}^{\mu(t)}\,S^{\mu(t)}$ and $\mu(t)$ is the index of the actual example. Initialize the weights as $\boldsymbol{w}(0) = 0$, but make sure that a training step is indeed performed for $E^{\mu(t)} = 0$.

The training should be performed until either a solution with <u>all</u> $E^\nu > 0$ is found (counted as a *success*) or until the maximum number of *sweeps* n_{max} is reached.

d) ... repeat the training for several randomized data sets, e.g. by running (a-c) within yet another loop. For a given value of P, use a number n_D of independently generated sets $\mathbb{D}$. Determine the fraction $Q_{l.s.}$ of successful runs as a function of $\alpha = P/N$, by repeating the experiment for different values of P.

Computer experiments and report:
The report should very briefly introduce the problem in an introductory section. Then, describe your solution in another short section. In general, do not provide your source code. We might ask you to do so, later, if necessary for the evaluation/grading. If you use particular *tricks*, you may present the corresponding commands or a few lines of (pseudo-) code in the report.
Run your code in order to study Perceptron training **at least** for the following parameter settings: $N = 20$, $P = \alpha N$ with $\alpha = 0.75, 1.0, 1.25, \ldots 3.0$, n_D=50, n_{max}=100.

As the key result, obtain $Q_{l.s.}$ as a function of α and display it as a graph. Discuss your result in words, compare it with the probability $P_{l.s.}(\alpha)$ that was derived in class, see lecture notes. If it differs, mention potential reasons in the discussion.

Remark:
Your actual choice of parameters will – of course – depend on your implementation and on available computing power. If (CPU-) time allows, improve the quality of your results by setting N, n_D and n_{max} as large as possible.

Potential bonus problems:

The following are only suggestions for the interested. Ideas of your own are, of course, also welcome.

• Observe the behavior of $Q_{l.s.}$ for different system sizes N as well. Does it approach a step function with increasing N? To this end, repeat the above experiments for several values of N with, for example, $\alpha = 1.75$ and $\alpha = 2.25$

• Consider a non-zero value of c as introduced in class for updates when $E^\mu < c$. Does the choice of c influence the result?

• Modify the algorithm in order to find also inhomogeneous perceptron solutions by adding a clamped input to all feature vectors. Does $Q_{l.s.}$ change significantly?

Note: Rules and regulations concerning the practicals, deadlines, bonus, and example reports of earlier assignments will be posted on NESTOR very soon.