Neural Networks (2019/20)
Practical Assignment I: Perceptron Training

The main 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.

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 $\boldsymbol{\xi}^\mu \in I\!R^N$ are vectors of independent random components ξ_j^μ with mean *zero* and variance *one*. You can use, for instance, Gaussian components $\xi_j^\mu \sim \mathcal{N}(0,1)$. The labels S^μ are taken to be independent random numbers $S^\mu = \pm 1$ with equal probability 1/2.

b) ... implement sequential perceptron training by cyclic representation 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 *epochs*, i.e. *sweeps* through the data set $\mathbb{D}$. Limit the number of sweeps to $n \leq n_{max}$ so that the total number of individual update steps will be at most $n_{max}P$.

c) ... run the Rosenblatt algorithm for a given data set $I\!D$:

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

   where $E^{\mu(t)} = \boldsymbol{w}(t) \cdot \boldsymbol{\xi}^{\mu(t)}\,S^{\mu(t)}$. 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. The result should resemble the probability $P_{l.s.}(\alpha)$ that was derived in class, see lecture slides and full text lecture notes.

Computer experiments and report:

The report should 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:

$$N = 20, \quad P = \alpha N \;\text{ with } \alpha = 0.75, 1.0, 1.25, \ldots 3.0, \quad n_D \sim 50, \quad n_{max} \sim 100.$$

As the key result, obtain $Q_{l.s.}$ as a function of α and display it as a graph in an appropriate fashion (caption text, axis labels, data points marked by symbols). Discuss your result in words, compare with the probability $P_{l.s.}(\alpha)$ that was derived in class. If your results differ, discuss potential reasons for the deviations.

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/or n_{max} as large as possible.

Possible extensions ('bonus'):
The following points are only a few example suggestions, ideas of your own are very welcome!

- Observe the behavior of $Q_{l.s.}(\alpha)$ for different system sizes N. Does it approach a step function with increasing N, as predicted by the theory? To this end, repeat the above experiments for several values of N. For this study, you might consider a limited range of α-valiues, e.g. $1.75 \leq \alpha \leq 2.25$. and perhaps consider a smaller increment of α.
- Determine the embedding strenghts x^μ (or formulate the algorithm in terms of the x^μ) to obtain a histogram which reflects their frequency in the linearly separable data cases upon convergence of the training.
- Consider a non-zero value of c as introduced in class for updates when $E^\mu < c$. Does the choice of c influence the results in terms of $Q_{l.s.}$?
- Modify the algorithm in order to find and count also inhomogeneous perceptron solutions by adding a clamped input to all feature vectors. Does the number of successful training processes change significantly?