

Neural Networks (2022/2023)

Practical Assignment I: Perceptron Training

The main topic of this assignment is the Rosenblatt perceptron algorithm. We apply it to randomized data and compare the results with our theoretical findings (capacity of a hyperplane) in computer experiments.

Rosenblatt Perceptron Algorithm

For this systematic study of linear separability, write a program 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, 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 repeated cyclic representation of the P examples. At *time step* $t = 1, 2, \dots$ present example $\mu(t) = 1, 2, \dots, P, 1, 2, \dots$

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 generated data set \mathbb{D} with updates of the form

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

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 is found such that $E^\nu > 0$ for all ν or 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:

Rules and recommendations concerning the content, style and submission of the report and/or the code will be announced in good time.

To study perceptron training, run your code **at least** for the following parameter settings:

$$N = 20 \text{ and } N = 40, \quad P = \alpha N \text{ with } \alpha = 0.75, 1.0, 1.25, \dots 3.0, \quad n_D = 50, \quad n_{max} = 100.$$

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 and weaknesses of the computer experiments.

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 much encouraged (please discuss them with the TA in advance).

- 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 larger values of N . For this study, you might want to consider a limited range of α -values, e.g. $1.5 \leq \alpha \leq 2.5$. and perhaps consider a smaller increment of α in this interval.
- 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 successful cases upon convergence of the training.
- Consider a non-zero value of c as introduced in class for updates when $E^\mu \leq c$. Does the choice of c influence the results in terms of $Q_{l.s.}$?
- Modify the algorithm by adding a clamped input to all feature vectors in order to find and count inhomogeneous perceptron solutions as well. Does the number of successful training processes change significantly?