

Neural Networks and Computational Intelligence

Practical Assignment II: Learning a lin. sep. rule

You should either hand in a report for assignment II **or** assignment III. If you hand in all three, report III will be completely disregarded and not graded.

Learning a linearly separable rule

The topic of this assignment is the learning of a linearly separable rule from example data. We define outputs $S^\mu = \pm 1$ which are provided by a *teacher perceptron*. By construction, the resulting data set is guaranteed to be linearly separable, and learning in version space is a reasonable strategy in the absence of noise in the data set.

Consider a set of random input vectors as in assignment (I) with similar dimensions N . However, here we consider training labels S^μ which are defined as

$$S^\mu = \text{sign}(\mathbf{w}^* \cdot \boldsymbol{\xi}^\mu)$$

by a *teacher perceptron*. You should consider a randomly drawn \mathbf{w}^* with $|\mathbf{w}^*|^2 = N$. Also, modify your code from assignment (I) so that it ...

- ... implements the sequential Minover algorithm:
at each time step t , determine the stabilities

$$\kappa^\nu(t) = \frac{\mathbf{w}(t) \cdot \boldsymbol{\xi}^\nu S^\nu}{|\mathbf{w}(t)|} \quad \text{for all examples } \nu$$

and identify the example $\mu(t)$ that has currently the minimal stability $\kappa^{\mu(t)} = \min_\nu \{\kappa^\nu(t)\}$. In case of a *tie*, it does not matter which of the examples is chosen.

With the selected example $\mu(t)$, perform a Hebbian update step

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \frac{1}{N} \boldsymbol{\xi}^{\mu(t)} S^{\mu(t)}$$

and go to the next time step. In contrast to the Rosenblatt algorithm, the sequence of examples is not fixed and in each step a non-zero update is performed. Note that Minover should not stop when $\{E^\nu > 0\}_{\nu=1}^P$, because the stability will increase further. Run the algorithm until a number of $t_{max} = n_{max} \cdot P$ single training steps have been performed in total. The final weight vector $\mathbf{w}(t_{max})$ with stability $\kappa(t_{max})$ for a given set of data should approximate the perceptron of optimal stability \mathbf{w}_{max} . Initialize the weight vector as $\mathbf{w}(t) = 0$ in each training process and implement a (reasonable) stopping criterion of your choice.

Please include the main piece of code in the report, i.e. the actual realization of the Minover learning step. Complete code should be submitted in a separate e-mail to the announced address.

- ... determine the generalization error (at the end of the training process) according to

$$\epsilon_g(t_{max}) = \frac{1}{\pi} \arccos \left(\frac{\mathbf{w} \cdot \mathbf{w}^*}{|\mathbf{w}(t_{max})| |\mathbf{w}^*|} \right).$$

By repeating the training for different P , determine the so-called *learning curve*, i.e. the final generalization error $\epsilon_g(t_{max})$ as a function of the scaled number of examples $\alpha = P/N$. Obtain the result on average over $n_D \geq 10$ randomized data sets per value of P .

Consider a somewhat larger range of α than in assignment (I). The system size N and the range and number of different values of α depends, of course, on your patience, available CPU power, and the efficiency of your implementation. Provide your results for at least the 8 values $\alpha = 0.25, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0$. If time allows, include more intermediate values and/or extend the experiments to even larger α .

Hints:

- (1) It is important that t_{max} is large enough for the algorithm to converge or at least get close to optimal stability. Run exploratory experiments first to determine a reasonable value of t_{max} to make sure that a good estimate of κ_{max} is obtained.
- (2) The division by $|\mathbf{w}|$ is an important part of the definition of κ^μ . However, if you compare different κ^ν for the same given weight vector, i.e. when identifying the minimum, you can of course drop it. In other words: for a given \mathbf{w} , the minimum of the E^ν identifies the relevant example.

Suggested extensions (bonus problems):

- Define and implement a reasonable stopping criterion that ends training when the perceptron system *does not change anymore* over a (large enough) number of steps. Note that a suitable criterion is less trivial to formulate than it seems at first sight.
- Determine the individual stabilities $\kappa^\nu(t_{max})$ at the end of training and generate histograms of the frequency of observed stabilities.
- Repeat the above experiments for the simple Rosenblatt Perceptron, which stops as soon as the classes are separated. Compare the learning curves $\epsilon_g(\alpha)$. Can you confirm that maximum stability yields better generalization behavior?
- Implement the AdaTron algorithm (in terms of embedding strengths) and compare the convergence behavior / speed with the Minover algorithm.
- Consider learning from noisy examples by replacing the true labels in the data set by

$$S^\mu = \begin{cases} +\text{sign}(\mathbf{w}^* \cdot \boldsymbol{\xi}^\mu) & \text{with probability } 1 - \lambda \\ -\text{sign}(\mathbf{w}^* \cdot \boldsymbol{\xi}^\mu) & \text{with probability } \lambda \end{cases}$$

Here $0 < \lambda < 0.5$ controls the noise level in the training data.

- For non-separable data (e.g. with noisy labels as suggested above) implement the "large margin with error" version of the AdaTron algorithm and perform a similar set of experiments.

• **Iris Flower data**

Determine the perceptron of maximum stability for a suitable sub-set of the Iris Flower data set. For details, see the separate document provided under "Iris data set" in the folder "Assignments".

Methods for non-separable data could be explored for the non-separable subset of Iris data, e.g. by means of the *pocket algorithm* or the *AdaTron with errors*.