The generalization probability of a perceptron using the Delta -rule

1992 J. Phys. A: Math. Gen. 25 L505

(http://iopscience.iop.org/0305-4470/25/8/021)

J. Phys. A: Math. Gen. 25 (1992) L505-L510. Printed in the UK

LETTER TO THE EDITOR

The generalization probability of a perceptron using the Δ-rule

Stephan Seewer† and Pál Ruján‡

† Institute of Theoretical Physics, University of Lausanne, CH-1015 Lausanne, Switzerland
‡ Universität Oldenburg, Postfach 2503, D-2900 Oldenburg, Federal Republic of Germany

Received 28 August 1991, in final form 5 January 1992

Abstract. We compare the generalization probability of a perceptron using the Δ-rule with other learning algorithms. Our numerical results indicate that the Δ-rule gives results which are superior to Hebb's rule, and comparable with the maximal stability perceptron rule (minover), both for randomly generated linear separable example sets and for optimal queries.

The large majority of the artificial neural networks developed recently use the back-propagation algorithm [1] or some of its variants. This class of learning algorithms is based on gradient descent search. As a result, many neural network researchers strongly argue for 'analogue' computing, meaning roughly working with continuous variables and differential equations.

Learning a linearly separable function with continuous input variables is a convex programming problem. When the input variables are discrete, the solution is no longer unique and different learning procedures may lead typically to different solutions. This situation presents thus a simple testing ground for 'analogue' versus 'digital' computing, albeit one where the gradient method is an advantage. While for multilayer networks it seems difficult to assess the extent to which the choice of a particular learning algorithm contributes to the performance of the network, model calculations can be done in a single layer perceptron [2]. In this letter, we present a numerical evaluation of the generalization probability of the Δ-rule on the class of linear separable functions, and compare it with other methods. Our results indicate that the Δ-rule can be tuned to close-to-optimal performance in both learning from examples and learning from queries situations, shadowed somewhat by the long convergence time.

First, let us consider the supervised learning scheme (learning from examples). We require a neural network to learn a Boolean function $f \in \mathcal{F}$, defined on N binary inputs $f(s_1, \ldots, s_N) = \pm 1$ where $s_i = \pm 1$, and $\mathcal{F}$ denotes the set of linearly separable functions $\mathcal{F} = LS$. The input space configuration is defined as the corners of an N-dimensional hypercube with coordinates $(s_1, \ldots, s_N)$. Only some of the corners are used as examples during the learning phase. For a given function f these are coloured black if the output is $+1$ and white if it is -1. We say that a function is linearly separable if it is possible to separate white and black corners with a hyperplane. A linearly separable function can be written in the form

$$f_w(s) = f(\{w, \delta\})(s) = \operatorname{sign}[w \cdot s - \delta]$$

where s is the input vector, w is the weight vector, and δ is the threshold. The equation of the hyperplane is given by $w \cdot s - \delta = 0$ where w is a vector perpendicular to the hyperplane.

0305-4470/92/080505+06$04.50

The calculation of the generalization probability has been formalized [3] as follows. We choose an arbitrary function $f_T(s)=f(\{T,\theta\})(s)\in LS$ called the *target* or *teacher function*. From some fixed distribution $P(s)$, we generate a set of M examples $\xi^{(\nu)}$ $(\nu=1,2\ldots,M)$, whose desired output corresponds to the target function output $\sigma^{(\nu)}=f_T(s^{(\nu)})$. During the learning phase, the algorithm determines a hyperplane with equation $J\cdot s-\Delta=0$ and thus an approximation for the target function $f_J(s)=f(\{J,\Delta\})(s)$ (*student function*). After M examples have been learnt, we wish to know the probability of further examples generated from $P(s)$ being correctly classified. This is the generalization probability, and is defined in the limit $N\to\infty$ as

$$G(\alpha)=1-P_E(\alpha)\qquad P_E(\alpha)=\sum_{\{s:f_J(s)\neq f_T(s)\}}P(s)\qquad \alpha=\frac{M}{N}.$$

To simplify, we assume that the thresholds θ and Δ are zero and that P is a uniform distribution. The generalization ability now depends only on the angle β between the 'teacher' T, and the actual 'student' approximation J [4]:

$$G(\alpha)=1-\frac{\beta}{\pi}=1-\frac{1}{\pi}\cos^{-1}\left(\frac{T\cdot J}{|T||J|}\right)$$

where

$$|T|^2=\sum_{i=1}^{N}T_i^2.$$

In the usual setup for the Δ-rule, we shall allow the output from neuron $\mathcal{O}$ to be a continuous variable $\mathcal{O}\in(-1,1)$ and use for updating the sigmoid function

$$\mathcal{O}^{\nu}=\tanh[mw\cdot s^{\nu}]. \tag{1}$$

$\mathcal{O}^{\nu}$ can be given a probabilistic interpretation [5] for real neurons. The Δ learning rule now consists of minimizing the energy function [1]

$$E=\sum_{\nu=1}^{M}(\sigma^{(\nu)}-\mathcal{O}^{(\nu)})^2. \tag{2}$$

provided $w\cdot w=$ constant, and m is fixed real parameter.

Consider the very simple situation illustrated in figure 1. Here, we have a linear separable problem whose solution is a hyperplane. Of the two solutions A and B,

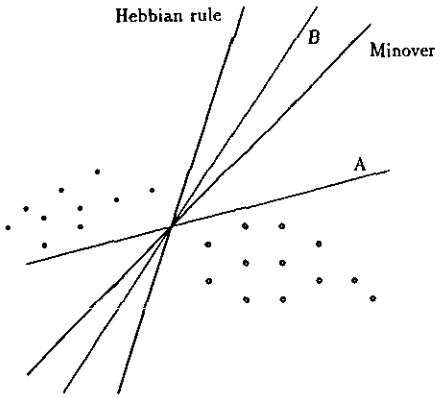

Figure 1. Comparison of different solutions for a linearly separable function.

solution B is better because it is more robust under rotation and translation of the separating plane, or conversely, in the face of uncertainty in the location of the example points [6]. One can tune m so that if m is small, B is the better solution, since the energy is smaller; on the other hand, if m is large, then the two solutions are equally valid because the energy is practically the same for both.

We seek thus to vary m in order to improve the 'quality' of the solution. In other words, if m is small, we hope that the student function will resemble the teacher function as closely as possible.

To ensure that the norm of w is constant, we use polar coordinates, $\theta_0, \ldots, \theta_{N-1}$ for w, and minimize with respect to these variables using the 'conjugate gradient' method [7]. This reduces convergence time by a factor of about 6 compared with the 'gradient descent' method, and of about 3 compared with the gradient descent with momentum term method [1]. For the initial condition, we choose random values of $\theta_0, \ldots, \theta_{N-1}$ within the interval $[0, \pi]$.

The simulations we made were of two kinds: in one case, the examples were chosen at random (learning by examples); in the second, we allowed the network itself to select the training examples (learning by queries): it chose those nearest to the hyperplane [4]. For convenience, we took $|w| = 5$, and 50 neurons throughout. The results for the two cases were compared with the 'minover' algorithm of Krauth and Mézard [8] and with the Hebbian rule

$$J_i' = J_i + \frac{1}{N} \sum_{\nu} \sigma^{\nu} \xi_i^{\nu}. \tag{3}$$

Here ξ_i^{ν} is the ith component of the example (or query) $\boldsymbol{\xi}^{\nu}$ and σ^{ν} the teacher's anser to the input $\boldsymbol{\xi}^{\nu}$.

In the case of learning by example, the differences in generalization probability were observed to depend to some extent inversely on m (see figure 2). In the solutions for $m = 0.5$, 1 and 1.5 we see that the smaller the value of m, the better the generalization probability, regardless of the value of α. However, when $m = 0.2$, this is only true when

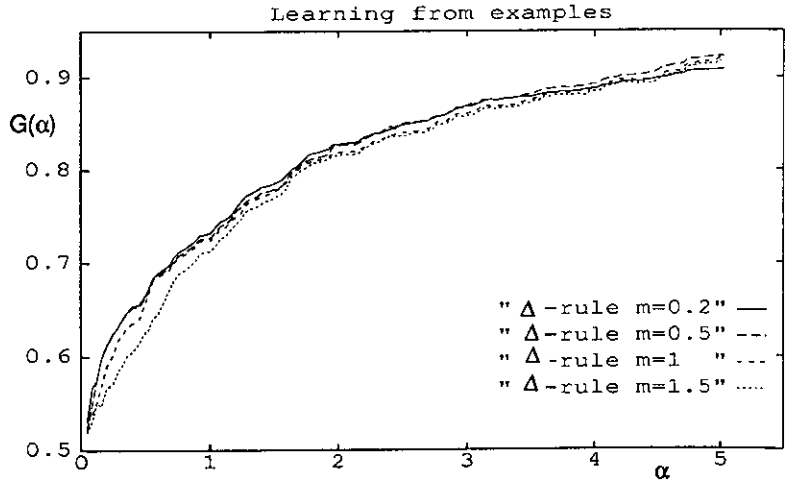

Figure 2. Comparison between generalization probabilities $G(\alpha)$ for the Δ-rule from a set of *Randomly generated examples* with $m = 0.2$, 0.5, 1, 1.5. The points indicate simulation results obtained after averaging over 15 networks with $N = 50$.

$\alpha < 2$. An explanation of this may be that when the slope is small, any local minimum or minimum plateau in the energy function would prevent the gradient descent algorithm from finding the true minimum. Thus, when m was in the approximate interval 0.5 to 1.5, the neuron network failed to learn 0.01% of the examples presented. For smaller m, the percentage of failure increased: for $m = 0.2$ there was 2.2% failure. As m increased beyond 1.5, the energy function became more and more step-like and convergence was not achieved.

When we compare the Δ-rule to the Hebbian rule and the minover algorithm, we find that the delta-rule is better than the Hebbian rule and comparable with the minover algorithm (figure 3). The difference between the generalization probabilities of the 'Hebbian rule' and minover algorithm can be understood by studying the learning methods of each algorithm. The most simple algorithm is the 'Hebbian rule' which learns in one step, giving

$$\boldsymbol{\omega} = \frac{1}{N}\sum \sigma^{\nu}\boldsymbol{\xi}^{\nu} = \frac{1}{N}\left(\sum_{\nu' \in \text{black points}} \boldsymbol{\xi}^{\nu'} - \sum_{\nu'' \in \text{white points}} \boldsymbol{\xi}^{\nu''}\right).$$

One can observe that the direction of $\boldsymbol{\omega}$ is from the centre-of-mass of the black points to the centre of the white. This solution is not robust under translations of the vector $\boldsymbol{\omega}$ and gives poor generalization probabilities (see figure 1) [6].

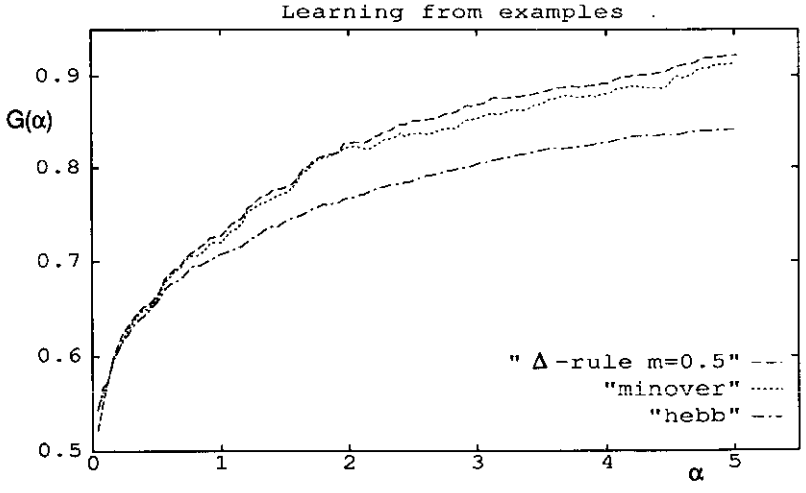

Figure 3. Comparison between generalization probabilities $G(\alpha)$ for the Δ-rule ($m = 0.5$), the Hebbian rule and the minover algorithm, from a set of *randomly generated examples*. The points indicate simulation reuslts obtained after averaging over 15 networks with $N = 50$.

On the other hand, the minover algorithm determines the 'minimal connector' which is the minimal distance between the white and black convex polytopes, where the white (index 0) and black (index 1) polytopes are defined as follows:

$$S_0 = \left\{ x_0 \in S_0 \,|\, x = \sum_{\nu=1}^{M_0} a_\nu \boldsymbol{\xi}^{\nu};\ \sum_{\nu} a_\nu = 1,\ a_\nu \geqslant 0 \right\}$$

$$S_1 = \left\{ x_1 \in S_1 \,|\, x = \sum_{\nu=1}^{M_1} b_\nu \boldsymbol{\xi}^{\nu};\ \sum_{\nu} b_\nu = 1,\ b_\nu \geqslant 0 \right\}.$$

With this notation, the minover algorithm gives ω as follows:

$$\omega \cdot \omega = \min_{a_\nu, b_\nu} L^2 \qquad \text{where } L^2 = (x_1 - x_0)^2$$

and thus ω is only sensitive to the closest black and white points. The minover algorithm thus constructs the perceptron with maximal stability and has the best generalization probability of all known single network linear discriminants [2].

The Δ-rule used in this letter is sensitive to those points whose distances from the hyperplane fall into the 'steep part' of the tangent hyperbolic function; the width of this region is controlled by m. In this way, it forms a compromise between the Hebbion rule (all the points) and the minover algorithm (only the nearest points). The good generalization probabilities show that this can be useful. With the Hebbian rule, the neuron network failed to learn 9% of the examples presented: with the minover algorithm, it learnt all the examples presented, as it should. The calculations were done on a Vax 9000: the average CPU times were 3 h 41 m 32 s for the Δ-rule, 13 m 50 s for the minover algorithm, and 17 s for the Hebbian rule.

In the case of learning by queries, the differences in generalization probability did not show a hierarchy of quality dependent on m, although the generalization probability was even better than in learning by examples (see figure 4). However, the examples presented were less well learnt (see table 1). The comparison with the Hebbian rule and minover algorithm yields similar results as before (figure 5): with the Hebbian rule, 13% of the examples presented were not learnt, while, as before, all the examples presented were learnt with the minover algorithm. The calculations were done on a

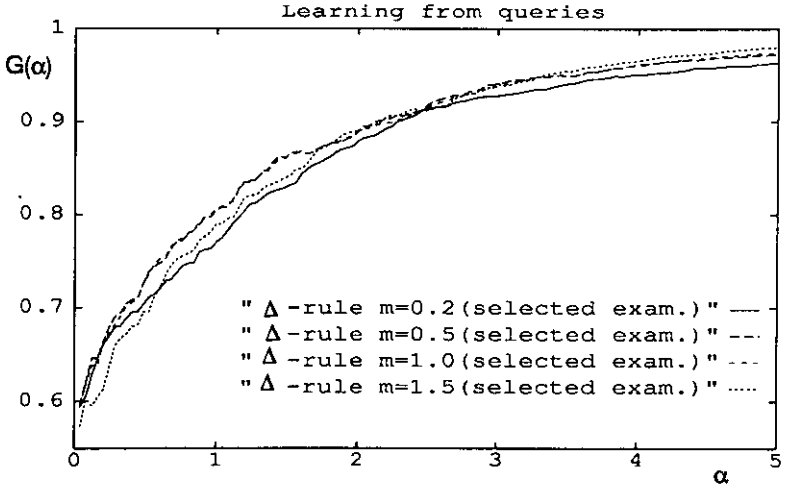

Figure 4. Comparison between generalization probabilities $G(\alpha)$ for the Δ-rule from a set of *selected examples* with $m = 0.2, 0.5, 1, 1.5$. The points indicate simulation results obtained after averaging over 15 networks with $N = 50$.

Table 1. Learning by queries, comparison of success in learning the examples presented for various values of m.

m	0.2	0.5	1	1.5
% failure	6.1	4.9	4.7	1.6

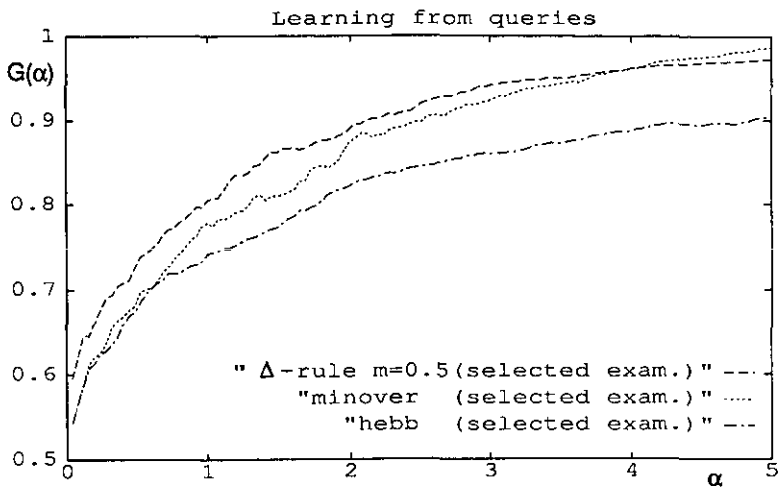

Figure 5. Comparison between generalization probabilities $G(\alpha)$ for the Δ-rule ($m = 0.5$), the Hebbian rule and the minover algorithm, from a set of *selected examples*. The points indicate simulation results obtained after averaging over 15 networks with $N = 50$.

Vax 9000: the average CPU times were 9 h 3 m 53 s for the Δ-rule, 16 m 32 s for the minover algorithm, and 1 m 32 s for the Hebbian rule.

In conclusion, we have shown that by a judicious choice of the slope for the sigmoid transfer function, the Δ-rule performs closely to the perceptron with maximal stability both in learning from examples and learning from queries protocols. However, in a typical situation, the longer time needed to achieve convergence and to optimize the slope might offset the gains in generalization ability. We do not see any computational reason favouring this 'analogue' approach over more 'orthodox' optimization approaches: they seem to us more like different alternatives for searching the space of solutions.

We thank P Erdös, F Pázmándi, D Rickebusch and G Schibler for useful discussions. The support of the Swiss National Science Foundation through grant no 20.28846-90 is gratefully acknowledged.

References

[1] Rumelhart D E, McClelland J L and the PDP research group *Parallel Distributed Processing* (Cambridge, MA: MIT Press)
[2] Kinzel W and Ruján P 1990 *Europhys. Lett.* **13** 473
[3] Valiant L G 1984 *Commun. ACM* **27** 1134
[4] Opper M, Kinzel W, Kleinz J and Nehl R 1990 *J. Phys. A: Math. Gen.* **23** L581
[5] Burnod Y and Korn H 1989 *Proc. Natl Acad. Sci. USA* **86** 352-6
[6] Ruján P 1991 *Statistical Mechanics of Neural Networks* ed L Garrido (*Lecture Notes in Physics* **368**) (Berlin: Springer)
[7] Press W H, Flannery B P, Teukolsky S A and Veetterling W T 1989 *Numerical Recipes in C* (Cambridge: Cambridge University Press)
[8] Krauth W and Mézard M 1987 *J. Phys. A: Math. Gen.* **20** L745