J. Phys. A: Math. Gen. 26 (1993) 4237–4247. Printed in the UK

Some notes on perceptron learning

Marco Budinich

Dipartimento di Fisica dell'Università di Trieste and INFN, Via Valerio 2, I-34127 Trieste, Italy

Received 21 September 1992; in final form 23 March 1993

Abstract. We extend the geometrical approach to the Perceptron and show that, given n examples, learning is of maximal difficulty when the number of inputs d is such that $n=5d$. We then present a new Perceptron algorithm that takes advantage of the peculiarities of the cost function. In our tests it is more than two times faster that the standard algorithm. More importantly it does not have fixed parameters, like the usual learning constant η, but it adapts them to the cost function. We show that there exist an optimal choice for β, the steepness of the transfer function. We present also a brief systematic study of the parameters η and β of the standard Perceptron algorithm.

1. Introduction

Perceptrons have long been the subject of intense study [1] and yet many of their features are not understood. Many fields are investigated nowadays, for instance learning and generalization [2, 3], the constrained weights case [4] and that of the Perceptron with a small percentage of errors [5].

In this paper we study some aspects of the Perceptron using a geometrical approach that was introduced long ago by Cover [6], and that is still quite useful [2, 4, 5]. In [7] we showed that the weight space is the conjugate space of the familiar input space. We proceed along this line to gain some insights into the properties of the cost function; this function is defined in weight space and summarizes the Perceptron performances.

We address two aspects: the theoretical capabilities versus the size of the problem and a modified Perceptron algorithm suited to meet the properties of the cost function.

In the first part we use $N_{\min}$, the average number of relative minima of the cost function, as a tool to investigate Perceptron performances. $N_{\min}$ reproduces well some of the known properties and shows that the learning is of maximal difficulty when, given n examples, the number of inputs d is such that $n=5d$. In this case $N_{\min}$ is maximal, indicating that the cost function is, on average, a real maze of relative minima. Consequently learning, that corresponds to finding its absolute minimum, becomes very difficult.

In the second part we propose a modified Perceptron algorithm with two features: it greatly reduces the number of movements in weight space necessary to find the solution and has a well defined halting condition when the solution doesn't exist (a badly missing property in the standard procedure). We show also that the steepness of

0305-4470/93/174237+11 \$07.50

the transfer function is parameter with a crucial role in learning and we show how to find the value that optimizes learning speed.

To make comparisons with the best possible version of the standard algorithm we carried out a systematic study of its performances varying the parameters. The results are in the appendix.

The first part of this paper is introductory: we review the formalism of Perceptron learning and subsequently we resume the results of [7]. In the second part of section 3 we study the characteristics of the cost function in weight space and in section 4 we present our new Perceptron algorithm together with its performances.

2. Standard Perceptron learning

We consider a Perceptron network with *d inputs i* and one *output o*. The output is a function of the inputs, of the *weights w* and of the threshold θ:

$$o=f\left(\beta\left(\sum_{k=1}^{d} w_k i_k-\theta\right)\right) \tag{1}$$

for the transfer function we choose $f(x)=1/(1+e^{-x})$. β is a scale factor that controls its steepness (it is the value of its derivative at $x=0$). When $\beta\to\infty$ $f(x)$ becomes the step function Θ.

For each set p of input values $i_k^p(k=1,\ldots,d)$ the net produces an output o^p. The *examples* (or learning set) are a set of n input patterns $i_k^p(k=1,\ldots,d,\ p=1,\ldots,n)$ each with its associated desired digital output ξ^p.

The cost function E is defined as the total number of errors (deviations between the actual and the correct output):

$$E=\frac{1}{2}\sum_{p=1}^{n}(\xi^p-o^p)^2. \tag{2}$$

The examples are learned when the weights are such that the net gives the desired output ξ^p for all the n input patterns; in this case the value of the cost function is zero.

Learning is achieved by finding those weights that minimize the cost function. The Perceptron theorem [1] guarantees that, if a solution exists, it can be found by changing iteratively each weight w_k by an amount $\Delta w_k=-\eta(\partial E/\partial w_k)$ where η is a learning factor. One easily obtains

$$\Delta w_k=\eta\sum_{p=1}^{n}(\xi^p-o^p)\frac{\partial o^p}{\partial w_k}=\eta\beta\sum_{p=1}^{n}(\xi^p-o^p)f'(x^p)i_k^p \tag{3}$$

where

$$x^p=\sum_{k=1}^{d} w_k i_k^p-\theta.$$

The constant η fixes the length of the vector $\Delta W=(\Delta w_1,\ldots,\Delta w_k,\ldots,\Delta w_d)$ and consequently the magnitude of the move in weight space. A very similar formula holds for the variation $\Delta\theta$ of the threshold θ.

Clearly this procedure is formally equivalent to descending along the gradient of the cost function. Thus the Perceptron theorem can be rephrased saying that, if a solution exists, the cost function has only one minimum that can be found by gradient descent.

Already here one can make one often ignored remark. Let us suppose that the input values for the examples are binary. A common argument is that one can choose freely the form 0 and 1 or ±1 for the input values since they are in one-to-one correspondence and each set of weights for one choice can be changed with simple algebra in a set of weights for the other one. Formula (3) shows that this is false *for learning* since if one makes the 0 and 1 choice, an example with 0 input will never contribute to Δw and can be difficult or even impossible to learn. Thus the ±1 choice is mandatory for the input of binary examples.

3. Geometrical interpretation

In this section we try to deepen our understanding of Perceptron learning by means of a geometrical interpretation. In the first part we resume the results of [7] to which the reader is addressed for a more detailed exposition.

The input patterns of a Perceptron with d input neurons can be thought as points in d-dimensional space and in this frame the action of the output neuron corresponds to separating these points with an oriented hyperplane. The output is 1 if the input configuration is in the 'positive' half-space and 0 otherwise (the positive definition is obviously arbitrary). The equation of the hyperplane is obtained from the argument of the function f in (1).

We extend this interpretation by introducing the familiar algebraic notion of conjugate space: the conjugate of the input space is the weight space where the cost function E is defined[1]. For each pattern in the input space there is a corresponding hyperplane in the weight space. Given n examples in the input space there will be a corresponding arrangement of n hyperplanes in the conjugate d-dimensional weight space. They partition it in $R(n, d)$ regions

$$R(n,d)=\sum_{k=0}^{d}\binom{n}{k}.$$

In each of these regions the cost function is almost constant (it is strictly constant if the transfer function $f(x)$ is a step function otherwise it is constant with smooth edges at the borders).

From this formula one derives the number of possible partitions of the input patterns $(2R(m-1,d))$ induced by the Perceptron for patterns in general position in d-space (no random 'alignment' between them). In the common case of digital patterns this is not generally true and $R(n,d)$ becomes an upper limit. Nevertheless it has been shown [8] that for large d and $n<O((d+1)^{3/2})$ the n digital patterns can be treated as if they were in a general position.

Figure 1 shows a cost function for a 2-d Perceptron in weight space. In this case the input space is a plane and the Perceptron function is a line in this space (not drawn).

[1] The conjugate space of a d-dimensional space is also a d-dimensional space and there is a one-to-one correspondence between points and hyperplanes between the two spaces. For example the conjugate space of the plane XY is a plane MO such that to each line of equation $Y=mX+q$ in XY corresponds a point (m,q) in MO and for each line in MQ there is a corresponding point in XY.

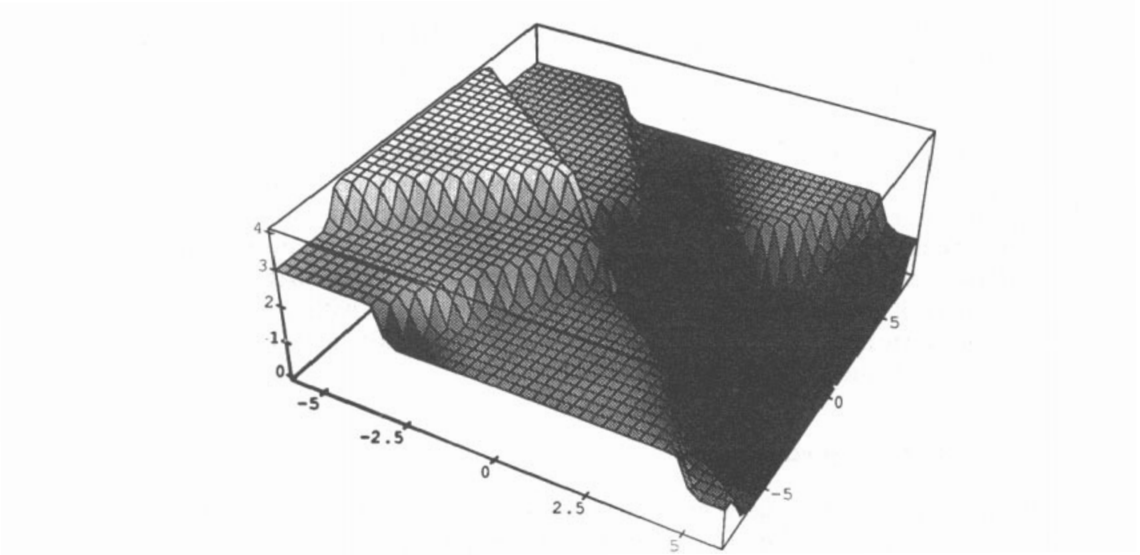


Figure 1. The cost function of a soluble Perceptron problem with $d=2$ and $n=4$.

The weight space is another plane whose coordinates are the coefficients of the line in input space. The four lines conjugate to the four input patterns partition the two-dimensional weight space in $R(4, 2)=11$ regions. In each of these 11 regions the cost function is almost constant. The region at the lower right is that in which $E=0$ corresponding to a choice of the weights that yields the exact solution.

If the given set of examples admits a solution the shape of the cost function is smooth with only one absolute minimum that can be found starting from any point in weight space using gradient descent (like in Figure 1). If for the given problem a solution doesn't exist the cost function becomes a maze of relative minima that ultimately trap any gradient descent algorithm.

In the hypothesis of random examples we can calculate the average number of relative minima. In the conjugate space there are $R(n, d)$ regions in which the cost function is practically constant. All together these $R(n, d)$ regions have $2nR(n-1, d-1)$ boundaries so, on average, each region has $S(n, d)$ 'sides'.

$$S(n, d)=\frac{2nR(n-1, d-1)}{R(n, d)}$$

On each border the cost function changes its value by 1 (either + or -1). Now let us consider a particular region and all the steps of the cost function on its borders. In the hypothesis of random examples there is no correlation between them. So the probability that a region, closed by S borders, is a relative minimum is the probability that on each of its borders the cost function has a $+1$ step. This probability is $1/2^{S(n, d)}$ and the average number of relative minima is:

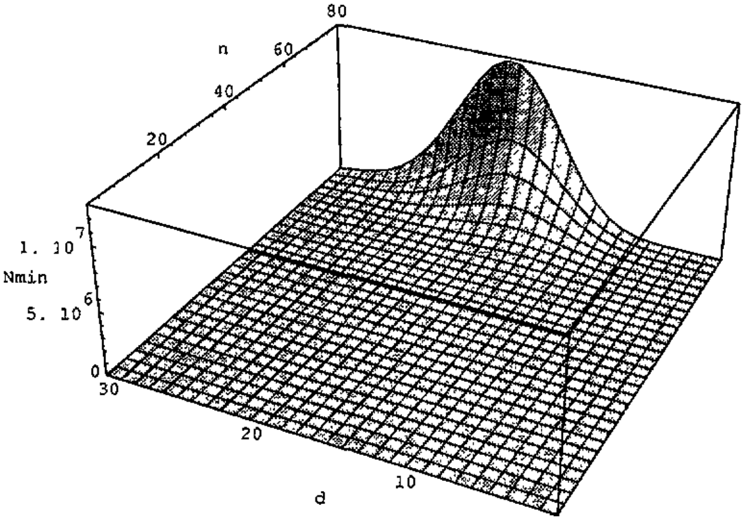
$$N_{\min}(n, d)=\frac{R(n, d)}{2^{S(n, d)}}. \tag{4}$$

Figure 2 contains plots in logarithmic and linear scale of $N_{\min}(n, d)$, the average number of relative minima of the cost function E. The independent variables are the number of inputs d and that of examples n. We can make several interesting observations:

n dependence. At fixed d and for large n, $N_{\min}(n, d)$ increases polynomially with n. Since for $n > 2d$ $R(n, d) = O(n^d)$ and $S(n, d) = O(2d)$ one easily gets

$$N_{\min}(n, d) = O\left(\left(\frac{n}{4}\right)^d\right) \qquad \text{for } n > 2d$$

that accounts for the observed trend.



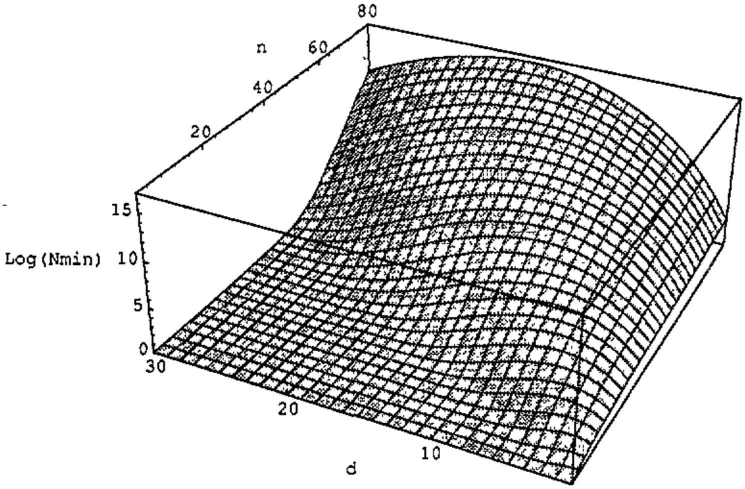


Figure 2. $N_{\min}(n, d)$ as a function of n and d in linear and logarithmic scale.

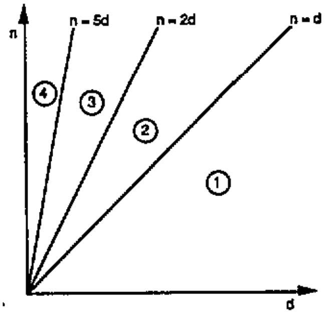


Figure 3. Different working regions for the Perceptron in the $n-d$ plane.

d dependence. At fixed n, $N_{\mathrm{min}}(n, d)$ has a pronounced maximum with varying d. With the same approximations just made we can write

$$N_{\mathrm{min}}(n, d) \approx \frac{\binom{n}{d}}{2^{2d}} \qquad \text{for } n > 2d$$

and we can study the maximum of this expression. We approximate the binomial coefficient using Stirling's formula for the factorial. Then we take the logarithm, we differentiate with respect to d and we get the following equation for the maximum

$$\frac{2d-n}{2d(n-d)} + \ln\left(\frac{n-d}{4d}\right) = 0$$

for $0<\mathrm{d}<n/2$ the first term adds a negligible contribution to that with the logarithm that goes to zero for $d=n/5$. This value for the position of the maximum is in excellent agreement with that observed numerically even for small d. From (4) we get that for $d=1$ $N_{\mathrm{min}}(n, 1) \approx (n+1)/4$ (for large n).

$n \leqslant d$ case. For $n \leqslant d$, $R(n, d) = 2^n$ and consequently $N_{\mathrm{min}}(n, d) = 1$. This fits nicely with the known fact that for $n \leqslant d$ the Perceptron always has an exact solution. (Apart from pathological cases of examples in degenerate position in d-space). Since a solution exists it follows that the Perceptron theorem holds and we saw that in this geometrical view the theorem says that the cost function has only one absolute minimum. The ability of $N_{\mathrm{min}}(n, d)$ to reproduce this known result strongly supports it as a good tool to understand Perceptron performances.

We can condense these results in a figure that accounts for all the possible combinations of $n-d$ values for the Perceptron. We partition the $n-d$ plane by means of three lines of equation $n=d$, $n=2d$ and $n=5d$ (Figure 3).

Region $1-n \leqslant d$. Here a Perceptron solution always exists; $N_{\mathrm{min}}(n, d) = 1$ and the average number of sides of a region $S(n, d) = n$.

Region $2-d<n \leqslant 2d$. Here the probability of finding a solution decreases from 1 to $\frac{1}{2}$ at $n=2d$ as proved by Cover [6]; in this region $S(n, d) = \mathrm{O}(n)$.

Region 3 $-2d<n \leqslant 5d$. Here the probability of finding a solution drops to zero [6]. At $n=5d$ $N_{\mathrm{min}}(n, d)$ reaches its maximum value with respect to d; in this region $S(n, d) \approx 2d$.

Region 4 – $n > 5d$. $N_{\min}(n, d)$ decreases with d to reach $(n+1)/4$ at $d=1$; in this region $S(n, d) \approx 2d$.

4. A new Perceptron algorithm

Now we use the geometrical interpretation of section 3 to propose a new learning algorithm for the Perceptron.

We have already stated that the standard Perceptron algorithm is strictly equivalent to a gradient descent on the cost function surface. Let us follow how the algorithm works looking at Figure 1. It begins selecting a random starting point in weight space and then it repeatedly moves along the gradient direction with step η. The Perceptron theorem guarantees that the solution, if it exists, will be ultimately found. If the solution does not exist there is no clear stop condition.

The limitations of this algorithm are:

- the uncertainty on the value of η (it is customary to keep it small);
- the lack of a halting condition. At the start one doesn't know how many moves in weight space will be necessary to find a solution or to abandon.

We propose here a variation of this algorithm that, taking advantage of the properties of the cost function, finds a remedy to these difficulties. There are three 'ideas' in it: the first is to abandon the fixed step size m, the second is to check and count the 'steps' of the cost function while moving, and the third is to use a variable β. We expose them one at a time.

Movements in weight space: looking at the cost function in Figure 1 we see that there are long plateaus with relatively sharp steps. It is intuitive that an adjustable step length can be very useful while moving in this space. An even more radical approach is to calculate the gradient and then search in that direction until a step is found. This search takes only a logarithmic number of trials. It turns out that in 96% of the cases it is sufficient to calculate the gradient only once per region to find the next descending step.

The steps of the cost function lie on the conjugate hyperplanes of the input examples and on each of them the cost function changes its value by $+1$ or -1. At the random initial point in weight space the cost function has a value in the closed interval $[0, n]$ say m (We assume m to be integer that is quite reasonable if the cost function is calculated far from the steps). Following a path along the negative gradient direction we will cross some of the steps. There are two possibilities: if our problem admits a solution we will cross m descending steps to arrive at the solution; if there is no solution the gradient descent will be trapped in a relative minimum higher than 0 after having crossed less than m steps. If we check the steps while moving we can detect the attempt to cross a rising step; this means that we are arrived in a minimum and that we can halt the algorithm.

The crucial role of β, the steepness of the transfer function $f(x)$. From (2) it is easy to deduce that the steeper is $f(x)$ the sharper are the steps of the cost function E. This produces two contrasting effects on any learning algorithm that moves on this function:

- low β produces a smooth cost function with high gradients at any point and consequent ease of learning. However, it can hide a narrow 'valley' in the cost function landscape;

- high β produces a sharply defined cost function that does not hide any detail but that is very flat when far from the edges. Consequently gradients are small and learning is difficult.

Clearly it is very important to choose β properly. A good theoretical rule would be that of selecting the smallest β that does not hide any detail of the cost function. This would be calculable knowing, for instance, the smallest diameter of the regions of weight space. Unfortunately this is not easy.

Now we show that there is a choice of β that maximizes learning speed. For clarity we consider the contributions to Δw_k of only one example: simplifying the notation of (3) we have (with $\eta = 1$):

$$\Delta w = \beta(\xi - f(\beta x))f'(\beta x)i. \tag{5}$$

For $\beta \rightarrow 0$ we have $\Delta w \rightarrow 0$ since β appears as a factor and all the other factors are limited. In this case the cost function becomes flat everywhere with no gradient. For $\beta \rightarrow \infty$ Δw is always zero except for $x = 0$ because of the derivative $f'(x)$. Substituting the expression for $f(x)$ in (5)

$$\Delta w = \frac{\beta i \, e^{-\beta x}(\xi + \xi \, e^{-\beta x} - 1)}{(1 + e^{-\beta x})^3}$$

Figure 4 contains a plot of Δw as a function of β. To get the position of the maximum we calculate $\partial \Delta w / \partial \beta$. For $\xi = 1$, $i = 1$ and $x > 0$, this derivative goes to zero when

$$\beta x = \frac{e^{\beta x} + 1}{2e^{\beta x} - 1}.$$

This equation has an approximate solution for $\beta_{\mathrm{opt}} \approx 0.89/x$. Similar formulas hold for different values of ξ and i. We can easily write also the formulas for the general case that takes into account all the examples but then the final equation that gives the position of the maximum and the optimal value of β does not have an easy solution.

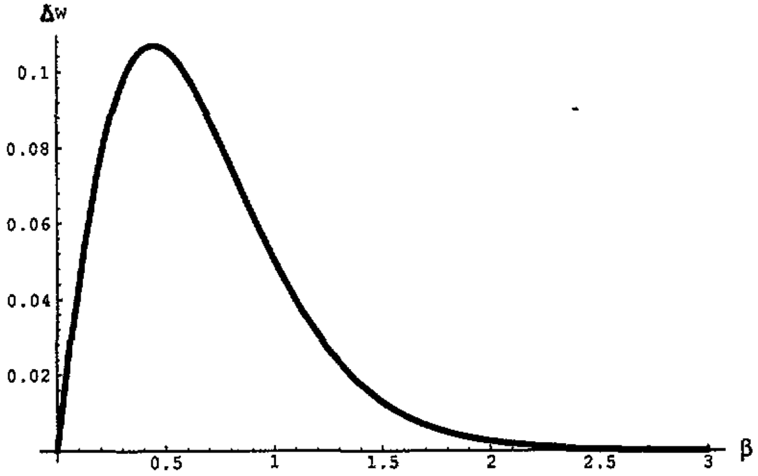


Figure 4. Δw as a function of β with $\xi = 1$, $i = 1$ and $x = 2$.

An approach that gave excellent results in our Perceptron algorithm was to adjust β during the calculations. For a given pattern β_{opt} maximizes Δw and its value is low when far from the edges, i.e. at large x, and high when near to them. In our algorithm, prior to the calculation of the ΔW, we set β to the average of the β_{opt} for all the wrongly classified patterns: this computation is done together with the cost function and does not introduce any significant overhead. A well-tuned β always gives high gradients and a faster learning.

We propose here a simplified version of the complete algorithm:

(1) select a random point in weight space and calculate the value m of the cost function;
(2) calculate the gradient of the cost function;
(3) move along the negative gradient direction until a step in the cost function is found and tentatively accept the new weights;
(4) calculate the value m' of the cost function after the step together with the new value of β;
(5) if $m'=0$ the solution is found: stop.
if $m'<m$ accept the new weights and the new β and go to 2;
if $m'>m$ there is an energy minimum in the cost function: no exact Perceptron solution exists: stop.

An actual implementation of the algorithm needs some care in treating the rounding errors produced in stages 2, 3 and 4 and a more articulated version of the test 5 that allows a recalculation of the gradient in suspect cases. A working *C* program together with algorithm details is available on request.

To test the algorithm we considered a four-dimensional digital Perceptron with 16 examples. With them we built the 65536 possible problems (of which 1882 are soluble) and we ran both algorithms on all problems. For the standard algorithm we used the values of the parameters β and η that optimize its global performances (see the appendix for details).

With the new algorithm we found that the cpu time reduced by a factor 2.4 and the cost function evaluated 20% less times with a mean of 42 times per problem (i.e. 2.6 times for each example).

5. Conclusions

These results sustain the sharp transition existing between the Perceptron with and without a solution. In the first case the cost function is well behaved and gradient descent works smoothly. In the other case the cost function is a maze of relative minima that ultimately trap any form of gradient descent.

Another aspect of the Perceptron without an exact solution is that of the capacity. It has been suggested that the capacity could be much higher if one tolerates a few errors. However, in contrast to this expectation [5] showed that the capacity is always linear in the number of inputs d, independently of the error rate.

The conclusion seems to be that the Perceptron without exact solutions has no advantages: capacity of the same order as the standard Perceptron accompanied by a severe learning difficulty.

The new Perceptron algorithm is undoubtedly faster but we think that its more

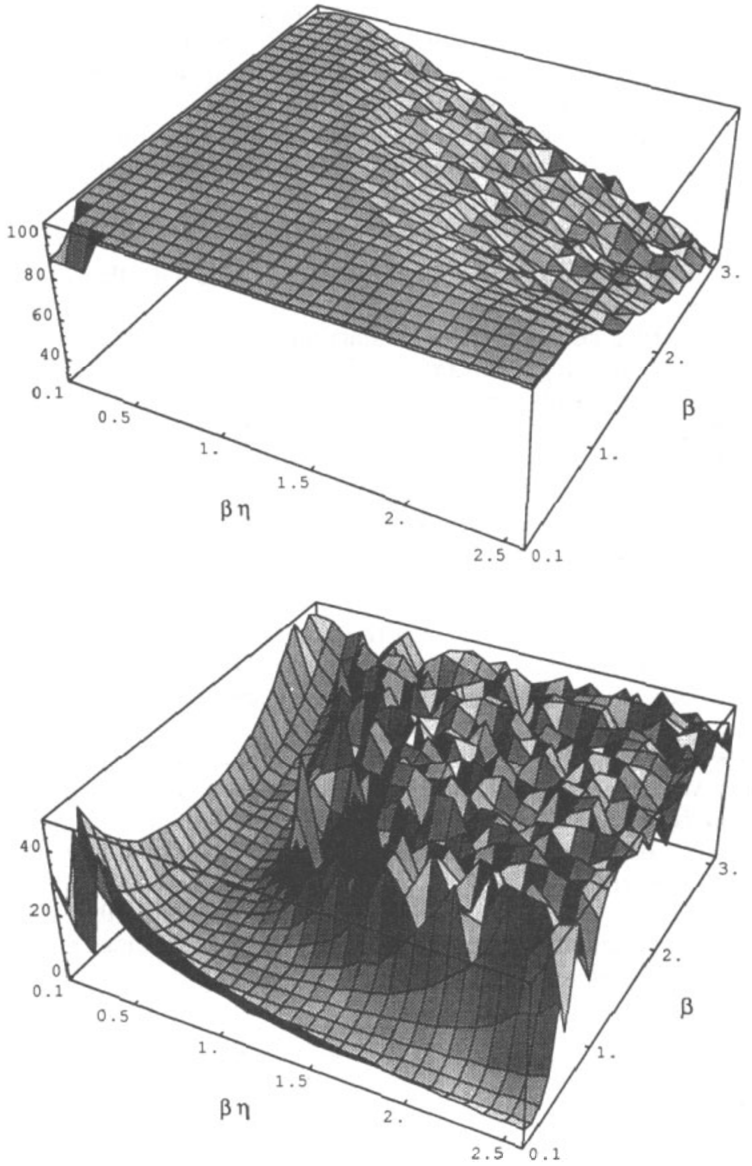


Figure 5. Number of solutions and maximum number of loops vs. β and $\beta\eta$.

important characteristic is its ability to adapt to the peculiarities of the cost function. We point out that it does not have any fixed parameter like η or β to be chosen at the beginning but can select them optimally at any move.

We expect to be able to extend these features to the more general and interesting case of back-propagation for feed-forward networks. This seems to fit nicely with the results presented in [3], that three-layer feed-forward nets, applied to linearly separable problems, are exactly soluble by gradient descent along the cost function.

Acknowledgments

The author warmly acknowledges the illuminating discussions with Edoardo Milotti. Most of the plots of this paper are done with Mathematica.

Appendix

We present here a study of the performances of the standard Perceptron algorithm. We considered a three digital input Perceptron with eight examples and with them we built the 256 possible problems of which 104 are soluble. We ran the algorithm on all problems for several β and η and Figure 5 contains the plot of two quantities chosen to indicate its performances. The first one is the number of solutions actually found (the total number of solutions is 104). The second one is the maximum number of movements in weight space done to solve a problem (bounded in software at 40). The variables are β, the steepness of the transfer function and $\beta\eta$, the length of the moves in weight space (see (3)).

The first plot, that of the solutions found, indicates the wide regions of β and $\beta\eta$ in which the algorithm finds all the solutions. In the second plot we see that the region of optimal performance is rather narrow. The best performances are for $\beta = 0.3$ and $\beta\eta = 1.5$ i.e. $\eta = 5$. In this region the performances do not depend on the step size.

This second plot is quite surprising: it shows that the common choice of a low value of η can give good performances (and quite independently of β) but it shows also that this is not the only possibility and that the best case is in a completely different situation.

References

[1] Minsky M L and Papert S 1988 *Perceptrons* 3rd edition (Cambridge, MA: MIT Press)

[2] Kinzel W and Rujàn P 1990 Improving a network generalisation ability by selecting examples. *Europhys. Lett.* 13(5) 473–477

[3] Gori M and Tesi A 1992 On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14(1) 76–86

[4] Campbell C and Robinson A 1991 On the storage capacity of neural networks with sign constrained weights. *J Phys A: Math Gen* 24 L93–L95

[5] Venkatesh S S and Psaltis D 1992 On reliable computation with formal neurons. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14(1) 87–91

[6] Cover T M 1965 Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers* EC14 326–334

[7] Budinich M and Milotti E 1992 Geometrical interpretation of the back-propagation algorithm for the perceptron. *Physica* 185A (1–4) 369–377

[8] Budinich M 1991 On linear separability of random subsets of hypercube vertices, *J. Phys. A: Math. Gen.* 24 L211–L213