Regression functions by neurocomputing

Nicola Cufaro Petroni[*] and **Massimiliano Nitti**[†]

ABSTRACT: *In this paper a preliminary analysis is performed about the ability of backpropagation neural networks to estimate regression functions from statistical data. In particular the stability under random initial conditions and the optimal choices of learning parameters are investigated. Finally a few examples of regression estimate are shown*

1. INTRODUCTION

Neural networks can be used to solve problems in an adaptive way: in fact they learn the solution through examples. Recently the ability of neural networks to perform statistical tasks has been extensively examinated [1] even if the theoretical basis of this approach still is on shaky grounds. Generally, a learning problem can be viewed as a problem of minimization of a risk functional, whose general expression for a wide class of tasks is:

$$R(\alpha) = \int L\big(y, q(x,\alpha)\big)\, dF(x,y) \tag{1}$$

where α is a vector of parameters, and $q(x,\alpha)$ represents the function that the learning machine must retrieve and which depends continuously on α. Of course the functional $L\big(y, q(x,\alpha)\big)$ depends on the particular task that must be performed: In particular the problems that can be approached in the statistical learning theory are [2]:

- Pattern recognition
- Regression estimate
- Density estimate

and the functionals associated are respectively

$$L(y, q(x,\alpha)) = \begin{cases} 0, & \text{if } y = q(x,\alpha); \\ 1, & \text{if } y \neq q(x,\alpha); \end{cases}$$

$$L(y, q(x,\alpha)) = \big[y - q(x,\alpha)\big]^2$$

$$L(q(x,\alpha)) = -\ln q(x,\alpha)$$

In this paper we will analyze the ability of a neural network to solve a problem of regression estimate, a problem important in several respects (from forecasting trends in industrial ouputs, to extracting relations among random variables in production process [3]). The neural networks

[*] INFM and Dipartimento di Fisica dell'Universtitá di Bari, via Amendola 173, 70126 Bari (Italy); E-mail CUFARO@BARI.INFN.IT

[†] MER-MEC, via Oberdan 68, 70043 Monopoli (Italy).

used in this paper are back propagation networks with three layers of neurons: the first layer is an input layer with just one neuron which has only a link role; the second is a hidden layer of k neurons, with sigmoidal transfer function; and finally, the output layer is composed of one neuron with linear transfer function. The input and output of the network are real (for details about neural networks and neurocomputing an extensive literature is today avaliable [4, 5, 6]).

If we consider a set of training examples $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, our neural network will, in the training phase, minimize the functional

$$H(w) = \sum_{i=1}^{n} \big(y_i - h(x_i, w)\big)^2 \tag{2}$$

where h are the transfer functions implemented by the network: they are parametrized by the weight matrix w of the network and can be written as:

$$h(x; w) = \sum_{j=0}^{k} \beta_j \Phi(u_j x - \theta_j) \tag{3}$$

where $w = (\beta, u, \theta)$ and

$$\Phi(x) = \frac{1}{1 + e^{-x}} \, .$$

In the training phase, the network find the optimal weight matrix, which minimizes $H(w)$, using the *generalized delta rule*. The generalize delta rule, also called gradient descent method, consists of two phases. In the first an initial weight matrix is randomly taken; in the second phase the learning mechanism starts: this consist in the updating of the w matrix following the prescription

$$w(n+1) = w(0) + \eta \nabla H(w(n))$$

where η is a parameter called learning rate. This updating is iterated many times, until a *good* approximation is achived. If N is the number of iterations, we have

$$w(N) = w(0) + \eta \sum_{i=1}^{N} \nabla H(w(i))$$

where $w(0)$ is the initial matrix. The learning process finally stops when one of the following conditions is satisfied:

1. The number of iteration reaches a fixed value;
2. The quantity $\nabla H(w)$ is zero, namely the functional H keeps a steady value.

This algorithm determines a minimum of $H(w)$ that however can in general be a local minimum: a problem that must be taken into account in every application of neural networks where absolute minima are generally required.

Why the regression problem can be approached by means of a neural network? To address this question is necessary to recall something about regression. Let X and Y be (a.c. to simplify the notations) random variables and $f_{Y|X}(y|x)$ the conditional density of Y with respect to X. The regression function is defined as:

$$g_Y(x) = \mathbf{E}[Y|X = x] = \int_{-\infty}^{+\infty} y f_{Y|X}(y|x)\, dy \tag{4}$$

If on the other hand $g(x)$ is an arbitrary function in L_2 it is well known that

$$\mathbf{E}|g(X) - Y|^2 \geq \mathbf{E}\big|\mathbf{E}[Y|X] - Y\big|^2 = \mathbf{E}\big|g_Y(X) - Y\big|^2, \qquad \forall\, g \in L_2 \tag{5}$$

namely that the random variable $g_Y(X) = \mathbf{E}[Y|X]$ minimizes the quantity $\mathbf{E}\big|g(X) - Y\big|^2$, in the sense that the minimum of the functional $\mathcal{E}[g] = \mathbf{E}|g(X) - Y|^2$ is realized exactly by $g_Y(x) = \mathbf{E}[Y|X = x]$. This remark points out that the regression $g_Y(x)$ can be determined as the solution of a suitable variational problem. However, since in general we have only a random sample of values of X and Y, we will be obliged to estimate the regression function form the sample average

$$H^*(\varphi) = \frac{1}{n}\sum_{j=1}^{n}\big(\varphi(x_j) - y_j\big)^2 \tag{6}$$

by determining the continuous function $\overline{\varphi}(x)$ that minimizes H^*. The evident analogy between (2) and (6) justifies now the use of a neural network to estimate the regression: if we consider for our back propagation neural network a set of training which is a sample from X and Y, the network will learn to implement a function $\overline{\varphi}(x)$ which is an estimate of the regression function $g_Y(x)$. In other words, since the general task of a neural network is to minimize the functional $H(w)$ in (2), it implements a mean square estimation metod to find an approximation to the regression function. Moreover, since the backpropagation neural networks are able to approximate a wide class of functions [7], they are also a suitable tool for the determination of non-linear regression.

Since it is not easy to prove theoretical results about the neural network performances, in this paper we will limit ourselves to the analysis of a few empyrical results. We will extract a set of sample by two simulated RV's and we will teach them to the network; then we will assess the results by means of a few suitable parameters by comparing the network performance with an *a priori* given regression function. In particular X will be uniformly distribuited in $[0, 1]$ and

$$f_{Y|X}(y|x) = \frac{1}{\sqrt{2\pi}\sigma} e^{(g(x)-y)^2/2\sigma^2} \tag{7}$$

so that $\mathbf{E}[Y|X = x] = g(x)$, where $g(x)$ is an a priori given function. As a consequence the sample of the RVs X and Y to be used to test the network performance will be generated by an appropriate computer program, for given regression function $g(x)$, number of sample's elements and variance σ^2.

The quantities used in the assessment of the performances will be the following:

1) energy of network:

$$H(w) = \sum_{i=1}^{n}\big(y_j - h(x_j, w)\big)^2\,; \tag{8}$$

it is the square deviation of the output $h(x;w)$ of the network from the requited answers y_i. As we have already said, this is the quantity that the generalized delta rule algorithm minimizes in the learning phase;

2) mean Square Error:

$$\epsilon(w) = \frac{1}{n}\sum_{i=1}^{n}\big(h(x_i,w) - g(x_i)\big)^2 \, ; \tag{9}$$

it measures the mean square deviation from the required regression $g(x)$ of the function $h(x;w)$ eventually implemented by the network.

3) maximum deviation:

$$\mu(w) = \max_i \big|h(x_i,w) - g(x_i)\big| \, ; \tag{10}$$

it evaluates the maximum deviation between the required values $g(x_i)$ of the regression and the outputs $h(x;w)$ of the network.

It is clear from the definitions that, while the learning algorithm works to minimize H, nothing guarantees that ϵ and μ too will be minimized. Moreover the two performance parameter (ϵ and μ) defined form the a priori knowledge of the regression $g(x)$ can be minimal under different conditions, so that in some performances we could have large ϵ and small μ, or vice-versa. In this paper we will consider in particular three preliminary problems: 1) indipendence of the performance form the initial conditions; 2) relation between η and N to obtain an optimal performance; 3) dependence of the performance on the number of neurons in the hidden layer. An investigation of these questions is indeed necessary as a preparation to the subsequent extensive simulations needed to comparatively assess the network performances with respect to other standard methods of regression estimate.

2. INDEPENDENCE OF LEARNING FROM THE INITIAL CONDITIONS

The initial conditions of the simulations are expressed by a matrix of weights $w = (\beta, \theta, u)$ where, as in (3) β represents the weights of the single sigmoidal trasformation in the linear combination of output neuron; θ represents the thresholds of hidden neurons; u represents the values of the sigmoid function derivatives in the hidden neurons. However, since

$$\Phi(ux - \theta) = \frac{1}{1 + e^{-(ux-\theta)}} = \frac{1}{1 + e^{-u(x-c)}}$$

with $c = \theta/u$ centre of the sigmoid, we can also consider as weights of the network the vetor $w = (\beta, u, c)$. This will be useful in the light of the following remarks. As already stated, the generalized delta rule consists of two phases: in the first an initial weight matrix is randomly taken; in the second phase the learning mechanism starts, consisting in the following updating of the w matrix:

$$w(n+1) = w(0) + \eta \nabla H\big(w(n)\big) \, .$$

This updating is iterated many times, until a good approximation is achived and, if N is the number of iterations, we will have

$$w(N) = w(0) + \eta \sum_{i=1}^{N} \nabla H\big(w(i)\big)$$

where $w(0)$ is the initial matrix. The learning process stops if one of the following conditions is satisfied:

1. the number of iterations reaches an a priori fixed value;
2. the quantity $\nabla H(w)$ is zero.

However, even if the condition 2 can be satisfied in many points in the space of weights w, just a few of these points will be acceptable (absolute minima of H). Hence, to guarantee that the system will eventually reach an acceptable solution, the values of the initial weight matrix could be very relevant. For our kind of problem it is realistic to think that a good initial condition is obtained when c (the centre of the sigmoid finctions) is in [0,1] while u and β must be empyrically determined. To test these ideas we have fixed the network architecture: 1 neuron of input layer, 6 neurons of hidden layer, 1 neuron of output layer. We fixed also $N = 2000$ and $\eta = 0.05$ and we have taken a sample set of 50 elements with $g(x) = 4x(1-x)$ and $\sigma = 0.3$.

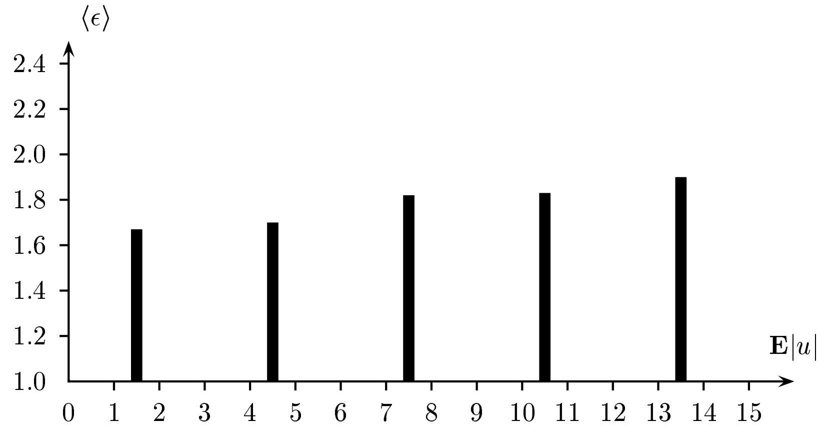

Fig. 1 Network performance as measured by $\langle\epsilon\rangle$ in function of $\mathbf{E}|u|$

With these fixed data we have tested the performance stability in different regions of the weight space: in each region we have generated 100 initial weight matrices and we have used it to run the training of the network. If $w^{(k)}$ is the k-th initial weight matrix, we calculate the corresponding ϵ_k and μ_k and then we consider the averages

$$\langle\epsilon\rangle = \frac{1}{100}\sum_{k=1}^{100}\epsilon_k\,, \qquad \langle\mu\rangle = \frac{1}{100}\sum_{k=1}^{100}\mu_k$$

and the standard deviations

$$S_\epsilon = \sqrt{\frac{1}{99}\sum_{k=1}^{100}\big(\epsilon_k - \langle\epsilon\rangle\big)^2}\,, \qquad S_\mu = \sqrt{\frac{1}{99}\sum_{k=1}^{100}\big(\mu_k - \langle\mu\rangle\big)^2}\,.$$

In the first test we have generate the initial matrices with weights β, c and u all uniformly distributed in [-3,3]. In the second test the initial matrices have weights uniformly distributed in the following intervals: c in $[0,1]$, u in $[-3,3]$, and β in $[-1,1]$. We then obtained the following results:

in the first test

$$\langle \epsilon \rangle = 0.19\,, \qquad S_\epsilon = 0.019\,; \qquad\qquad \langle \mu \rangle = 0.32\,, \qquad S_\mu = 0.038\,;$$

in the second test

$$\langle \epsilon \rangle = 0.17\,, \qquad S_\epsilon = 0.011\,; \qquad\qquad \langle \mu \rangle = 0.29\,, \qquad S_\mu = 0.024\,.$$

As can be seen, the performance is rather stable in this situation even if greater stability (smaller standard deviations) and better performances are exhibited in the second simulation, pointing to a slight preference for the second choice in the distribution of initial data. Moreover, since for larger values of u the learning is more influenced by the sample noise, we have tested the performances of the network against larger and larger values of $|u|$. We have generated 100 initial matrices with c uniformly distributed in $[0,1]$, β in $[-1,1]$ and $|u|$ uniformly distribuited respectively in $[0,3]$, $[3,6]$, ..., $[12,15]$. The results are summarized in Fig. 1 and 2.

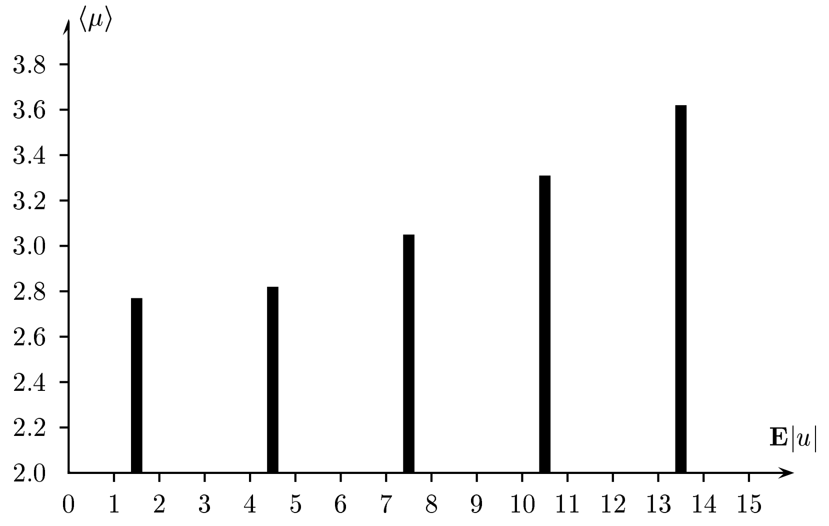

Fig. 2 Network performance as measured by $\langle \mu \rangle$ in function of $\mathbf{E}|u|$

and they constitute an empyrical confirmation of the fact that the values of $|u|$ must be kept in intervals small enough to make the performance not too sensitive to the sample noise.

3. RELATION BETWEEN η AND N IN OPTIMAL PERFORMANCES

Since the weight updating equations are

$$w(N) = w(0) + \eta \sum_{i=1}^{N} \nabla H\big(w(i)\big)\,,$$

the following parameters are of relevance along the training: the learning rate η, the number of iterations N, the initial weight matrix and the number of the sample elements. In the updating equation the initial matrix, the learning rate and the number of iterations are present in an explicit way. In an implicit way there is also the number of the sample elements which appears in the $\nabla H\big(w(i)\big)$.

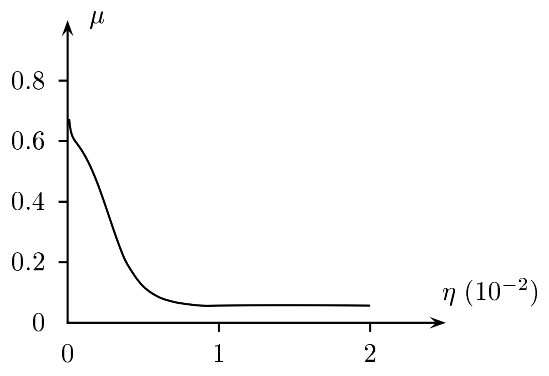

Fig. 3 Dependence of the maximal deviation μ on the learning rate η.

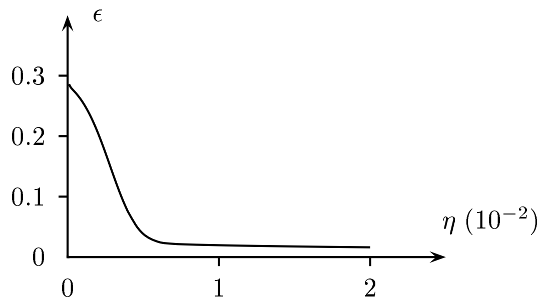

Fig. 4 Dependence of the mean square deviation ϵ on the learning rate η.

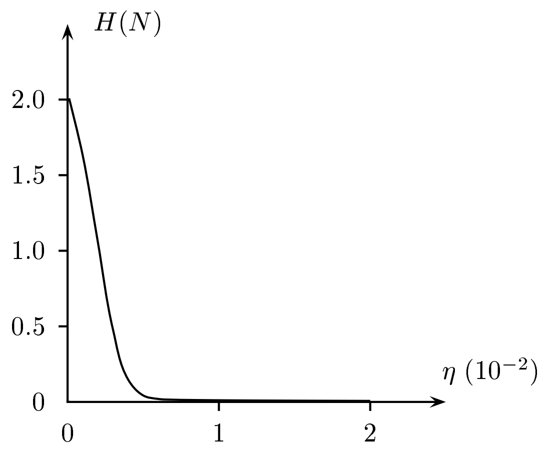

Fig. 5 Final value of the energy H as a function of the learning rate.

We will analyze here only the relation betwen η and N, with particular interest for the so- called overtraining, or overfitting [4, 5]. Let us initially suppose that the random variables X and Y are in a very simple functional dependence, so that X is uniformly distributed in $[0, 1]$, and $Y = g(X)$. A sample of X and Y, will represent a sample of the function $g(x)$. In this case, for a large enough sample, the neural network should realize a good approximation of the function $g(x)$.

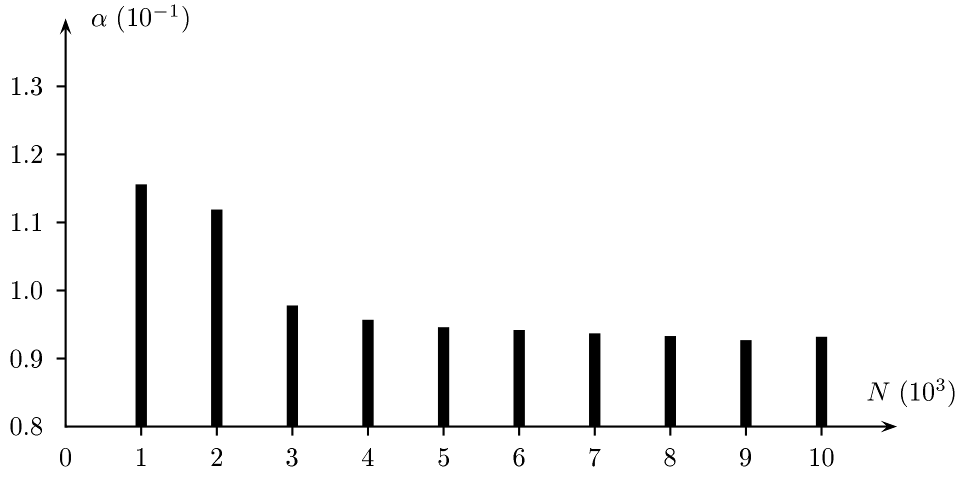

Fig. 7 Dependence of the minimal mean square error α on the number N of presentations

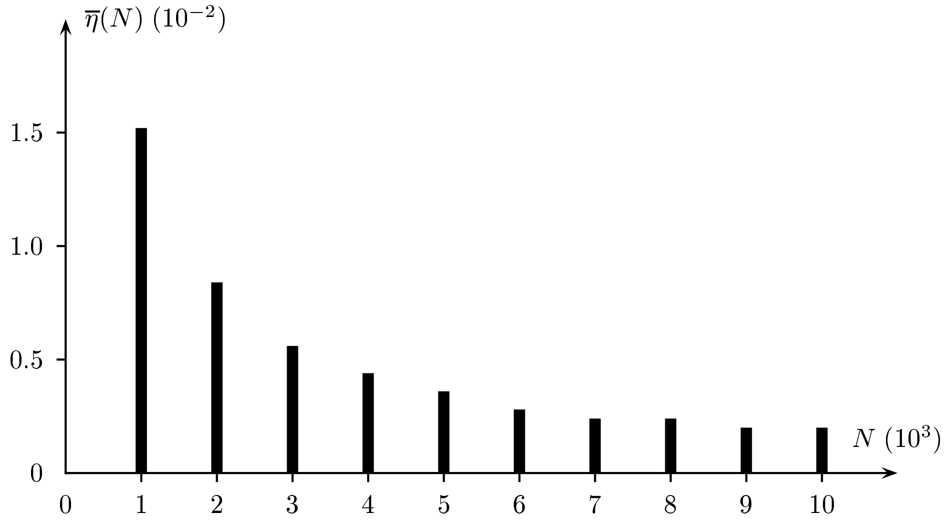

Fig. 8 Dependence of the optimal (w.r.t. the mean square error) learning rate $\overline{\eta}$ on the number N of presentations.

If now we keep fixed the initial weight matrix, the network architecture, the amplitude of the sample and the regression function exactly as in section 2, but we allow the learning rate to vary from 0.0001 to 0.02 (with a step 0.0001), we obtain the results summarized in Fig. 3 and 4: the first shows how μ changes with η; the second shows how ϵ changes with η. It is important to remark here that the network errors are monotonically decreasing vith the values of the learning rate. Moreover Fig. 5 shows how the final (just before the updating algorithm stops) value $H(N)$ of the energy changes with η.

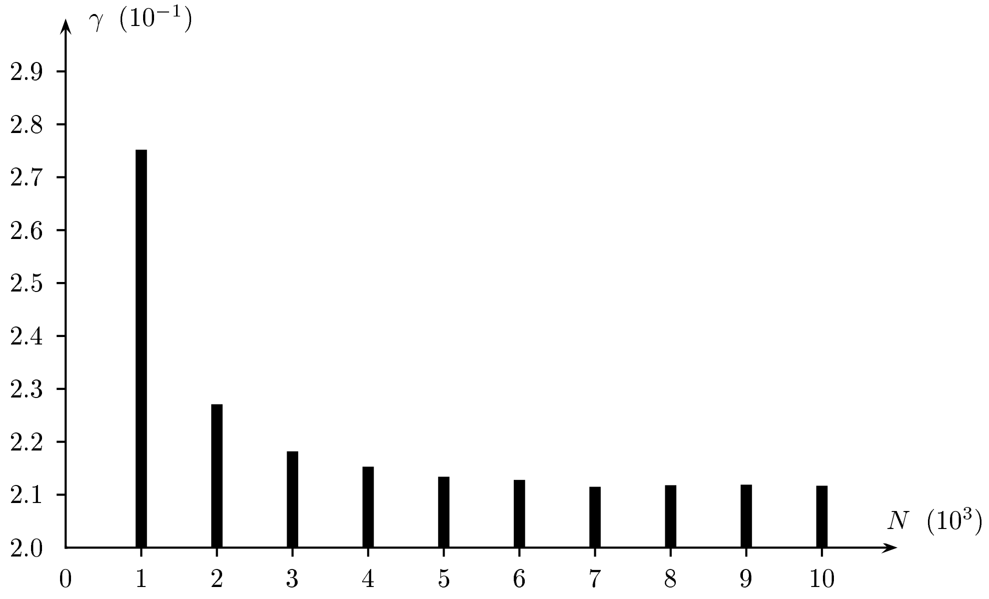

Fig. 9 Dependence of the minimal maximum deviation γ on the number N of presentations

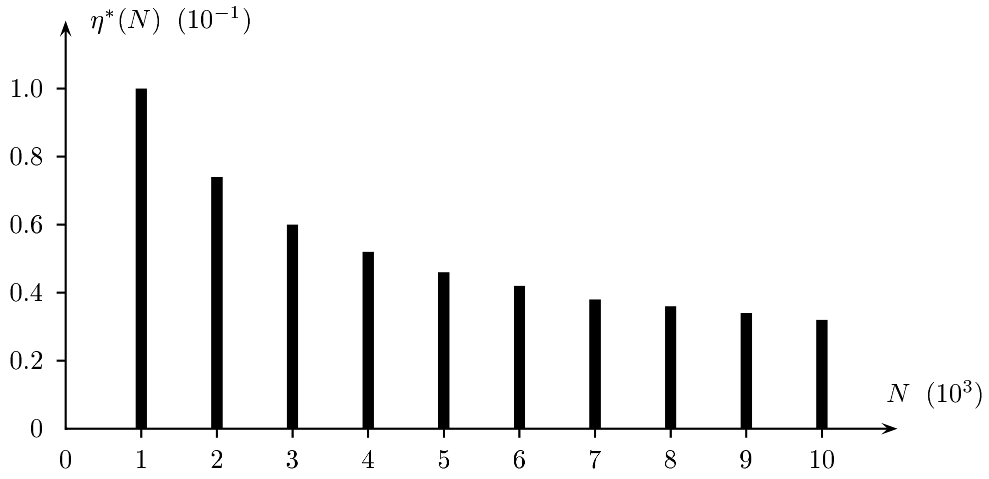

Fig. 10 Dependence of the optimal (w.r.t. the minimal deviation) learning rate η^* on the number N of presentations.

As a consequence these diagrams describe how the network performances variate variates with the value of the learning rate and it individuates two regions of the η values: for $\eta \leq 0.005$ the learning rate is too small to make the network properly converge (this region is characterized by non zero values of the H derivative); on the other hand for $\eta \geq 0.005$ the last values of the energy derivative is zero indicating that now the network converged to a stable state. Of course this type of problems can be viewed as a particular (and simpler) case of regression determination: that characterized by a deterministic relation between the two variables with no noise on the sample. However in general there is no functional relation between X and Y, but only a statistical relation which makes meaningful a regression analysis. Hence we have analyzed the performances of the neural network for a sample of two random variables in the same statistical relation described in the Section 2. To determine the relation between the optimal values of η and N we must analyze the global behaviour of the two functions $\epsilon(\eta, N)$ and $\mu(\eta, N)$, the other parameter being kept fixed. In order to do that we have defined some auxiliary functions: first of all we define

$$\alpha(N) = \min_{\eta} \epsilon(\eta, N) \tag{11}$$

which is plotted in Fig. 7. It is remarkable that this function is monotonically decreasing. Now let us call $\overline{\eta}(N)$ the particular value of the variable η such that, for an arbitrary given N, we get

$$\epsilon\big(\overline{\eta}(N), N\big) = \alpha(N)\,. \tag{12}$$

The previous relation implicitely defines the function $\overline{\eta}(N)$ plotted in Fig. 8 which represents the optimal value of the learning rate for a given number N of presentations. In other words the function $\overline{\eta}(N)$ summarizes the couples of values of η and N that these parametrs must assume to have an optimal performance in terms of ϵ. The same analysis can be carried out for the second error parameter μ: we define

$$\gamma(N) = \min_{\eta} \mu(\eta, N) \tag{13}$$

a function plotted in Fig. 9, and then we call $\eta^*(N)$ the particular value of the variable η such that, for an arbitrary given N, we get

$$\mu\big(\eta^*(N), N\big) = \gamma(N)\,. \tag{14}$$

This implicitely defines the function $\eta^*(N)$ plotted in Fig. 10. It is remarkable that the curves $\overline{\eta}(N)$ and $\eta^*(N)$ are far from coincident and respect the general rule $\overline{\eta}(N) < \eta^*(N)$. This witnesses the fact that the two performance parameters ϵ and μ are in fact rather different so that same sort of compromise must be hammered in the practical cases between the two conflicting requirements of ϵ and η minimization.

4. DEPENDENCE OF THE PERFORMANCE ON THE NUMBER OF NEURONS IN THE HIDDEN LAYER

The last test tries to determine how the performance of the network variates with the number of neurons in the hidden layer. It is known [5] that there are methods to calculate the number of hidden neurons in order to achieve good performances. This is often calculated from the number of extremal points of the function to approximate plus 1. If this function is $4x(1-x)$, as happens in our simulations, the number is 2. If the number of neurons increses, the performance of network improves up to a point: after that it worsens. This has two reasons: first of all the computational complexity of the network increases and the system converge more slowly. Moreover a large number of neurons make the network too prone to be influenced by the noise. To quantitatively test these behaviours.

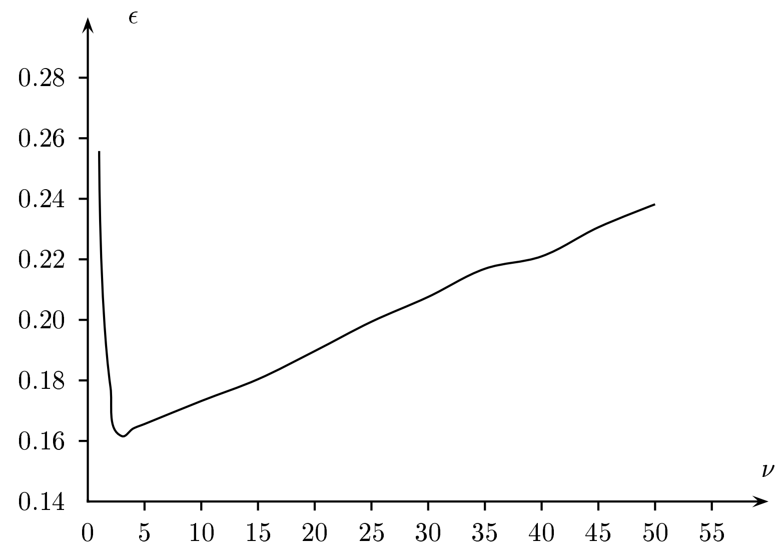

Fig. 11 Mean square error as a function of th number of hidden neurons.

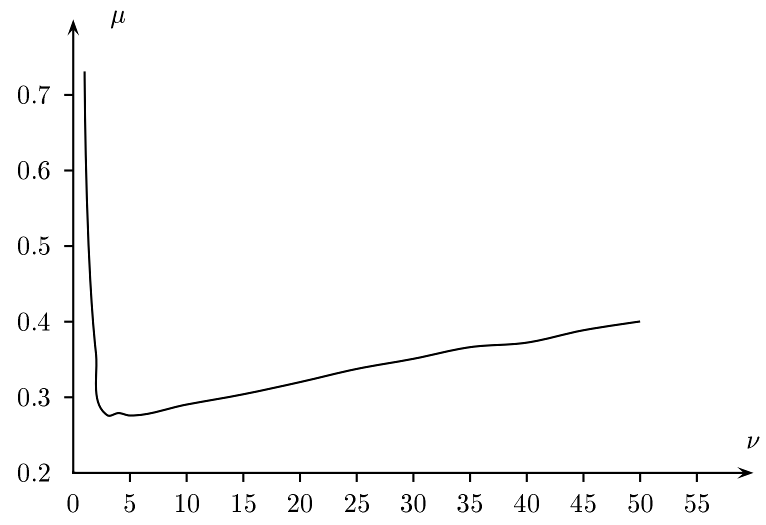

Fig. 12 Maximum deviation as a function of the number of hidden neurons.

We have now fixed a network architecture in this way: 1 neuron of input layer; 1 neuron of output layer; $N = 2000$; $\eta = 0.005$ (an evident compromise between the values $\overline{\eta}(2000) \simeq 0.008$ and $\eta^*(2000) \simeq 0.07$). On the other hand the number ν of neurons in the hidden layer is alloved to run from 5 to 50 by steps of 5 neurons. Moreover the sample consists of 50 elements and as usual $g(x) = 4x(1-x)$ and $\sigma = 0.3$. As both Fig. 11 and 12 show, the performance clearly worsens (both in terms of η and μ) when the number ν of hidden neurons increases beyond a few units. This confirms the well known fact that too *clever* neural networks can have a *stupid* behaviour in the sense that they learn too much, including the noise.

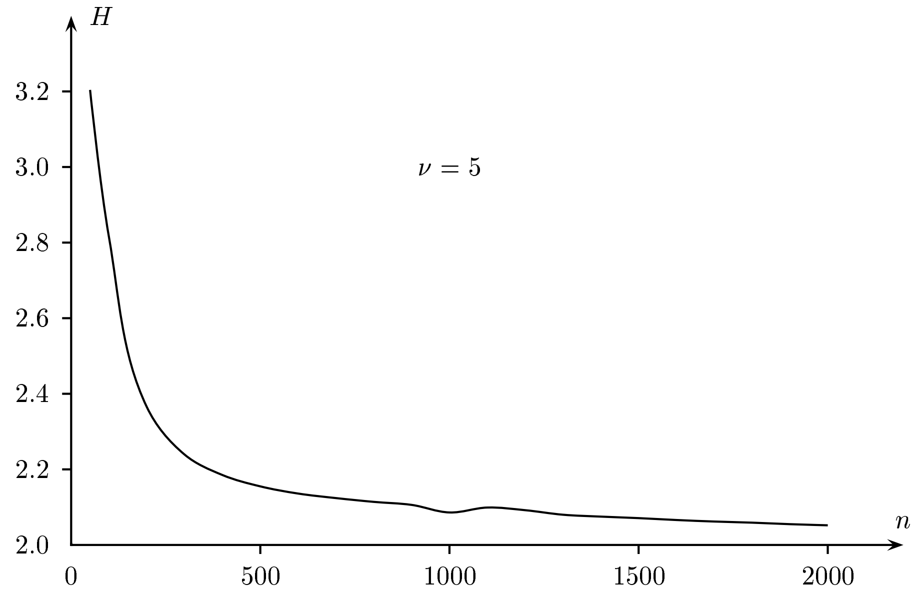

Fig. 13 Fast fall of the energy H for a network with 5 hidden neurons.

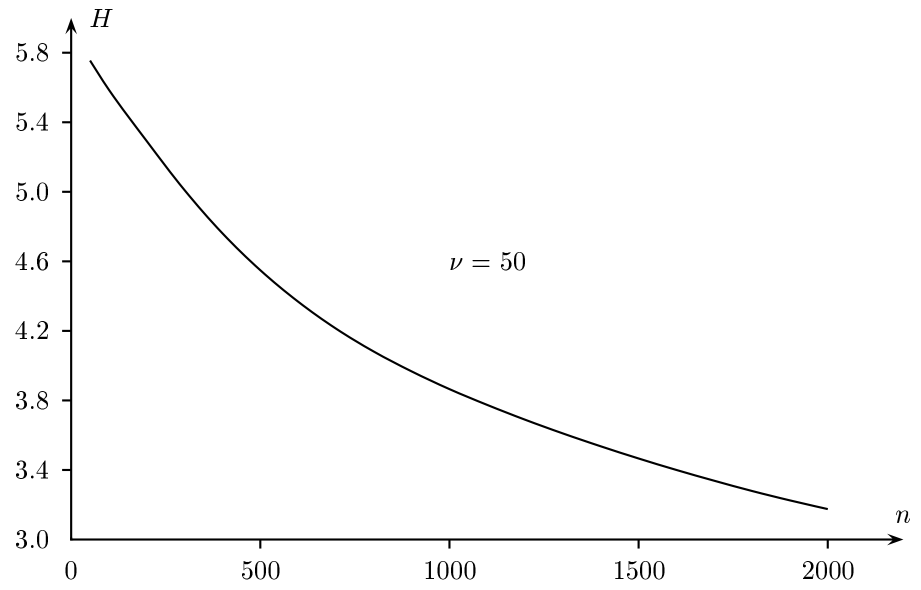

Fig. 14 Slow drop of the energy H for a network with 50 hidden neurons.

This is the very analog of a problem of filtering in the sense that if the filter has a too wide bandwidth it will inevitably accept signal *and* noise and it will reproduce the actual signal in a way too faithful with no cleaning of the signal from the noise. On the other hand a filter with a too narrow bandwidth (the analog of a neural network with only a few hidden neurons) will much more effectively eliminate the noise, but it will also impaire the signal that we would like to extract from the noise with an unacceptable level of distortion. As always in these situations the difficult task is to assess the values of the parameters (number of hidden neurons, filter bandwidth) which optimize the performance of the system. As another indication of this situation we analyzed the behaviour of the energy $H(N)$ both in the case of 5 and 50 hidden neurons. Fig. 13 then shows that in a learning process with 5 hidden neurons the function $H(N)$ decreases much more quickly than in the case of 50 neurons (Fig. 14).

5. CONCLUSIONS

We will conclude this paper by showing an example of regression estimate by means of a backpropagation neural network in one of the typical configurations discussed in the previous sections. The results are summarized in Fig. 15.

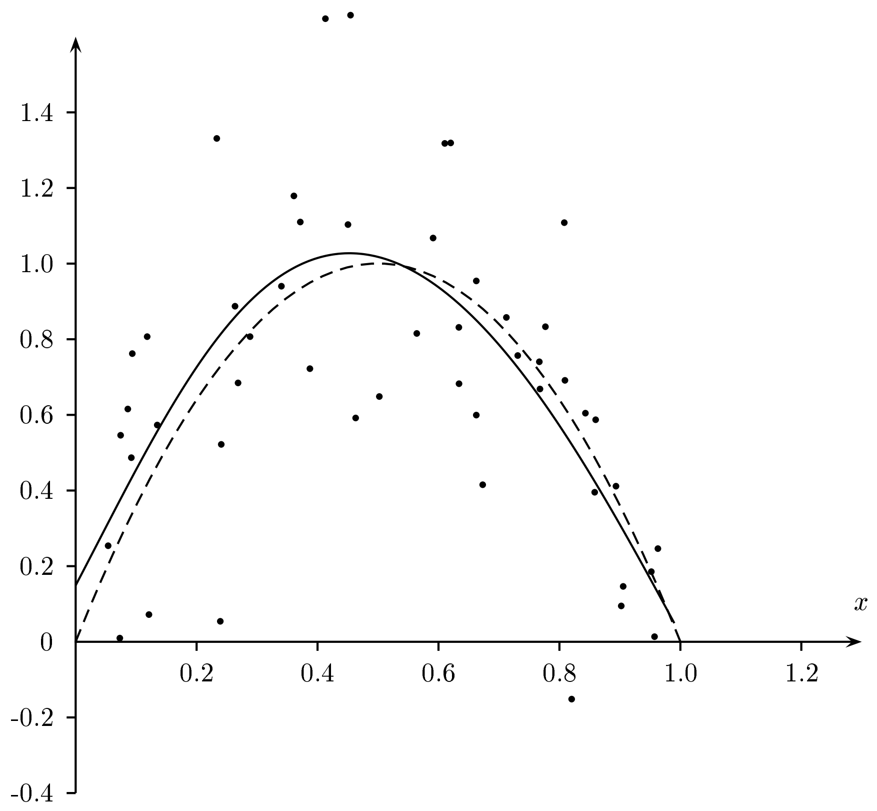

Fig. 15 Regression function (dashed line) and network answer (solid line) from a 50 elements sample (dots).

The 50 points sample has been generated by supposing a regression theoretical function $g(x) = 4x(1-x)$ (shown as a dashed line in Fig. 15), with a superimposed gaussian noise with $\sigma = 0.3$.

The backpropagation neural network had just one neuron in the output and in the input layers and six neurons in the hidden layer. The learning rate was chosen as $\eta = 0.05$ and the maximum number of iterations was $N = 2000$. The initial values of the network parameters were uniformily distributed in suitable intervals: the centers c of the sigmoidal functions are in $[0, 1]$, the derivatives u of sigmoids in $[-3, 3]$ and the weights β of the sigmoidal transformations in $[-1, 1]$. In this configuration the network will learn to approximate the regression function in 2 or 3 minutes (with a 486 PC at 100 MHz). Unfortunately these timings grow very quickly with the number N of iterations so that the training phase would be considerably longer (of the order of 10^2 minutes) for $N = 10000$ iterations. However with faster PC's and a suitable design of the simulation it would be surely possible to cut these times at a reasonable size.

R E F E R E N C E S

1) F.Murtagh: *Int. Stat. Rev.*, 62 (1994) 275.

2) V.N.Vapnik: *The nature of statistical learning theory*, Springer, New York, 1995.

3. A.Sen and M.Srivastava: *Regression analysis*, Springer, New York, 1990.

4. R.Hecht-Nielsen: *Neurocomputing*, Addison-Wesley, Reading, 1990.

5. J.Hertz, A.Krogh and R.G.Palmer: *Introduction to the theory of neural computation*, Addison-Wesley, Redwood, 1991.

6. B.D.Ripley: *Pattern recognition and neural networks*, Cambridge Unuiversity Press, Cambridge, 1996.

7. T.Poggio and F.Girosi: *A theory of networks for approximation and learning*, A.I.Memo No. 1140, M.I.T. Artificial Intelligence Laboratory, 1989.