The Best Neural Network Architecture

Angel Kuri-Morales

Instituto Tecnológico Autónomo de México
Río Hondo No. 1
México 01000, D.F.
México
akuri@itam.mx

Abstract. When designing neural networks (NNs) one has to consider the ease to determine the best architecture under the selected paradigm. One possible choice is the so-called multi-layer perceptron network (MLP). MLPs have been theoretically proven to be universal approximators. However, a central issue is that the architecture of the MLPs, in general, is not known and has to be determined heuristically. In the past, several such approaches have been taken but none has been shown to be applicable in general, while others depend on complex parameter selection and fine-tuning. In this paper we present a method which allows us to determine the said architecture from basic theoretical considerations: namely, the information content of the sample and the number of variables. From these we derive a closed analytic formulation. We discuss the theory behind our formula and illustrate its application by solving a set of problems (both for classification and regression) from the University of California at Irvine (UCI) data base repository.

Keywords. Neural Networks, Perceptrons, Information Theory, Genetic Algorithms.

1. Introduction.

In the original formulation of a NN a neuron gave rise to a simple analogy corresponding to a perceptron, shown in Figure 1. In this perceptron $y_i = \varphi\left(\sum_{j=0}^{m_O} w_{ij} x_j\right)$; where $x_0=1$; $w_{i0}=b_i$. The weights (w_{ij}) employed here define the coupling strength of the respective connections and are established via a learning process, in the course of which they are modified according to given patterns and a learning rule. Originally, the process of learning was attempted by applying individual perceptrons but it was shown [1] that, as individual units, they may only classify linearly separable sets. It was later shown [2] that a feed-forward network of strongly interconnected perceptrons may arbitrarily approximate any continuous function.

In view of this, training the neuron ensemble becomes a major issue regarding the practical implementation of NNs. Much of the success or failure of a particular sort of

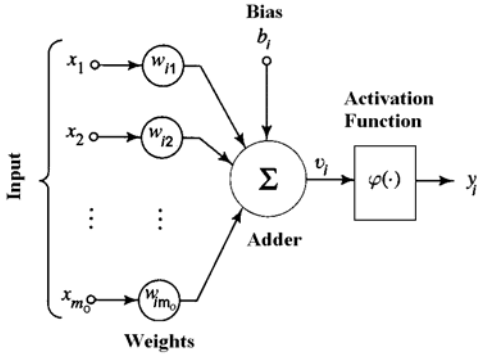

Fig. 1. A Perceptron.

NN depends on the training algorithm. In the case of MLPs its popularity was boosted by the discovery of the back-propagation learning rule [3]. It is a simple and efficient iterative algorithm which, by requiring a differentiable activation function, radically speeds up and simplifies the training process. The theoretical formalization of these basic concepts may be traced back to the original proof [4] of the Universal Approximation Theorem (UAT) which may be stated as follows:

"Let $\varphi(\cdot)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let I_{m_o} denote the m_O-dimensional unit hypercube $[0,1]^{m_O}$. The space of continuous functions on I_{m_o} is denoted by $C(I_{m_o})$. Then, given any function and $f \in (I_{m_o})$ and $\varepsilon > 0$, there exist an integer M and sets of real constants α_i, β_i, w_{ij}, where $i=1,2,...,m_I$ and $j=1,2,...,m_O$ such that we may define:

$$F(x_1,...,x_{m_O}) = \sum_{i=1}^{m_I}\left[\alpha_i \cdot \varphi\left(\sum_{j=0}^{m_O} w_{ij}x_j\right)\right] \quad (1)$$

as an approximate realization of the function $f(\cdot)$, that is,

$$|F(x_1,...,x_{m_O}) - f(x_1,...,x_{m_O})| < \varepsilon \quad (2)$$

for all $x_1,...,x_{mO}$ in the input space."

□

The UAT is directly applicable to multilayer perceptron networks [5] and states that *a single hidden layer is sufficient for a multilayer perceptron to compute a uniform* ε *approximation to a given training set of pairs represented by a) The set of inputs* $x_1,...,x_{mO}$ *and b) A desired (target) output* $f(x_1,...,x_{mO})$.

To take practical advantage of the UAT, data must be mapped into the [0,1] interval. If the data set does not represent a continuous function, though, the UAT does not generally hold. This is the main reason to include a second hidden layer. This second layer has the purpose of mapping the original discontinuous data to a higher dimensional space where the discontinuities are no longer present [6].

However, it is always possible to replace the original discontinuous data by a continuous approximation with the use of a natural spline (NS) [7]. By properly using a

NS, the user may get rid of the necessity of a second hidden layer and the UAT becomes truly universal. A discontinuous function interpolated with NS is shown in figure 2. On the left, an original set of 16 points; on the right 100 equi-distant interpolated points. Notice that all the original points are preserved and the unknown interval has been filled up with data which guarantees the minimum curvature for the ensemble. A similar effect is achieved by including a second hidden layer in a NN. What we are doing is relieving the network from this tacit interpolating task.

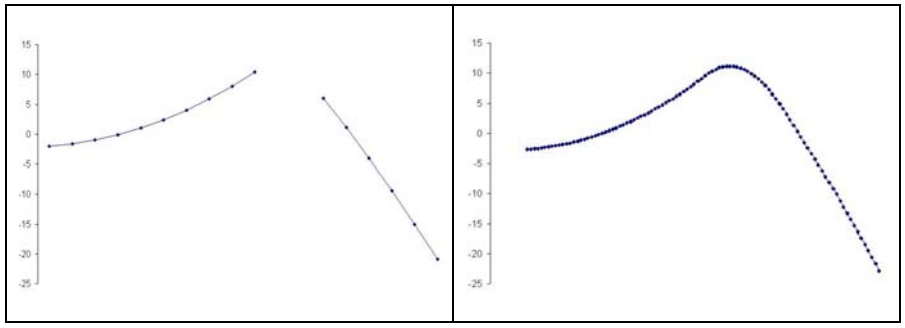

Fig. 2. Use of a Natural Spline to avoid discontinuities.

Later studies led to alternative approaches where the architecture is rendered by the paradigm. Among them we find the Radial-Basis Function Networks (RBFN) [8] and the Support Vector Machines (SVM) [9]. Graphical representations of both paradigms are shown in Figure 3. Notice that MLPs may have several output neurons.

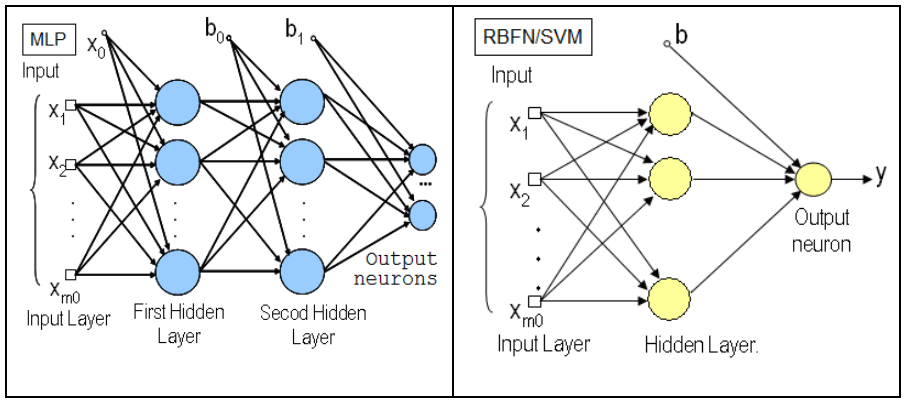

Fig. 3. Basic architecture of a Neural Network.

RBFNs and SVMs are well understood and have been applied in a large number of cases. They, nonetheless, exhibit some practical limitations [10]. For example, as opposed to MLPs, RBFNs need unsupervised training of the centers; while SVMs are

unable to directly find more than two classes. For this reason, among others, MLPs continue to be frequently used and reported in the literature.

The architecture of a MLP is completely determined by a) m_O (the input neurons), b) m_I (the hidden neurons), c) $\varphi(\cdot)$ (the activation function) and d) the w_{ij}'s. With the exception of m_I, all the values of the architecture may be readily found. m_I remains to be determined in every case and is not, in general, simple to assess. It determines 1) The complexity of the network and, more importantly, 2) Its learning capability. The proper and closed determination of m_I is the central topic of this work. The rest of the paper is organized as follows. In part 2 we derive a closed formulation for the lower bound on the value of m_I. In part 3 we present some experimental results. In part 4 we present our conclusions.

2. Determination of a Lower Bound on m_I

There have been several previous approaches to determine the value of m_I. A dynamic node creation algorithm for MLPs is proposed by Ash in [11], which is different from some deterministic process. In this algorithm, a critical value is chosen arbitrarily first. The final structure is built up through the iteration in which a new node is created in the hidden layer when the training error is below a critical value. On the other hand, Hirose et al in [12] propose an approach which is similar to Ash [11] but removes nodes when small error values are reached. In [13], a model selection procedure for neural networks based on least squares estimation and statistical tests is developed. In [14] Yao suggests an evolutionary programming approach where the architecture of a MLP is evolved and the actual connections may be optimized along with the number of hidden neurons. The Bayesian Ying-Yang learning criteria [15, 16] put forward an approach for selecting the best number of hidden units. Their experimental studies show that the approach is able to determine its best number with minimized generalization error, and that it outperforms the cross validation approach in selecting the appropriate number for both clustering and function approximation. In [17] an algorithm is developed to optimize the number of hidden nodes by minimizing the mean-squared errors over noisy training data. In [18] Fahlmann proposes a method dynamically increasing the number of neurons in the hidden layer while Reed [19] reduces the number of the connections between the neurons from a large a network and removes such connections which seem to have no significant effect on the network's functioning. In [20] Xu and Chen propose an approach for determining an optimal number of the hidden layer neurons for MLPs starting from previous work by Barron [21], who reports that, using MLPs for function approximation, the rooted mean squared (RMS) error between the well-trained neural network and a target function f is bounded by

$$O\left(C_f^2 / m_I\right) + O\left((m_I m_O / N) \log N\right) \tag{3}$$

where N is the number of training pairs, and C_f is the first absolute moment of the Fourier-magnitude distribution of the target function f. Barron then mathematically proves that, with $m_I \sim C_f (N/(m_O \log N))^{1/2}$ nodes, the order of the bound on the RMS error is optimized to be $O(C_f ((m_O /N) \log N)^{1/2})$. We can then conclude that if the

target function *f* is known then the best (leading to a minimum RMS error) value of m_I is

$$m_I = C_f\,(N/(m_O \log N))^{1/2} \tag{4}$$

However, even though *f* were assumed unknown, from the UAT, we know it may be approached with $\varepsilon \rightarrow 0$. In this case, Xu and Chen [20] use a complexity regularization approach to determine the constant *C* in

$$m_I = C\,(N/(m_O \log N))^{1/2} \tag{5}$$

by trying an increasing sequence of *C* to obtain different values of m_I, train a MLP for each m_I and then observe the m_I which generates the smallest RMS error (and note the value of the *C*). Notice that C_f depends on an unknown target function *f*, whereas *C* is a constant which does not.

2.1 Statistical estimation of m_I's Lower Value

Instead of performing a costly series of case-by-case trial and error tests to target on the value of m_I as in [20] our aim is to obtain an algebraic expression yielding m_I's lower bound from

$$m_I = \sum_{i=0}^{d1} \sum_{j=0}^{d2} K_{ij} m_O^i N^j \tag{6}$$

where d_1 and d_2 are selected a priori and the K_{ij} are to be adequately determined for a predefined range of values of m_O and *N*. That is, we aim at having a simple algebraic expression which will allow us to establish the minimum architecture of a NN given the number of input variables and the number of training pairs. The values of m_I depend on *N*, m_O and *C*. Even though are several possible values of *C* for every pair (m_O, *N*) an appropriate value of the lower bound value of *C* may be set by considering values for *C* in a plausible range and calculating the mean (μ_C) and standard deviation (σ_C) for every pair (m_O, *N*). The upper value of the range of interest is given by the fact that the maximum of m_I is N/m_O;. the lower value of the range is, simply, 1. Thus, we may find a statistically significant lower value of *C (*denoted by C_{min}) and the corresponding lower value of m_I from Chebyshev's theorem [22] which says that

$$P(\mu_C - k\,\sigma_C \le C \le \mu_C + k\,\sigma_C) > 1 - 1/k^2 \tag{7}$$

and makes no assumption on the form of the *pdf* of *C*. If, however, we consider that *C*'s *pdf* is symmetric we have that $P(C > \mu_C - \sqrt{5}\,\sigma_C) \ge 0.9$. A very reliable and general value of C_{min} is, therefore, given by

$$C_{min} = \mu_C - \sqrt{5}\,\sigma_C \tag{8}$$

Now we propose to analyze all combinations of m_O and *N* in a range of interest and obtain the best regressive polynomial from these combinations. We considered the range of m_O between 2 and 82 (i.e. up to 82 input variables); likewise, we considered

N between 100 and 25,000. That is we consider up to 25,000 *effective* objects in the training set. By *effective* we mean that they correspond to a sample devoid of unnecessarily redundant data, as will be discussed in the sequel. The number of combinations in this range is 81 x 25,000 (or 2,025,000 triples), which would have demanded us to find an algebraic expression for these many objects. To reduce the data set to more manageable proportions we sampled the combinations of m_O and N by increasing the values of m_O in steps of 5, while increasing those of N in steps of 100. The number of objects in the sample reduced to 4,250. The general procedure is as follows:

(a) Select lower and upper experimental values of m_O and N which we denote with m_L, N_L, m_H and N_H. These are set to 2, 100, 82 and 25,000, respectively.

(b) Define step sizes D_m and D_N (these were set to 5 and 100, respectively) which determine the values between two consecutive (m_O, N) pairs. That is, m_O will take consecutive values m_L, m_L+D_m, , . . . , m_H. Likewise, N will take consecutive values N_L, N_L+D_N,…, N_H.

(c) Obtain the values for all combinations of m_O and N in the range between (m_L, N_L) and (m_H ,N_H), i.e. (m_L,N_L), (m_L,N_L+D_N),…,(m_H,N_H).

(d) For every pair (m_O, N) calculate μ_C and σ_C. Then obtain C_{min} from equation (8).

(e) For every pair (m_O, N) obtain $m_I = C_{min}\ (N/(m_O \log N))^{1/2}$.

The maximum and minimum values for (m_O, N, m_I) are shown in Table 1.

	m_O	N	m_I
Max	82	25,000	549
Min	2	100	0

Table 1. Values of (m_O,N,m_I) in the range of interest.

(f) Store every triple (m_O, N, m_I) in a table T.

Once steps (c) to (f) have been taken, we have spanned the combinations of all triples (m_O, N, m_I) in the interval of interest.

(f) From T get a numerical approximation as per equation (6), which will be described in what follows.

The triples (m_O, N, m_I) were mapped into the interval [0,1]. The corresponding scaled values are denoted with (m_O*, N*, m_I*).

Therefore,

$m_O^* = (m_O - m_L)/(m_H - m_L)$ → $m^* = (m_O - 2)/80$

$N^* = (N - N_L)/(N_H - N_L)$ → $N^* = (N - 100)/24900$

$m_I^* = (m_I - m_{min})/(m_{max} - m_{min})$ → $m_I^* = m_I / 549$

From the 4,250 scaled vectors we obtained the purported polynomial expression $m_I^* = f(m_O^*, N^*)$ with $d_1 = d_2 = 7$, thus:

$$m_I^* = \sum_{i=0}^{7} \sum_{j=0}^{7} \vartheta_{ij} K_{ij} (m_O^*)^i (N^*)^j \tag{9}$$

$$|\vartheta| = 12 \tag{10}$$

Where K_{ij} denotes a coefficient and ϑ_{ij} is an associated constant which can take only the values 0 or 1. Only 12 of the possible 64 combinations of *(i, j)* are allowed. This number was arrived at by trying several different values and calculating the associated RMS error, as shown in table 2.

Terms	7	8	9	10	11	12	13
RMS	0.07104	0.03898	0.06574	0.05889	0.04618	0.03410	0.03885
Terms	14	15	16	17	18	19	20
RMS	0.03190	0.03712	0.02718	0.02500	0.01405	0.03068	0.02345

Table 2. Best values of RMS for different number of terms.

The best RMS error corresponds to $|\vartheta| = 18$; however, the simpler approximation when $|\vartheta| = 12$ is only marginally inferior (2%) and, for simplicity, we decided to remain with it. The final 12 coefficients are shown in table 3.

K01	K12	K13	K15	K22	K24	K32	K35	K42	K45	K52	K63
0.9307	-33.6966	24.5008	-3.4958	107.4970	-41.0848	-209.3930	52.8180	205.6786	-32.6766	-80.8771	9.7786

Table 3. Coefficients for $|\vartheta| = 12$.

The subindices denote the powers of the associated variables. Therefore, we have that

$$m_I^* = K_{01}N^* + K_{12}mO^*(N^*)^2 + K_{13}mO^*(N^*)^3 + K_{15}mO^*(N^*)^5 + K_{22}(mO^*)^2(N^*)^2 + K_{24}(mO^*)^2(N^*)^4 + K_{32}(mO^*)^3(N^*)^2 + K_{35}(mO^*)^3(N^*)^5 + K_{42}(mO^*)^4(N^*)^2 + K_{45}(mO^*)^4(N^*)^5 + K_{63}(mO^*)^6(N^*)^3 \quad (12)$$

Finally,

$$m_I = (m_L - m_H)/m_I^* + m_L \quad \rightarrow \quad m_I = 549m^*_I \quad (13)$$

According to these results $K_{ij}=0$ except for the combinations of *(i,j)* shown in table 3. Two views of equation (12) are shown in Figure 4.

How were the coefficients of (12) found? As follows: we define a priori the number $|\vartheta|$ of desired monomials of the approximant and then select which of the p possible ones these will be. There are $C(p, |\vartheta|)$ combinations of monomials and even for modest values of p and $|\vartheta|$ an exhaustive search is out of the question. This is an optimization problem which we tackled using the genetic algorithm (EGA) discussed in [23] [24].

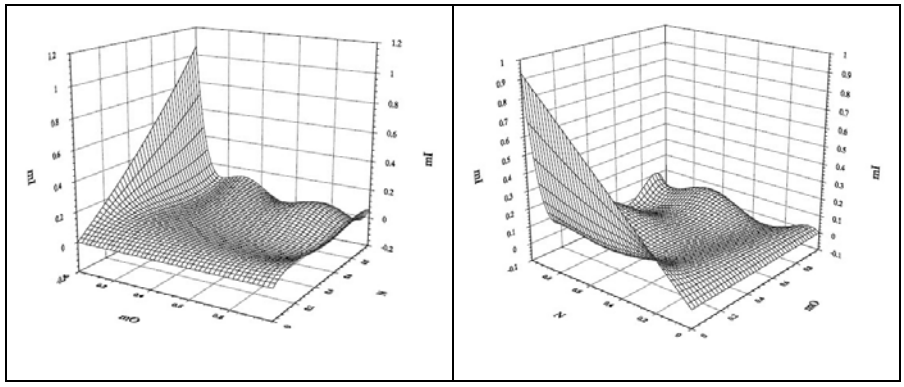

Fig. 4. Graphical representations of equation (13)

2.1.1 Determination of the Coefficients of the Approximant

Consider an approximant of the form

$$f(v_1,\ldots v_n) = \sum_{i_1=0}^{d_1} \ldots \sum_{i_n=0}^{d_n} C_{i_1 \ldots i_n} {v_1}^{i_1} \ldots {v_n}^{i_n} \qquad (14)$$

There are $p = (d_1+1)x\ldots x(d_n+1)$ possible monomials. We associate to this polynomial a chromosome which is a binary string of size p. Every bit represents a monomial ordered as per the sequence of the consecutive powers of the variables. If the bit is '1' it means that the corresponding monomial is retained while if it is a '0' it is discarded. One has to ensure that the number of 1's is equal to $|\vartheta|$. Because of this, the population of the EGA consists of a set of binary strings of length p having only $|\vartheta|$ 1's. For example, if $y=f(v_1, v_2, v_3)$ and $d_1=1$, $d_2=d_3=2$ the powers assigned to the $2\times 3\times 3 = 18$ positions of the genome are 000, 001, 002, 010, 011, 012, 020, 021, 022, 100, 101, 102, 110, 111, 112, 120, 121, 122. Taking $|\vartheta|=6$ the chromosome 110000101010000001 would correspond to $P(v_1,v_2,v_3) = k_{000}+ k_{001}v_3+ k_{020}v_2^2+ k_{022}v_2^2v_3^2+ k_{101}v_1v_3+ k_{122}v_1v_2^2v_3^2$。For every genome the monomials (corresponding to the 1's) are determined by EGA. Then the so-called Ascent Algorithm ("AA"; discussed at length in [25]) is applied to every individual's polynomial's form and yields the set of $|\vartheta|$ coefficients which minimize $\varepsilon_{MAX} = max(|f_i - y_i|) \ \forall i$. For this set of coefficients the mean squared error ε_{RMS} is calculated. This is the fitness function of EGA. The EGA's 25 individuals are selected, crossed over and mutated for 200 generations. In the end, we retain the individual whose coefficients minimize ε_{RMS} out of those which best minimize ε_{MAX} (from the AA). From this procedure we derived the coefficients of table 3.

2.2 Considerations on the Size of the Training Data

We have successfully derived an algebraic formulation of the most probable lower bound of m_I as a function of m_O and N. The issue we want to discuss here is how to determine the effective size of the training data N so as to find the best architecture.

Intuitively, the patterns that are present in the data and which the MLP "remembers" once it has been trained are stored in the connections. In principle we would like to have enough connections for the MLP to have sufficient storage space for the discovered patterns; but not too much lest the network "memorize" the data and lose its generalization capability. It is for this obvious relation between the learning capability of the MLP and the size of the training data that equation (13) was formulated. A formalization of these concepts was developed by Vapnik and co-workers [26] and forms the basis of what is now known as statistical learning theory. Here we wish to stress the fact that the formula of (13) tacitly (but now we make it explicit) assumes that the data is rich in information. By this we mean that it has been expressed in the most possible compact form.

Interestingly, none of the references we surveyed ([11], [12], [13], [14], [15, 16], [17], [18], [19], [20], [21]) makes an explicit consideration of the role that the information in the data plays when determining m_I. In the SVM paradigm, for example, this issue is considered when determining the number of support vectors and the so-called regularization parameter which reflects a tradeoff between the performance of the trained SVM and its allowed level of misclassification [10]. We know that the number of weights (connections) $| w |$ in the MLP directly depends on the number of hidden neurons, thus

$$| w | = m_O m_I + 2m_I + 1 \tag{15}$$

But it is easy to see that even large amounts of data may be poor in information. The true amount of information in a data set is exactly expressible by the Kolmogorov Complexity (KC) which corresponds to the most compact representation of the set under scrutiny. Unfortunately, the KC is known to be incomputable [27]. Given this we have chosen the PPM (Prediction by Partial Matching) algorithm [28] as our compression standard because it is considered to be the state of the art in lossless data compression; i.e. the best practical approximation to KC. Therefore, we will assume that the training data has been properly expressed by first finding its most compact form. Once having done so, we are able to estimate the effective value of N. Otherwise, (13) may yield unnecessarily high values for m_I .

To illustrate this fact consider the file F1 comprised of 5,000 equal records consisting of the next three values: "3.14159 <tab> 2.71828 <tab> 1.61803" separated by the ASCII codes for <cr><lf>. That is, $m_O=3$; $N=5{,}000$. This yields 125,000 bytes in 5,000 objects. A naïve approach would lead us to solve equation (13) yielding $m_I = 97$. However, when compressed with the PPM2 (PPM algorithm of order 2) the same data may be expressed with 49 bytes, for a compression ratio of 2,551:1. That is, every bit in the original file conveys less than 1/2,551 (i.e. ≈.00039) bits of information. Thus the true value of N is, roughly, 2; for which $m_I=1$. On the other hand, a file F2 consisting of 5,000 lines of randomly generated bytes (the same number of bytes as the preceding example), when compressed with the same PPM2 algorithm yields a compressed file of 123,038 bytes; a 1:1.02 ratio. Hence, from (13), $m_I = 97$. There-

fore, for (13) to be applied it is important to, first, estimate the effective (in the sense that it is rich in information) size of N.

3. Experiments

Now we want to ascertain that the values obtained from (13) do indeed correspond to the lowest number of needed neurons in the hidden layer of a MLP. To exemplify this we analyze three data sets. Two of them are from UCI's repository. The third regards the determination of a MLP approximating the data of Figure 3. For every one we determine the value of m_I and show that it is the one resulting in the most efficient architecture.

Problem 1 [29] is a regression problem with $m_O=6$, $N=209$. In Figure 5 we show the learning curves using $m_I=2$ and $m_I=3$. The value of m_I from eq. (13) is 3. If we use a smaller m_I the RMS error is 4 times larger and the maximum absolute error is 6 times larger.

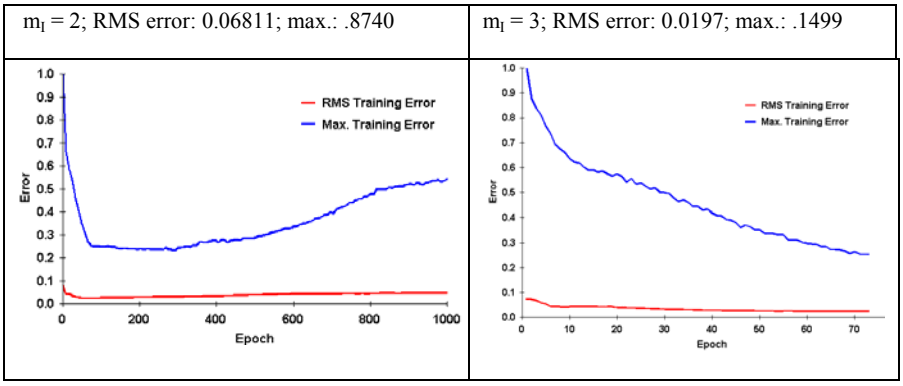

Fig. 5. Learning curve for problem 1 (m_I=2 and m_I=3)

Problem 2 [30] is a classification problem with $m_O=13$, $N=168$. The learning curves using $m_I=1$ and $m_I=2$ are shown in Figure 6. It is trivial to transform a classification problem into a regression one by assigning like values of the dependent variable to every class. In this case the classes 1, 2 and 3 were identified by the scaled values 0, 0.5 and 1. Therefore, a maximum absolute error (MAE) smaller than 0.25 is enough to guarantee that all classes will be successfully identified. The value of m_I from eq. (13) is 2. If we use a smaller m_I the MAE is 0.6154. If we use $m_I=2$ the MAE is 0.2289. The case where MAE>0.25 ($m_I=1$) and MAE<0.25 ($m_I=2$) are illustrated in Figure 7, where horizontal lines correspond to the 3 classes. As shown, these were poorly identified when $m_I=1$. The case $m_I=2$ leads to correct identification of the classes and 100% classification accuracy.

Problem 3 has to do with the approximation of the 4,250 triples (m_O, N, m_I) from which equation (12) was derived (see Figure 4). We used it to determine the architecture of the best MLP which approximates these data. PPM2 compression finds a 4:1 ratio between raw and compressed data. Hence, the effective value of N is 1,060,

for which m_I = 20. Training the MLP for 20 hidden neurons (HN) yields a maximum absolute error of 0.02943 and an RMS error of 0.002163; doing it for 19 HNs yields larger corresponding errors of 0.03975 and 0.002493. If we go on to 21 HNs we get 0.03527 and 0.002488. In other words, "20" corresponds to the lowest effective number of HNs for this problem.

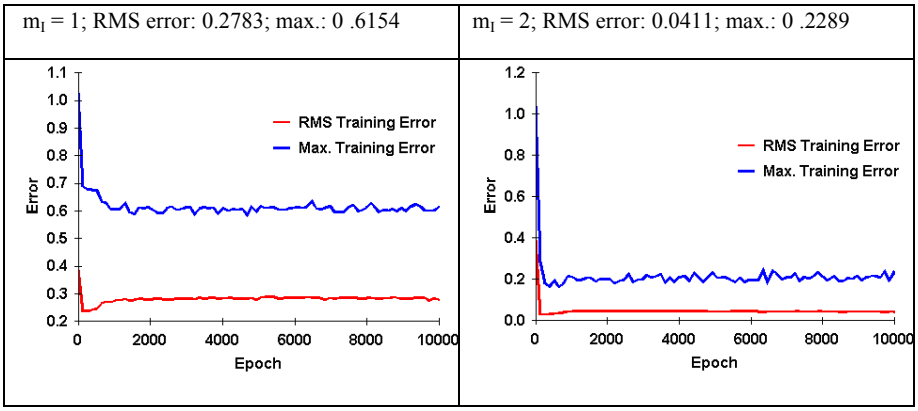

Fig. 6. Learning curve for problem 2 (m_I=1 and m_I=2)

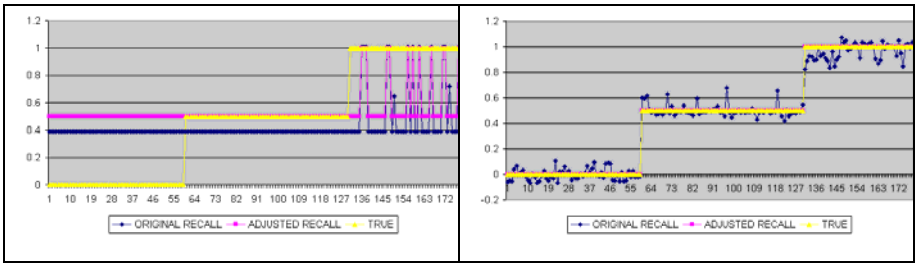

Fig. 7. Effective classification in problem 2 (m_I=1 and m_I=2)

4. Conclusions

We discussed the problem of finding the adequate number of neurons (m_I) in the hidden layer of a MLP network. We argued that MLPs offer advantages over alternative paradigms if data is continuous and scaled into [0, 1]. We have seen that approximate continuity of the training data is enough to render more than one hidden layer unnecessary and that such characteristic may be attained in practice by using natural splines to enrich the data. Hence, the conclusions derived from the UAT are directly applicable to any MLP network. Furthermore, we pointed out that the correct assessment of m_I depends on the determination of the effective size of the training data (N). The actual algorithmic information of N is incomputable but it may be

approximated by considering the data after PPM2 compaction. Thus, we have shown that it is possible to determine a lower practical bound on the number of neurons in the hidden layer (m_I) of a MLP with only one such layer. We also showed how to obtain a closed and compact algebraic expression from the partial enumeration of the possible values of (m_O, N and m_I) based on the theoretical work of Barron [21]. To do this, we used a GA which selects the elements of an approximation polynomial. From experimental runs we determined that no more than 12 terms are needed for an adequate RMS error ($\approx 3.40\%$). The approximation polynomial directly delivers the smallest expected value of m_I with 90% reliability in the range ($2 \leq m_O \leq 82$) and ($100 \leq N \leq 25,000$). Finally, we offered a few experimental examples in which the correctness of the lower bound on m_I is clearly exhibited. The purpose of the experiments was to offer a practical illustration of the conclusions drawn from theoretical considerations. Similar results will be obtained for any data set which complies with the stipulated pre-conditioning. We finish our conclusions by pointing out that the purported algebraic expression $m_I = f(m_O, N)$ might have been replaced (as shown in experiment 3) by a properly trained MLP. The corresponding minimum MLP has 81 connections as opposed to only 12 coefficients of equation (12). An explicit algebraic expression is to be preferred if, as shown, it is accurate enough. As opposed to the "black box" nature of a MLP, it allows us to explore the relation between the input variables (m_O and N) and the dependent variable m_I. For instance, a combination of variables m_O*, N* of degree 9 [$(mO^*)^6(N^*)^3$] is enough for our approximation. Therefore, by allowing a direct determination of m_I the only practical inconvenience of the MLP paradigm has been superseded and the best architecture is reachable without the need to resort to heuristics.

References

[1] Minsky, Marvin L., and Seymour A. Papert. Perceptrons - Expanded Edition: An Introduction to Computational Geometry. Boston, MA:: MIT press, 1987.

[2] Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." Neural networks 2.5 (1989): 359-366.

[3] Hecht-Nielsen, Robert. "Theory of the backpropagation neural network." Neural Networks, 1989. IJCNN., International Joint Conference on. IEEE, 1989.

[4] Cybenko, George. "Approximation by superpositions of a sigmoidal function." Mathematics of control, signals and systems 2.4 (1989): 303-314.

[5] Neural Networks, A Comprehensive Foundation, Second Edition, Ch. 4, p. 294, Notes and References 8, Prentice Hall International, 1999.

[6] Huang, Guang-Bin. Learning capability and storage capacity of two-hidden-layer feedforward networks. Neural Networks, IEEE Trans. on, 2003, vol. 14, no 2, p. 274-281.

[7] Shampine, Lawrence F., and Richard C. Allen. Numerical computing: an introduction. Ch. 1.3, pp. 54-62. Harcourt Brace College Publishers, 1973.

[8] Buhmann, Martin D., "Radial basis functions." Acta Numerica 2000 9 (2000): 1-38.

[9] Hearst, M. A., Dumais, S. T., Osman, E., Platt, J., & Scholkopf, B. (1998). Support vector machines. Intelligent Systems and their Applications, IEEE, 13(4), 18-28.

[10] Haykin, Simon S., et al. Neural networks and learning machines. Vol. 3. Upper Saddle River: Pearson Education, 2009.

[11] Ash T., 1989, Dynamic Node Creation In Backpropagation Networks, Connection Science, Volume 1, Issue 4, pp 365 – 375.
[12] Hirose Y. Yamashita I.C., Hijiya S., 1991, Back-Propagation Algorithm Which Varies the number of hidden units, Neural Networks, Vo1.4. 1991.
[13] Rivals, I., Personnaz, L., 2000, A statistical procedure for determining the optimal number of hidden neurons of a neural model, Second International Symposium on Neural Computation (NC'2000), Berlin, May 23-26 2000.
[14] Yao, Xin. Evolving Artificial neural networks. Proceedings of the IEEE, 1999, vol. 87, no 9, p. 1423-1447.
[15] Xu, L., 1995. Ying-Yang Machine: A Bayesian- Kullback scheme for unified learnings and new results on vector quantization. Keynote talk, Proceedings of International Conference on Neural Information Processing (ICONIP95), Oct. 30 - NOV. 3, 977 – 988.
[16] Xu, L., 1997. Bayesian Ying-Yang System and Theory as A Unified Statistical Learning Approach: (III) Models and Algorithms for Dependence Reduction, Data Dimension Reduction, ICA and Supervised Learning. Lecture Notes in Computer Science: Proc. Of International Workshop on Theoretical Aspects of Neural Computation, May 26-28, 1997, Hong Kong, Springer-Verlag, pp. 43-60.
[17] Fletcher, L. Katkovnik, V., Steffens, F.E., Engelbrecht, A.P., 1998, Optimizing The Number Of Hidden Nodes Of A Feedforward Artificial Neural Network, Proc. of the IEEE International Joint Conference on Neural Networks, Volume: 2, pp. 1608-1612.
[18] Fahlman, S.E. An empirical study of learning speed in back propagation networks. Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufman 1988.
[19] Reed, R.: Pruning Algorithms A Survey. IEEE Trans. On Neural Networks, Volume 4, Number 5, 1993, pp. 707-740.
[20] Xu, Shuxiang; Chen, Ling. A novel approach for determining the optimal number of hidden layer neurons for FNN's and its application in data mining. International Conference on Information Technology and Applications: iCITA. 2008. p. 683-686.
[21] Barron, A. R., (1994), Approximation and Estimation Bounds for Artificial Neural Networks, Machine Learning, (14): 115-133, 1994.
[22] Saw, John G.; Yang, Mark Ck; Mo, Tse Chin. Chebyshev inequality with estimated mean and variance. The American Statistician, 1984, vol. 38, no 2, p. 130-132.
[23] Kuri-Morales, Angel, and Edwin Aldana-Bobadilla. "The Best Genetic Algorithm I." Advances in Soft Computing and Its Applications. Springer Berlin Heidelberg, 2013. 1-15.
[24] Kuri-Morales, Angel Fernando, Edwin Aldana-Bobadilla, and Ignacio López-Peña. "The Best Genetic Algorithm II." Advances in Soft Computing and Its Applications. Springer Berlin Heidelberg, 2013. 16-29.
[25] Cheney, Elliott Ward. "Introduction to approximation theory.", Ch. 2, pp. 45-51, (1966).
[26] Vapnik, Vladimir. The nature of statistical learning theory. Springer, 2000.
[27] Li M and Vitányi P, An introduction to Kolmogorov complexity and its applications, Springer Verlag, New York, 2nd Ed, 1997.
[28] Teahan, W. J. "Probability estimation for PPM." In Proceedings NZCSRSC'95. http://www. cs. waikato. ac. nz/wjt. 1995.
[29] Phillip Ein-Dor and Jacob Feldmesser, Ein-Dor, "Computer Hardware Data Set" : Faculty of Management, Ramat-Aviv; https://archive.ics.uci.edu/ml/datasets/Computer+Hardware
[30] Forina, M. et al, "Wine Data Set", PARVUS, Via Brigata Salerno, htps://archive.ics.uci.edu/ml/datasets/Wine