J. Phys. A: Math. Gen. 25 (1992) 1903-1914. Printed in the UK

Properties of feedforward neural networks

Marco Budinich and Edoardo Milotti

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

Received 1 May 1991

Abstract. In his seminal paper Cover used geometrical arguments to compute the probability of separating two sets of patterns with a perceptron. We extend these ideas to feedforward networks with hidden layers. There are intrinsic limitations to the number of patterns that a net of this kind can separate and we find quantitative bounds valid for any net with d input and h hidden neurons.

1. Introduction

Feedforward neural networks have become one of the popular paradigms in the rapidly expanding field of neural models: a sufficiently complex feedforward network is capable of carrying out complex pattern recognition tasks using only very elementary processing units ('neurons'), and there are 'programming algorithms' that resemble—more or less closely—a learning process.

The simplest such network is the perceptron (figure 1), first devised by Rosenblatt [2]. A perception is made of a 'layer' of input units and of a processing unit. The processing unit computes a weighted sum of the states of the input units and outputs 1 if the sum exceeds a given threshold value and 0 if it does not (or vice versa). In this way each input state is labelled by 0 or 1, and the perceptron acts as a simple pattern classifier.

In 1964 Cover [1] studied the perceptron in a geometrical setting and gave estimates of its efficiency. Minsky and Papert [3] showed that the perceptron is actually a rather poor classifier.

However, the negative conclusions of Minsky and Papert cannot be extended to feedforward networks, and many workers have successfully programmed feedforward networks to solve some rather complex logical and pattern recognition problems.

In the next section we start with a brief review of the results of Cover. Then, following the same guidelines, we calculate an upperbound for the number of mappings

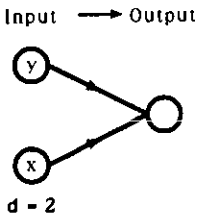

Figure 1. A simple perceptron.

0305-4470/92/071903+12$04.50

that can be implemented by a feedforward neural network with hidden layers and we show that there exists a maximum number of patterns that a given network architecture can separate without errors. Our results extend and generalize those of Mitchison and Durbin [4] obtained with a similar approach. In the last section we give some numerical results and simple rules for good network design.

2. The perceptron

It is well known that the input patterns of a perceptron with d input neurons can be thought of as points in d-dimensional space and that in this frame the action of the output neuron corresponds to separating these points with an oriented hyperplane. We take perceptrons with d input neurons and n input patterns ($n \leqslant 2^d$ since, without loss of generality, we use digital neurons).

Consider now the possible input patterns of the perceptron with $d = 2$ input units (figure 2): they are the $2^d = 4$ vertices of a square. Different perceptron functions correspond to different ways of drawing a straight line among them. Since each straight line corresponds to two threshold functions and there are seven different ways to draw a line, there are 14 different perceptron functions†.

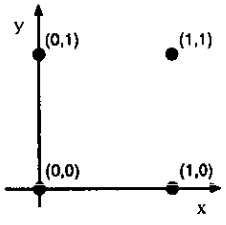

Figure 2. Space of $d = 2$ perceptron.

On the other hand $n = 4$ objects can be partitioned in $2^n = 16$ ways. This means that there are $16 - 14 = 2$ functions that this perceptron cannot compute, and it is well known that they are the XOR and its complement.

This counting argument was used by Cover [1], to find general results for perceptrons with d neurons and n input patterns.

Let $C(n, d)$ be the number of ways in which one can put an oriented hyperplane between n points in general position in d-dimensional space. $C(n, d)$ has been derived several times by different authors (see [1, 5] and references therein) and is given by

$$C(n, d) = 2 \sum_{k=0}^{d} \binom{n-1}{k}$$

† In this paper we treat the terms 'threshold function', 'linear partition' and 'half-space' almost as synonyms. If s_i is the state of the ith input neuron and w_i, θ are given constants, then the inequality $\Sigma_i w_i s_i > \theta$ separates the input space in two parts (the 'half-spaces'). The same concept can be rephrased in terms of the step function of the argument $(\Sigma_i w_i s_i - \theta)$—and this is what we call the 'threshold function'. The value of the step function in each half-space 'labels' that half-space, and therefore reversing the inequality amounts to an exchange of the half-space labels. Notice also that since the points in the input space belong to a discrete lattice, then threshold functions are grouped in equivalence classes of functions that perform the same action on the lattice points. In what follows we do not distinguish between a threshold function and the equivalence class to which it belongs.

then

$$C(n, d) = 2^n \qquad \text{if } n \leq d+1$$
$$C(n, d) = O(n^d) \qquad \text{for } n \gg d. \tag{1}$$

Each of the $C(n, d)$ ways to put an oriented hyperplane between the n points corresponds to a different perceptron function. Cover compared this number with the total number of partitions of the n points, i.e. 2^n, to obtain the probability $P(n, d)$ of separating with a perceptron two subsets of a set of n random points in d-space†

$$P(n, d) = \frac{\text{number of linear partitions}}{\text{total number partitions}} = \frac{2\sum_{k=0}^{d}\binom{n-1}{k}}{2^n} \tag{2}$$

3. Feedforward networks

The case of the perceptron is actually quite simple: there is just one separating hyperplane, and all the points that fall on one side of this hyperplane are labelled the same way and it is easy to tell whether this labelling is just what we want and if the network behaves as a good classifier or not.

The perceptron can be easily generalized by introducing more layers, with neurons that behave individually like simple perceptrons. When there is no feedback loop these networks are called 'feedforward networks'. Also, without loss of generality, we assume that each layer of neurons is connected only to the adjacent layers.

Feedforward networks with at least one hidden layer are much more complex than perceptrons: now there are more labellings—i.e. those provided by the hidden layers—and this intermediate pattern classification may lead to a loss of information.

In order to clarify what we mean, consider a network with just one layer of h hidden neurons (figure 3).

Any neuron of the hidden layer is a perceptron and so for any of them we can have $C(n, d)$ different functions.

Therefore the total number of mappings of the input layer to the hidden layer is $C(n, d)^h$.

Not all mappings are worthy of consideration: take for example the extreme case in which all hidden neurons output 0 in response to every input pattern. The output

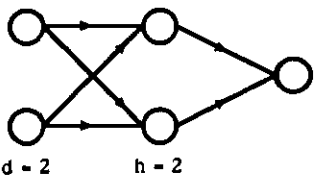

Figure 3. A net with one hidden layer.

† This result is strictly valid only when the n points are in general position in d-space (i.e. if and only if any k-tuple ($k \leq d+1$) of them is linearly independent) but it has been shown that the results apply also to digital neurons even though hypercube vertices are not always in general position [6].

layer does not receive any information from the hidden layer and cannot classify the input patterns.

To proceed further in this direction, we start by defining the concepts of 'problem', 'network function' and 'solution'. Consider a network with d input neurons and n input patterns, an unspecified number of hidden layers (even no hidden layers at all, i.e. a perceptron), and a single output neuron, then:

(i) A *problem* is one of the 2^n different ways of partitioning the set of n input patterns in two subsets.

(ii) A *network function* is specified by the set of the threshold functions associated with all the neurons after the input layer. Two network functions are different when the associated sets of threshold functions are different. Just as it did for the perceptron, a network function assigns a label to each input pattern. However, in a multilayer net, different network functions can return the same labelling of the input patterns. The set of all the labellings generated by all the possible network functions is a subset of the set of problems: we will call it the set of soluble problems.

(iii) A *solution* is any network function that partitions the set of input patterns as in the original problem.

Figures 4, 5 and 6 illustrate these concepts graphically for a net like that of figure 3. The patterns are represented as small circles, and the axes are shown only to remind us that the patterns are actually points in a d-dimensional space.

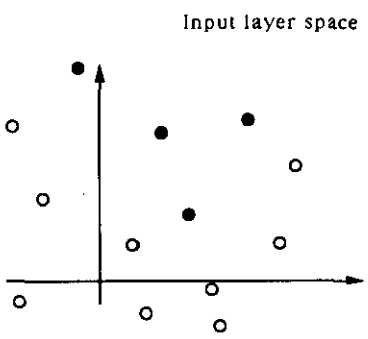

Figure 4. A problem in input space.

Figure 5 illustrates the concept of network function: the upper part part shows the input space and the two broken lines are the separating hyperplanes in this space corresponding to the threshold functions of two hidden neurons. This choice partitions the input space: all the patterns contained in the regions marked 00 to 11 in input space are mapped to the corresponding patterns marked 00 to 11 in the hidden layer space depicted in the lower graph. The hidden layer space is in turn partitioned by the output neuron threshold function, and each input pattern is thus labelled in this final pass.

Finally in figure 6 we see that the network function of figure 5 is actually a solution for the particular problem of figure 4, since all the 'black' patterns are assigned the label '0' and all the 'white' patterns are assigned the label '1'.

The same problem may admit different solutions, e.g. the network function obtained replacing the threshold function 'x' with that marked 'x''' in figure 6 is also a solution. This is not true for the perceptron where, if a solution exists, then it is unique (to be convinced simply look at the perceptron solution realized in the two lower frames of figure 6: any change of the separating line no longer gives the exact solution).

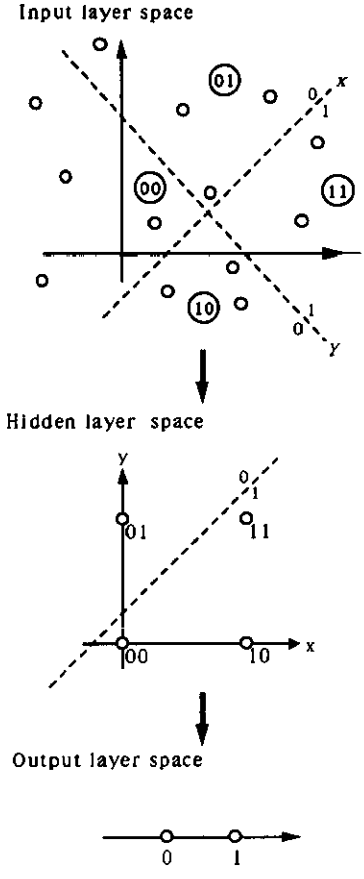

Figure 5. A network function for $d=2$, $h=2$.

With these definitions the probability P_S that a randomly chosen problem has a solution is

$$P_s = \frac{\text{number of soluble problems}}{\text{number of problems}} = \frac{\text{number of soluble problems}}{2^n}.$$

This is a number that we do not know how to compute. However, the total number of network functions is at least as great as the number of soluble problems, and we calculate the upper bound,

$$P_S = \frac{\text{number of soluble problems}}{2^n} \leqslant \frac{\text{number of network functions}}{2^n} \tag{3}$$

where the equality holds in the perceptron case for which the number of soluble problems is equal to the number of network functions given by (2).

Let us see how the number of network functions can be calculated exactly in the general case of feedforward networks.

We start by considering a net with an input layer with d neurons connected to a layer of h neurons. n patterns are presented to the network, and any particular network function maps these n d-bit patterns onto m h-bit patterns.

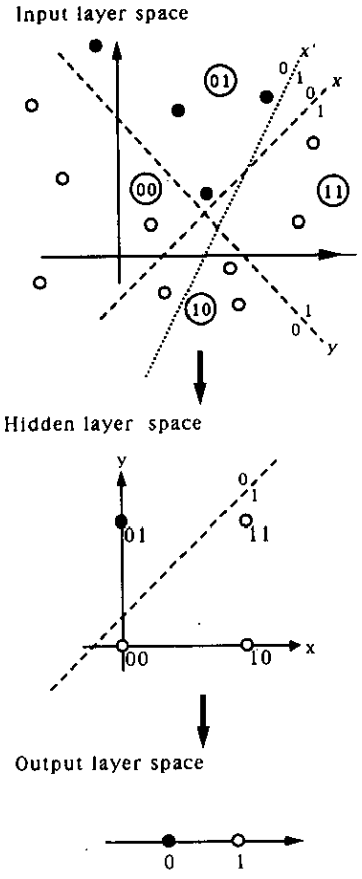

Figure 6. A solution for a a net like that of figure 3.

m may vary between 1 and the maximum value $R(h, d)$, which is the number of regions in which d-space is divided by h hyperplanes in general position.

Let $B_m(n, d, h)$ be the number of network functions that map n patterns in d-space onto exactly m patterns in h-space, then

$$\sum_{m=1}^{R(h,d)} B_m(n, d, h) = C(n, d)^h. \tag{4}$$

It is easy to show (e.g. see [7]) that

$$R(h, d) = \frac{C(h+1, d)}{2} \tag{5}$$

and, therefore, using (1) and assuming that $h \leqslant d$ (a very natural and common choice),

$$R(h, d) = 2^h \qquad \text{if } h \leqslant d. \tag{6}$$

Then, if the output neuron is also taken into account we can calculate exactly the total number of network functions that map the input layer onto the output neuron. For each mapping of the input layer onto the hidden layer there are $C(m, h)$ threshold functions that map the hidden layer onto the output neuron (the hidden layer and the

output neuron form a simple perceptron); therefore, the total number of network functions is

$$\sum_{m=1}^{R(h,d)} B_m(n, d, h)C(m, h). \tag{7}$$

If there are more hidden layers one proceeds similarly, so that for two hidden layers one obtains

$$\sum_{m_1=1}^{R(h_1,d)} \sum_{m_2=1}^{R(h_2,h_1)} B_{m_1}(n, d, h_1)B_{m_2}(m_1, h_1, h_2)C(m_2, h_2).$$

With the approximate values of $B_m(n, d, h)$ derived in the appendix (14) these formulae lead to an explicit estimate of the upper bound (3). However, a tighter bound can be derived by introducing the subset of 'non-mixing' functions.

The concept of the 'mixing' function can be easily understood as follows: take a problem, i.e. partition the set of input patterns into two sets—call them 'black' and 'white'—then, if a network function maps one black and one white pattern onto the same intermediate pattern, some information is lost and an exact solution can no longer be found. This is what we call a mixing function (figure 7). The upper bound (3) can be made tighter if we take into account only the subset of non-mixing network functions:

$$P_S \leqslant \frac{\text{number of non-mixing functions}}{2^n}. \tag{8}$$

If a network function maps the n input patterns onto exactly m hidden layer patterns, there are 2^m ways of assigning the m patterns to one of the two subsets of

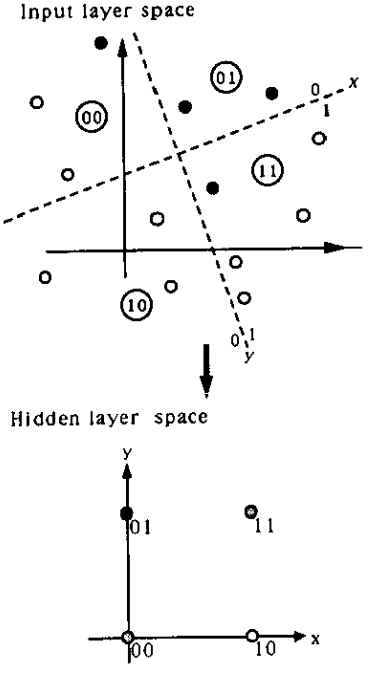

Figure 7. A mixing function for a $d = 2$, $h = 2$ net.

the partition. Since the total number of partitions of the input space is 2^n, the average number of non-mixing functions that map the input layer to the hidden layer is†

$$\frac{1}{2^n}\sum_{m=1}^{R(h,d)} B_m(n,d,h)2^m. \tag{9}$$

Using (4) we can get exact upper and lower bounds for expression (9):

$$\frac{1}{2^n}\sum_{m=1}^{R(h,d)} B_m(n,d,h)2^m \leqslant \frac{2^{R(h,d)}}{2^n}\sum_{m=1}^{R(h,d)} B_m(n,d,h) = \frac{2^{R(h,d)}}{2^n}C(n,d)^h$$

$$\frac{1}{2^n}\sum_{m=1}^{R(h,d)} B_m(n,d,h)2^m \geqslant \frac{1}{2^n}\sum_{m=1}^{R(h,d)} B_m(n,d,h) = \frac{C(n,d)^h}{2^n}$$

so that

$$\frac{C(n,d)^h}{2^n} \leqslant \frac{1}{2^n}\sum_{m=1}^{R(h,d)} B_m(n,d,h)2^m \leqslant 2^{R(h,d)}\frac{C(n,d)^h}{2^n}. \tag{10}$$

From (1) and (5) we find that the bounds (10) approach 0 when $n \gg 1$, so that there is (obviously) an intrinsic limit to the number of patterns n that a single layer can treat without losing information. Similar formulae hold for more layers.

The inequalities (10) give bounds for the average number of non-mixing functions per problem for a layer with d input to h hidden neurons.

Now if we go back to (3) and (8) we see that the probability P_S can also be interpreted as an average number of network functions per problem. In this sense the upper bound in (10) is also an upper bound to the probability of finding a 'good' (non-mixing) function for a d-to-h layer and consequently an upper limit for the probability P_S of any net whose first layer is a d-to-h layer. In other words, whatever the net after layer h, the probability P_S that it can solve a random problem is bounded by

$$P_S \leqslant 2^{R(h,d)}\frac{C(n,d)^h}{2^n}. \tag{11}$$

In the next section we give numerical and approximate solutions for this formula.

Formula (11) resembles superficially the bound obtained by Mitchison and Durbin in [4]. They quote an upper bound $P_S \leqslant C(n,d)^h/2^n$ for a net with d input and h hidden neurons. However, their result is for a net with a fixed output function while (11) applies whatever the mapping that follows the first hidden layer. But if the output function is fixed, the network can solve less problems than a 'free' network, and this is the origin of their lower bound‡.

4. Results and conclusions

It is interesting to choose the network size—i.e. d and h—and compare the behaviour of the approximate average number of non-mixing functions (given in the appendix, see (15)), and of the bounds (10) versus the number of input patterns n.

† We average over all problems, because the number of non-mixing functions is different for different problems. A similar calculation, but with the perhaps more familiar terminology of balls and boxes used in combinatorial problems, is performed in the appendix.

‡ The bound of Mitchison and Durbin can be obtained replacing the factor $C(m,h)$ in (7) with 1 (remember that they fix the output function) thus obtaining, with (4), their total number of functions. This number of functions is then plugged into inequality (3).

We did this for $d = 10$ and $h = 5$ and plotted the logarithms of the results in figure 8. There, the upper full line represents $C(n, d)^h$ and the broken lines are the bounds (10) for a net with $d = 10$ and $h = 5$. The approximate number of non-mixing functions (15) is bracketed between them. Now we wish to pinpoint some of its features.

When n is small the average number of non-mixing functions is a large fraction of the total number of functions $C(n, d)^h$. As n grows, $C(n, d)^h$ changes slope because of the change of regime from 2^n to n^d (see (1)), and at the same time the average number of non-mixing functions reaches a maximum and then approaches 0.

From (10) it is easy to find the approximate position of the maximum $n_{\max}$ (of both bounds) and of the zero-crossing value n_{zero} (of the upper bound)†:

$$n_{\max} \approx \frac{dh}{\log 2} \qquad n_{\text{zero}} \sim 2^h + dh^2 \qquad (\text{if } 2^h \gg dh \log_2 n_{\text{zero}}).$$

This shows that for a random problem for a net with d input and h hidden neurons the probability of finding a solution vanishes if the number of patterns n is larger than $2^h + dh^2$.

Figure 9 contains the same kind of plot as in figure 8, i.e. the approximate number of non-mixing functions done for several values of h ranging from 1 to 6.

The task of finding a non-mixing function becomes increasingly difficult as n grows. This can be seen from the ratio

$$\frac{\text{average number of non-mixing functions}}{C(n, d)^h}$$

plotted in figure 10 for a net with $d = 10$ and various h. This ratio is also the probability of finding a non-mixing function taking a random set of weights.

From (10) we easily obtain bounds for this ratio:

$$\frac{1}{2^n} \leqslant \frac{\text{average number of non-mixing functions}}{C(n, d)^h} \leqslant \frac{2^{R(h,d)}}{2^n}$$

and the upper bound is $\frac{1}{2}$ when $n = 2^h + 1$ (if $h \leqslant d$ and (6) can be used).

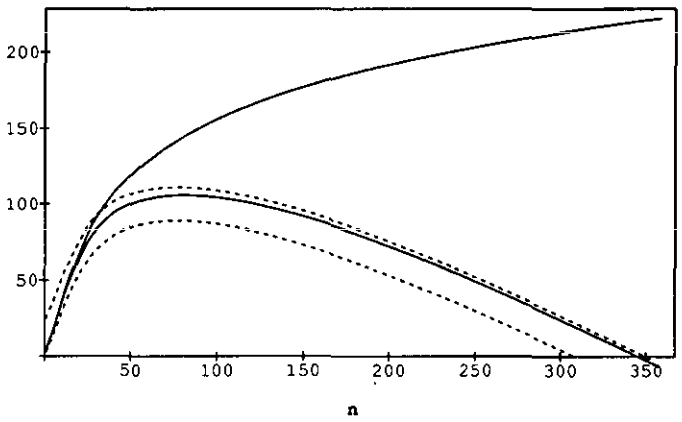

Figure 8. Plot of the logarithm of the number of functions versus n for a net with $d = 10$ and $h = 5$.

† It is sufficient to use the asymptotic value (1) and then take logarithms.

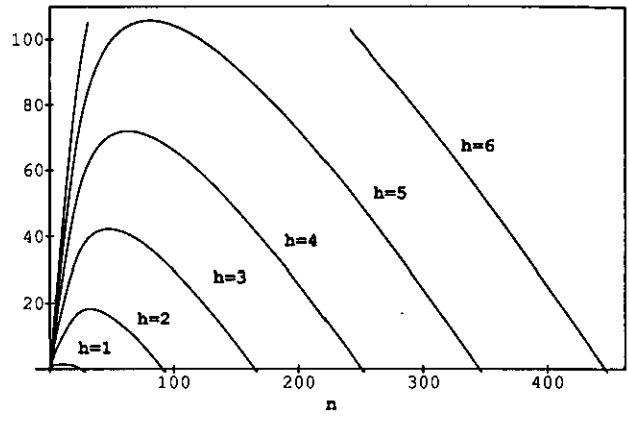

Figure 9. Plot of the logarithm of the number of functions versus n for a net with $d = 10$.

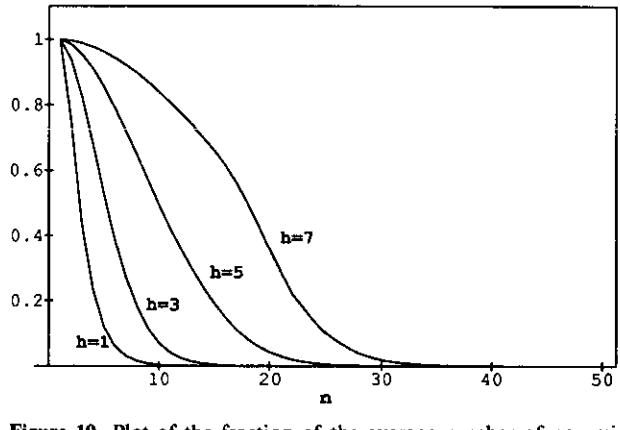

Figure 10. Plot of the fraction of the average number of non-mixing functions versus n for a net with $d = 10$ and various values of h.

How can these results be used to synthesize a working network? We remark that in general for interesting problems d is rather large, and that all $n = 2^d$ patterns may be presented to the network.

Moreover, we give bounds for the probability of finding exact solutions to the original problem. However, it is seldom desired to find an exact solution and more often one is content with an efficient algorithm, i.e. something that gives a reasonable answer with a reasonably high efficiency.

The answer to this dilemma is that the programmer can choose a 'skeleton' of p ($\ll 2^d$) patterns for which the exact answer is desired. Then a first layer is constructed with a number h of hidden neurons such that the probability of finding a non-mixing function is fairly high. This means that the subsequent 'teaching' of this first layer is an 'easy' task (it may be accomplished by back propagation, etc.). The process can then be iterated layer by layer until a complete exact solution for the p patterns is found.

In a way such a strategy resembles the 'tilting algorithm' of Mezard and Nadal [8] and the hope is that this may eventually provide an efficient network. We are planning to explore this and similar algorithms in the near future.

Acknowledgments

We warmly acknowledge stimulating discussions with Luciano Ristori.

Appendix

Here we give an approximation for $B_m(n, d, h)$ as defined in (4).

Take n balls and R_b boxes: there are R_b^n different arrangements of the balls in the boxes. Let $A_m(n)$ be the number of ways in which one can put n balls into exactly m boxes leaving no box empty. Then, since there are

$$\binom{R_b}{m}$$

ways of choosing the m boxes, one finds

$$R_b^n = \sum_{m=1}^{R_b} \binom{R_b}{m} A_m(n) \tag{12}$$

and $A_m(n)$ can be obtained from the inclusion–exclusion principle [9]:

$$A_m(n) = \sum_{k=0}^{m} (-1)^k \binom{m}{k} (m-k)^n.$$

If the balls are painted black and white and we ask what is the number of ways in which we can put the balls in the R_b boxes without mixing balls of different colour in the same box we obviously find a number which is smaller than R_b^n. Now there is a total of $(2R_b)^n$ ways of painting balls and filling boxes: on the other hand there are 2^m ways of choosing the colour of the balls in the m boxes. Therefore, the number of ways in which we can put balls without mixing is

$$\sum_{m=1}^{R_b} \binom{R_b}{m} A_m(n) 2^m \leqslant (2R_b)^n \tag{13}$$

and, if we divide this number by 2^n, we find the average over all the possible colourings:

$$\frac{1}{2^n} \sum_{m=1}^{R_b} \binom{R_b}{m} A_m(n) 2^m \leqslant R_b^n.$$

Notice that (12) and (13) correspond to (4) and (9), the only difference being that while in the earlier problem we were limited to linear functions in transforming d-bit patterns (balls) into h-bit patterns (boxes) no similar restriction exists here. But while we do not know how to calculate $B_m(n, d, h)$ in (9) we can calculate its corresponding term

$$\binom{R_b}{m} A_m(n).$$

This suggests a way to perform an approximation.

The basic idea is that $B_m(n, d, h)$ behaves like

$$\binom{R_b}{m} A_m(n_e),$$

with an 'equivalent number of objects' n_e. To calculate n_e we request that the total number of functions in (4) and (12) be the same when we have the same number of boxes, say $R(h, d)$. This means that n_e must satisfy

$$R(h, d)^{n_e} = C(n, d)^h$$

then by taking logarithms

$$n_e(n, d) = \frac{h \log_2 C(n, d)}{\log_2 R(h, d)}$$

and remembering that under the assumption $h \leq d$ then $R(h, d) = 2^h$ (6) we have

$$n_e(n, d) = \log_2 C(n, d) \qquad \text{if } h \leq d.$$

With this definition of n_e the sums (4) and (12) are equal and we further assume that this equality extends to individual terms, i.e.

$$B_m(n, d, h) \approx \binom{R(h, d)}{m} A_m(n_e(n, d)). \tag{14}$$

With this approximation the number of non-mixing functions (9) becomes

$$\frac{1}{2^n} \sum_{m=1}^{R(h,d)} \binom{R(h, d)}{m} A_m(n_e(n, d)) 2^m \tag{15}$$

and that is the approximation we used for the numerical calculations of section 4. Assumption (14) is further justified by the fact that it is exactly true for low n and h. Indeed, if $n \leq d+1$ and $h \leq d$, then $C(n, d) = 2^n$ and linear functions can implement any partition of the n points.

References

[1] Cover T M 1965 Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition *IEEE Trans. Electronic Computers* **EC14** 326-34

[2] Rosenblatt F 1962 *Principles of Neurodynamics* (New York: Spartan)

[3] Minsky M L and Papert S 1988 *Perceptrons* 3rd edn (Cambridge, MS: MIT) pp. xv-292

[4] Mitchison G J and Durbin R M 1989 Bounds on the learning capacity of some multi-layer networks *Bio. Cyber.* **60** 345-56

[5] Winder R O 1966 Partitions of N-space by hyperplanes *SIAM J. Appl. Math.* **14**(4) 811-18

[6] Budinich M 1991 On linear separability of random subsets of hypercube vertices *J. Phys. A: Math. Gen.* **24** L211-13

[7] Budinich M and Milotti E 1992 Geometrical interpretation of the back-propagation algorithm for the perceptron *Physica* A in press

[8] Mezard M and Nadal J-P 1989 Learning in feedforward layered networks: the tiling algorithm *J. Phys. A: Math. Gen.* **22** 2191-203

[9] Feller W 1970 *An Introduction to Probability Theory and its Application* 3rd edn, vol 1 (New York: Wiley) pp xviii-510