Learning in feedforward layered networks: the tiling algorithm

1989 J. Phys. A: Math. Gen. 22 2191

(http://iopscience.iop.org/0305-4470/22/12/019)

J. Phys. A: Math. Gen. 22 (1989) 2191-2203. Printed in the UK

Learning in feedforward layered networks: the tiling algorithm

Marc Mézard† and Jean-Pierre Nadal‡

† Laboratoire de Physique Théorique, Ecole Normale Supérieure, 24 rue Lhomond, F-75231 Paris Cedex 05, France

‡ Laboratoire de Physique Statistique, Ecole Normale Supérieure, 24 rue Lhomond, F-75231 Paris Cedex 05, France

Received 18 January 1989

Abstract. We propose a new algorithm which builds a feedforward layered network in order to learn any Boolean function of N Boolean units. The number of layers and the number of hidden units in each layer are not prescribed in advance: they are outputs of the algorithm. It is an algorithm for growth of the network, which adds layers, and units inside a layer, at will until convergence. The convergence is guaranteed and numerical tests of this strategy look promising.

1. Introduction

Among the various possible architectures of neural networks, the feedforward layered systems (Rumelhart and McClelland 1986) are relatively simple. There is no feedback and the information is processed from one layer of neurons to the next in parallel. The first layer is the input and the last one the output; the intermediate layers contain the so-called 'hidden units'. In the following we shall use binary neurons (formal neurons) which react through a threshold function to a weighted sum of signals of the neurons in the previous layer. It is known that such a class of networks has a great computational ability. In principle it can implement any function of the N_0 Boolean units in the input layer, provided there is at least one (large enough) intermediate layer.

In using these networks one is immediately confronted with the problem of learning. How does one determine the couplings between neurons in successive layers, in order to build a network which achieves a given task? The general framework is that the network learns through the presentation of examples. A training set of p_0 patterns is presented in the input layer. To each such input ξ^μ, $\mu = 1, \ldots, p_0$, (each ξ^μ is one of the 2^{N_0} configurations of the input layer) one wants to associate an output σ^μ which is known by the operator. For simplicity in the following we shall suppose that the output is binary: there is only one output neuron (the generalisation of our approach to N' output neutrons will be commented upon in the conclusion). Our first problem is to build a network which produces the correct output σ^μ for each input ξ^μ in the training set. This is basically the learning problem which we address in this paper. A second problem which is of interest is that of generalisation: if the mapping $\xi^\mu \to \sigma^\mu$ is a relatively 'smooth' function (this notion will be made more precise in § 3), and if the training set is large enough, one may hope that the network will infer the rule from the examples which it has seen, so that it will associate correct outputs to new inputs which were not present in the training set. This problem will be looked at in our numerical simulations.

0305-4470/89/122191+13$02.50

The problem of learning is relatively simple when there are only two layers, i.e. no hidden units (the perceptron architecture) (Rosenblatt 1962, Minsky and Papert 1969). Unfortunately this architecture can only implement linearly separable functions, so that for more involved tasks one must add hidden units. Then the learning is much harder because the states taken by the hidden units are not fixed *a priori* (Rumelhart and McClelland 1986).

The most widely used strategy is known as back propagation (Le Cun 1985, Rumerlhart *et al* 1986, see also Werbos 1974). For a given network one computes the effective outputs σ'^{μ} and tries to minimise an error (like, e.g., $\Sigma_{\mu}(\sigma'^{\mu}-\sigma^{\mu})^2$) by a gradient descent procedure. Apart from the computer time, the drawbacks are that the structure of the network (number of layers, number of hidden units in each intermediate layer) has to be guessed, one must use analogue neurons even in cases where only digital ones would be necessary, and the minimisation is not guaranteed to converge to an absolute minimum with zero error.

Another strategy introduced recently (Grossman *et al* 1988) uses digital neurons, but always within a structure which is fixed *a priori* with one intermediate layer, and a procedure of trial and error for fixing the values of the hidden units.

In our strategy units are added like tiles whenever they are needed. The first unit of each layer plays a special role: we call it the master unit. During the growth of the network, the master unit of the more recently built layer gives a strictly better approximation of the desired output than the previous one, so that eventually it gives the exact output. Convergence is guaranteed for any problem: the algorithm always produces a finite number of layers. In each layer, once the master unit has been obtained, one checks if it gives the exact output. If not, new 'ancillary' units are added to the layer until this layer gives a 'faithful representation' of the problem: any two patterns with distinct outputs have distinct internal representations. Once this condition is fulfilled, the current layer is considered to have been achieved, and one proceeds to build the master unit of the next layer. Each new unit is generated through a perceptron-type algorithm. The main advantage of this strategy is that the structure of the network is not fixed in advance, but generated dynamically during learning. The network grows until it reaches a size which enables it to implement the desired mapping. As we shall see this is also useful for generalisation.

A recent independent work (Rujan and Marchand 1988) has adopted the same kind of philosophy of adding neurons at will, but in practice the two algorithms are completely different. In Rujan and Marchand (1988) only one intermediate layer of hidden units is generated, and the couplings between the input and hidden layers are found through a kind of exhaustive search procedure inside a restricted set of possible couplings. Our algorithm can generate several layers; the use of the perceptron algorithm instead of an exhaustive search enables us to treat large systems (i.e. problems with a large number of input units).

The paper is organised as follows. In § 2 we describe our tiling algorithm in detail. In § 3 we present results of some numerical simulations on exhaustive learning and generalisation. Perspectives are drawn in § 4.

2. The tiling algorithm

2.1. Basic notions and notation

We consider layered nets, made of binary units which can be in a plus or minus state

(formal neurons). A unit i in the Lth layer is connected to the N_{L-1} units of the preceding layer, and its state $S_i^{(L)}$ is obtained by the threshold rule

$$S_i^{(L)} = \operatorname{sgn}\left(\sum_{j=0}^{N_{L-1}} w_{i,j}^L S_j^{(L-1)}\right) \tag{1}$$

where $(w_{i,j}^L)$, $j = 1, \ldots, N_{L-1}$, are the couplings. The threshold is taken into account by a zeroth unit in each layer, clamped in the $+1$ state ($S_0^{(L)} = 1$), so that $w_{i,0}^L$ is the bias. For a given set of p_0 (distinct) patterns of N_0 binary units, $\{\xi^\mu = (\xi_j^\mu = \pm 1, j = 1, \ldots, N_0), \mu = 1, \ldots, p_0\}$, we want to learn a given mapping $\xi^\mu \to \sigma^\mu$.

The mapping can be any Boolean function of the N_0 Boolean input units or, if $p_0 < 2^{N_0}$, the restriction to the p_0 patterns of any Boolean function. We will refer to the set of p_0 patterns on which learning is performed as to the training set, or the set of input patterns.

When the function (more precisely the restriction of the mapping to the training set) is linearly separable, a perceptron architecture is sufficient and the perceptron algorithm (Rosenblatt 1962, Minsky and Papert 1969) allows us to find a set of weights $w = (w_j)$, $j = 1, \ldots, N_0$, such that, for $\mu = 1, \ldots, p_0$,

$$\sigma^\mu = \operatorname{sgn}\left(\sum_{j=0}^{N_0} w_j \xi_j^\mu\right) \tag{2}$$

(with the above convention $\xi_0^\mu = 1$). There are several variants of this algorithm and in particular the 'Minover' algorithm (Krauth and Mézard 1987) allows one to find the solution w giving the largest possible stabilities.

In the non-linearly separable case, the perceptron algorithm does not converge: for any set of weights some patterns will have the wrong output. However, there is a simple variant of it, the 'pocket algorithm' (Gallant 1987), which allows one to find a good solution. It consists in running the usual perceptron learning algorithm, with a random presentation of the patterns, but keeping in memory (in one's pocket) the set of couplings which has produced the smallest number of errors so far. It has been shown that with probability as close to one as desired, it will give the set of weights w with the least number of errors. We will make extensive use of these algorithms.

Note that each hidden unit is a particular realisation of the perceptron: it realises one of the possible mappings of the patterns of the previous layer. We now explain how we build a net to learn a mapping $\xi^\mu \to \sigma^\mu$ for a given set of p_0 patterns of N_0 input units ($p_0 \leqslant 2^{N_0}$).

2.2. Theorem for convergence

We want to build a network layer by layer. Suppose that we have built the Lth layer, and that it is made of N_L units (plus the threshold unit). To each input pattern ξ^μ there corresponds a set of values of the neurons in the Lth layer. We shall refer to this set of values as the *internal representation* of pattern ξ^μ in the layer L. (The $L = 0$ layer is the input layer; the 0th representations are simply the input patterns themselves.) We say that two patterns belong to the same class (for the layer L) if they have the same internal representation, which we call the *prototype* of the class. If there are p_L distinct classes, we have thus p_L patterns (of $N_L + 1$ units), each of which is the representation of at least one input pattern. The problem is now to map these p_L prototypes onto the desired output.

In each layer L we will distinguish two types of units. The first unit will play a special role: we call it the master unit. In order to understand the role of the master unit, one has to compare for each input pattern the state of this unit in layer L with the desired output state. There is a certain number of patterns e_L for which the state of the master unit is not the desired output, and the remaining p_0-e_L patterns for which the master unit gives the desired output. This means that if the network is stopped at layer L, taking as output unit the master unit of this layer, then the network would produce e_L errors. From layer $L-1$ the algorithm will generate the master unit of layer L trying to minimise the number of errors e_L, and in particular it will always find a solution such that

$$e_L \leqslant e_{L-1} - 1. \tag{3}$$

Clearly, such an algorithm will converge after a finite number of layers. In the last layer L^*, $e_{L^*} = 0$, the master unit is identical to the desired output unit.

All the other units in layer L, and ancillary units, will be used only in order to fulfil the faithfulness condition. Clearly a necessary condition in order to learn the mapping is that two input patterns having different output should have different internal representations. We will say that a class is 'faithful' if all the patterns belonging to this class have the same output—and then this common output is the desired output for the prototype of this class. Otherwise the class is said to be 'unfaithful'.

The theorem which ensures convergence is the following.

Theorem. Suppose that all the classes in layer $L-1$ are faithful, and that the number of errors of the master unit, e_{L-1}, is non-zero. Then there exists at least one set of weights $\boldsymbol{w}$ connecting the $L-1$ layer to the master unit such that $e_L \leqslant e_{L-1} - 1$. Furthermore, one can construct explicitly one such set of weights $\boldsymbol{u}$.

Let us now give the proof which is very simple.

Proof. Let $\tau^\nu = (\tau_j^\nu, j = 0, \ldots, N_{L-1})$, $\nu = 1, \ldots, p_{L-1}$, be the prototypes in layer $L-1$. Each prototype τ^ν is the internal representation of a certain number V_ν of input patterns ($\Sigma_\nu V_\nu = p_0$), and all of them have the same output, which we shall denote s^ν. The master unit of the Lth layer is connected to the $N_{L-1}+1$ unit of the $L-1$ layer by a set of weights $\boldsymbol{w} = (w_j, j = 0, \ldots, N_{L-1})$. Obviously the couplings $\boldsymbol{w}$, defined by $w_1 = 1$ and $w_j = 0$ for $j \neq 1$, would give exactly the same master unit in layer L as in layer $L-1$, so that $e_L = e_{L-1}$. Consider now the following perturbation of this solution. Let μ_0 be one of the patterns for which $\tau_1^{\mu_0} = -s^{\mu_0}$, and define the set of weights $\boldsymbol{u}$ by $u_1 = 1$ and $u_j = \lambda s^{\mu_0} \tau_j^{\mu_0}$ for $j \neq 1$, where λ is some positive constant to be fixed later on. For a prototype μ, the master unit obtained with $\boldsymbol{u}$ will take the value

$$\begin{aligned} m^\mu &= \operatorname{sgn}\left(\sum_{j=0}^{N_{L-1}} u_j \tau_j^\mu \right) \\ &= \operatorname{sgn}\left(\tau_1^\mu + \lambda s^{\mu_0} \sum_{j \neq 1, j=0}^{N_{L-1}} \tau_j^{\mu_0} \tau_j^\mu \right). \end{aligned} \tag{4}$$

Clearly the prototype μ_0 is stabilised (i.e. $m^{\mu_0} = s^{\mu_0}$) if $\lambda > 1/N_{L-1}$. Consider now any stable prototype μ of the $L-1$ layer ($\tau_1^\mu = s^\mu$). The quantity $s^\mu s^{\mu_0} \Sigma_{j \neq 1} \tau_j^\mu \tau_j^{\mu_0}$ can take the values $-N_{L-1}$, $-N_{L-1}+2, \ldots, N_{L-1}$. However, the fact that the representations in the $L-1$ layer are faithful means exactly that the worst case, $-N_{L-1}$, can never be

obtained. Otherwise, one would have $s^{\mu}\tau_j^{\mu} = -s^{\mu_0}\tau_j^{\mu_0}$ for all j, including $j=1$, and this means (taking $j=0$) $s^{\mu} = -s^{\mu_0}$, $\tau_j^{\mu} = \tau_j^{\mu_0}$: μ and μ_0 would be two identical prototypes with different outputs. Thus one can choose $\lambda = 1/(N_{L-1}-1)$, so that in addition to all the prototypes μ stabilised in layer $L-1$, μ_0 is also stabilised. Hence $\boldsymbol{u}$ is one particular solution which, if used to define the master unit of layer L, will give $e_L \leqslant e_{L-1} - V_{\mu_0} \leqslant e_{L-1} - 1$.

This theorem opens a gate for the definition of various algorithms. We study now in detail one specific algorithm, the tiling algorithm. Clearly one would like to generate a network with the minimal architecture. Having this in mind as a general guide line, we propose an algorithm based upon: (i) a strategy for generating the master unit, in such a way that at each layer e_L is *at worst* equal to the value given by $\boldsymbol{u}$ (this is explained in § 2.3 below); (ii) a strategy for adding other units in the current layer, in such a way that one always ends up with faithful representations (this is presented in § 2.4 below). Thus convergence is guaranteed. Alternative algorithms inspired by the same ideas and the convergence theorem will be briefly discussed in the conclusion.

2.3. Generating the master unit

Suppose we have achieved the $(L-1)$th layer. We keep to previous notation: $\tau^{\nu} = (\tau_i^{\nu}, i=0, \ldots, N_{L-1})$, $\nu = 1, \ldots, p_{L-1}$, are the prototypes, each of them being the internal representation of a certain number V_{ν} of input patterns, the output of which is s^{ν}. We try to learn the mapping $\tau^{\nu} \to s^{\nu}$, $\nu = 1, \ldots, p_{L-1}$, with the following variant of the pocket algorithm. We run a perceptron algorithm with a random presentation of the patterns. At each update we keep 'in our pocket' the set of couplings which has produced the smallest possible number of errors with respect to the *input* patterns. This means that for each set of couplings $\boldsymbol{w}$ visited by the perceptron, we compute the number $e(\boldsymbol{w})$ of prototypes for which this set would not give the desired output, each prototype ν being weighted by its volume V_{ν}:

$$e(\boldsymbol{w}) = \sum_{\nu=1}^{p_{L-1}} V_{\nu}\delta\left(s^{\nu}\sum_{j=0}^{N_{L-1}} w_j\tau_j^{\nu}, -1\right) \tag{5}$$

(where δ is the Kronecker symbol). Eventually we get the set of couplings which minimises $e(\boldsymbol{w})$ among the $\boldsymbol{w}$ which have been visited by the perceptron. This 'optimal' set $\boldsymbol{w}^*$ gives a certain number of errors $e_L = e(\boldsymbol{w}^*)$. If in fact the perceptron algorithm converges, i.e. $e_L = 0$, the net is achieved: the $(L-1)$th representations are linearly separable, the master unit in the Lth layer takes the correct state σ^{μ} when pattern μ is taken as input.

If the particular set $\boldsymbol{u}$ of the previous section is taken as initial set in the pocket algorithm described above, the output set $\boldsymbol{w}$ will always satisfy (3). It is important to note that (3) is satisfied independently of the use of the pocket algorithm. The point of the pocket algorithm is just to speed up this convergence (i.e. generate less layers).

2.4. Building the ancillary units: divide and conquer

We now suppose that the master unit is still not identical to the desired output unit, and we explain how we add units in order to obtain faithful classes. Recall that we want to learn a mapping for $p = p_{L-1}$ distinct prototypes $\tau^{\mu} = (\tau_i^{\mu}, i=0, \ldots, N)$ of $N = N_{L-1}$ units plus the threshold unit. In this section we will call these prototypes 'patterns'. Each τ^{μ} is the prototype of one faithful class of the $(L-1)$th layer, and its

output s^μ is the common output of all the patterns of this class. We already know the zeroth unit of the Lth layer, i.e. the (future Lth) threshold unit, which is in the state 1 for every pattern, and the first unit, the master unit. At this point, the patterns belong to one of two classes whose prototypes are $(\tau'_0 = 1, \tau'_1 = 1)$ and $(\tau'_0 = 1, \tau'_1 = -1)$. The fact that the master unit is not equal to the output unit means that at least one of the two classes is unfaithful.

Suppose now that we have built $1 + N'$ units, each one being defined by its set of weights $w_i = (w_{ij})$, $j = 1, \ldots, N (i = 0, \ldots, N')$, and each pattern $\mu = 1, \ldots, p$ has its current representation in the Lth layer and its desired output s^μ. In general the number p' of distinct representations is smaller than p. By going through all the p patterns one can easily make an ordered list of these p' distinct current representations, $\tau'^\nu = (\tau'^\nu_i, i = 0, \ldots, N')$, $\nu = 1, \ldots, p'$, the new class prototypes, and associate with each of the p patterns a class number, i.e. the number of the prototype which is the representation of that pattern. Note that in order to see whether the representation of a pattern is identical to one of the prototypes it is sufficient to compute their overlap (scalar product), and to compare it with N'. As explained above, if all the classes are faithful, the layer is considered as achieved. If not, the next unit is obtained by picking one of the unfaithful classes, and one tries to learn (by a perceptron algorithm) the mapping $\tau^\mu \to s^\mu$ for the patterns μ belonging to this class only. In the best case, all this subset is learned: the class is then broken into two faithful classes. If not, it is always possible to learn (again by a perceptron algorithm) at least a well chosen part of this class in such a way that it will indeed be broken into two classes—although now at least one of them will still be unfaithful. In particular it is always possible to learn the mapping for any two patterns of the class having opposite outputs. (Here we suppose that there are no contradictory data-identical patterns leading to distinct outputs.) In practice, in order to pick one of the unfaithful classes, we ordered the classes by increasing sizes, and chose to learn the smallest one. Moreover, whenever the class was perfectly learned, we then tried to learn in addition the next class (with the same unit), and so on.

With this procedure, for each new unit at least one class is broken into two classes, so that with at most p units all the classes are faithful. We now give the results of the numerical simulations.

3. Simulations

3.1. Exhaustive learning

We have tested our algorithm on several classical tasks with training on the full set of the 2^{N_0} patterns. We give here the results for two problems of very different natures: the learning of parity and the learning of random Boolean functions.

3.1.1. Parity task. In the parity task for N_0 Boolean units the output should be 1 if the number of units in state $+1$ is even, and -1 otherwise. We have run the tiling strategy on this task for $N_0 \leqslant 10$. We end up with one layer of hidden units, with N_0 hidden units (plus the threshold unit) up to $N_0 = 9$, and 11 units for $N_0 = 10$. As an example we give in table 1 the solution found for $N_0 = 6$. The reader can check that this solution is strictly equivalent (through a 'gauge transformation') to the 'human' solution, where all the weights between the 6 input units and the 6 hidden units are equal $(+1)$, and the 6 thresholds $w_{i,0}$ are odd integers $(-5, -3, -1, 1, 3, 5)$ so that each

Table 1. The network generated by the algorithm when learning the parity task with $N_0 = 6$. There is only one hidden layer of 6 units.

Hidden layer							
Unit number, i	Threshold ($w_{i,0}^{(1)}$)	Coupling from the input layer to the hidden unit i					
1	−55	+11	+11	+11	−11	+11	−11
2	+33	−11	−11	−11	+11	−11	+11
3	−11	+11	+11	+11	−11	+11	−11
4	−11	−11	−11	−11	+11	−11	+11
5	+33	+11	+11	+11	−11	+11	−11
6	−55	−11	−11	−11	+11	−11	+11
Output unit							
Threshold		Couplings from the hidden layer to the output unit					
+11		+11	+13	+13	+13	+13	+11

hidden unit is coding for the patterns having at least a given number of +1 inputs (respectively 6, 5, 4, 3, 2, 1). In table 1 we have re-ordered the hidden units with respect to their thresholds. Once the internal representations are known (i.e. after the algorithm has converged), one can recompute the couplings with the Minover algorithm, leading to the optimal couplings for these internal representations. The weights quoted in the table have been obtained in this way. (Strictly speaking, the most elegant solution has all couplings from the hidden layer to the output unit equal. The Minover algorithm gives this optimal solution with any prescribed accuracy. Here we have asked only for a 10% accuracy.)

3.1.2. Random Boolean function. A random Boolean function is obtained by drawing at random the output (±1 with equal probability) for each input configuration. We present this academic example to show the ability of the tiling strategy to deal with any Boolean function, even when there are no regularities. Table 2 gives, for $N_0 = 4$, 6 and 8, the mean number of layers $\langle L \rangle$ for 100 random functions, and for each layer the mean number of hidden units $\langle N_L \rangle$ (without counting the threshold unit).

It is not surprising that for a function without any regularities, the numbers of layers and of hidden units increase rapidly with N_0. This example shows that it is possible to end up with a rather complicated architecture.

3.2. Generalisation

Once a network has been built by the presentation of a training set, it performs the correct mapping for all the patterns in this training set. The next question is: how does it perform on new patterns? The general belief is that these generalisation properties essentially depend on 'phase space', or 'entropic', factors. For a fixed geometry of the network, varying the couplings, one generates a space Ω of networks. Training selects a subspace Ω_t of Ω (the subspace of all networks which give the correct output for each input of the training set). Now, inside Ω_t lies the subspace of solutions Ω_s consisting of networks which actually implement the function one is learning

Table 2. Learning random Boolean functions with $N_0 = 4, 6$ and 8. For each N_0 we give the average number of layers (for 100 random functions) and the average number of hidden units in each layer. For each layer the average is over the trials for which the total number of layers is at least equal to L. The percentage of these trials is given in parentheses.

	$N_0 = 4$	$N_0 = 6$	$N_0 = 8$
$\langle L \rangle$	2.08 ± 0.05	3.75 ± 0.07	7.13 ± 0.08
$\langle N_1 \rangle$	2.68 ± 0.08 (100%)	8.55 ± 0.09 (100%)	15.57 ± 0.10 (100%)
$\langle N_2 \rangle$	1.15 ± 0.04 (95%)	4.68 ± 0.12 (100%)	11.55 ± 0.10 (100%)
$\langle N_3 \rangle$	1.0 (13%)	2.09 ± 0.10 (99%)	10.80 ± 0.12 (100%)
$\langle N_4 \rangle$		1.20 ± 0.06 (64%)	8.50 ± 0.14 (100%)
$\langle N_5 \rangle$		1.0 (12%)	5.84 ± 0.16 (100%)
$\langle N_6 \rangle$			3.01 ± 0.17 (100%)
$\langle N_7 \rangle$			1.73 ± 0.11 (75%)
$\langle N_8 \rangle$			1.06 ± 0.06 (36%)
$\langle N_9 \rangle$			2.0 (1%)
$\langle N_{10} \rangle$			1.0 (1%)

(remember that one function can be realised by several networks). The ability to generalise has been found to increase as the ratio of the volumes of Ω_s to Ω_t increases (Carnevali and Patarnello 1987, Denker *et al* 1987). From this idea one may expect that the generalisation will improve whenever one is able to learn the same training set with fewer hidden units. If this turns out to be the case, our strategy has the great advantage of generating only the hidden units that it needs in order to learn the training set, not more.

In practical applications N_0 need not be small (Sejnowski and Rosenberg 1987), and the size of the training set is smaller than 2^{N_0}. There are two reasons why the network is able to achieve good performance nevertheless. First of all the task can be relatively simple (while non-linearly separable), and secondly what is very important is that the actual patterns which are presented to the network are in general strongly correlated: they never span the space of all configurations uniformly. For these reasons we have decided to test the generalisation properties of the tiling strategy on a problem where N_0 is not small (such that the complete scan of all patterns is impossible), and the patterns one wants to analyse are generated stochastically, but with strong correlations which impose that only a small fraction of configuration space has to be analysed in practice.

We have chosen a simple example which belongs both to statistical physics and to pattern recognition. The N_0 input neurons are organised in a one-dimensional chain, and the problem is to find out whether the number of domain walls is greater or smaller than three. (A domain wall is the presence of two neighbouring neurons pointing in opposite directions. We use periodic boundary conditions so that if the first and last neurons point in opposite directions this is interpreted as a domain wall. Obviously the number of domain walls must be even, and is twice the number of domains. This is exactly the problem which has been studied (Denker *et al* 1987) on small samples, under the name 'two or more clumps'.) The patterns are generated through a stochastic procedure: they are thermalised configurations of the one-dimensional Ising chain, obtained through a Monte Carlo method (Binder 1979). The temperature of thermalisation has been chosen as $T = 2/\ln(N_0/k)$ so that there will be an average of k domain walls in the patterns. The use of the free parameter k allows us to tune the difficulty

of the problem. If k is much larger (respectively, much smaller) than three, the problem is easy in the sense that there are nearly no patterns with less (respectively, more) than three domain walls and the output is nearly always 1 (respectively -1). The problem is harder for k of the order of three.

Most of the simulations were performed with $N_0 = 25$, although we also used $N_0 = 50$. (Actually the value of N_0 is not a limitation for our algorithm. In the actual implementation of the perceptron algorithm, we use only the matrix of overlaps $Q_{\mu\nu} = \Sigma_{i=1,\ldots,N_0} \xi_i^\mu \xi_i^\nu$. The computation of this matrix is the only place where an increase in N_0 increases the complexity of our algorithm.) Training sets of various sizes ranging from $p_0 = 50$ to $p_0 = 600$ were used, and the test sets had the same size as the training sets. Of course in all cases the algorithm found a network which learnt exactly the whole training set. The generalisation was measured as the ratio of the number of correct answers in the test set to the size of this set. The results were averaged over several samples (different choices of random patterns in the training and test sets). The results are presented in figure 1, for values of k equal to 1, 3, and 5. As expected k of order 3 is the hardest case. In simple cases like $k = 1$ the algorithm generates a small network which performs well even after a short training, but there is not much improvement when larger training sets are used, since there are almost no patterns which show what is the rule one wants to extract. For $k = 3$ the performance is not as good, but it improves with training. One should notice that no hint of any kind has been given to the network, so that the tiling strategy seems to perform better than back-propagation which apparently does not generalise, even for smaller N_0, if no hint is given (Denker *et al* 1987).

As is well known (Rosenblatt 1962, Minsky and Papert 1969), a suitable preprocessing can render the learning task easy—i.e. learnable by a perceptron algorithm for

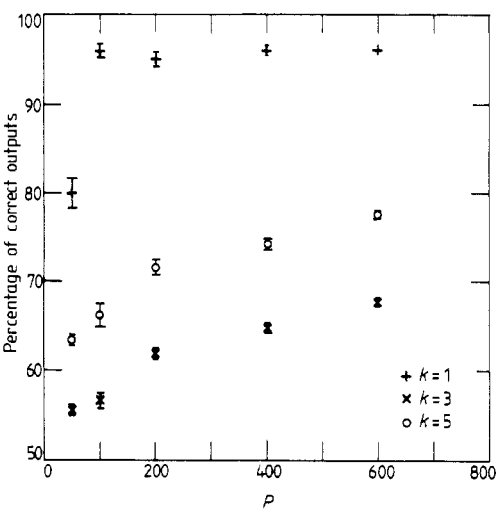

Figure 1. Learning to discriminate Ising chains by the number of domain walls (greater or smaller than three). Shown is the percentage of patterns of $N_0 = 25$ units with the correct output in the test set, as a function of the size p_0 of the training set (equal to the size of the test set). The average number of domain walls in the presented patterns is k. For each k and p_0, data are averaged typically over 25 samples (250 samples for $p_0 = 50$). For $p_0 = 600$, the average number of layers of the network was respectively 3.0, 8.5 and 5.5 for $k = 1$, 3 and 5.

example. For instance one can make a first layer of N_0 units which is a detector of domain walls (see, e.g. Denker *et al* 1987). Then the remaining task is to count the number of units in the +1 state in this layer, a task learnable by a perceptron algorithm. The strategy did not find here this 'human' solution—contrary to the case seen above of the exhaustive learning of the parity problem. This may be simply due to the fact that here learning is performed on a rather limited training set. This problem is certainly worth further studies.

3.3. Quality of convergence

To quantify the quality of convergence one might think of at least two parameters. The first is the number of errors e_L which would be produced by the network if stopped at layer L, as defined in § 2. In addition to having the mapping exactly learned, one would like that the internal representation gets smaller and smaller (i.e. the number of classes diminishes) from one layer to the next one. Thus a second interesting parameter is the number of distinct internal representations (classes) p_L in each layer L.

We have measured numerically the evolution of p_L and e_L as functions of the layer index L for the problem of domain wall discrimination defined in § 3.2. The data shown on figure 2 result from an average over 25 samples with $N_0 = 25$, $p_0 = 600$ and $k = 3$. The average number of layers was found to be 8.5 ± 0.2 and for all the samples the algorithm converged with at least 7 layers and at most 11 layers. It is interesting to notice that in the range $2 \leq L \leq 7$ the decrease in e_L is linear in L, and this seems

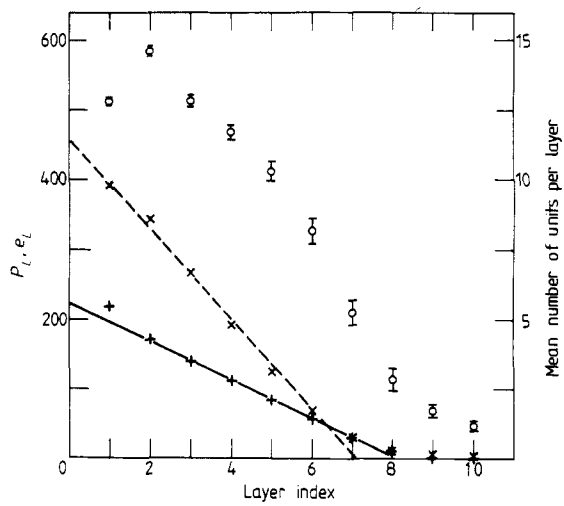

Figure 2. Statistics (obtained by averaging over 25 samples) of the architecture of the network after learning of $p_0 = 600$ patterns of $N_0 = 25$ units, having on average $k = 3$ domain walls. (The learning task is to discriminate the patterns according to the number of domain walls, greater or smaller than 3.) Three quantities are plotted as functions of the layer index. (i) The mean number of units in a layer (○) (measured against the right-hand scale), which has a non-monotonous behaviour before decaying regularly to 1. (ii) The number of classes p_L (×) (measured against the left-hand scale). (iii) The number of errors e_L (+) (measured against the left-hand scale) as measured with respect to the master unit of layer L (see text). The straight lines are only guides for the eye. When not shown, the error bars are less than the size of the points.

to be also the case for p_L. We have observed a similar behaviour in the case of random Boolean functions. Although we have no interpretation so far for these linear behaviours, it is thus tempting to use the slope of the linear decrease of the percentage of errors as a measure of the complexity of the problem to be learnt.

3.4. Comments

Before we conclude, several comments are in order. Clearly it is useful to limit as much as possible the number of hidden units. In this respect, variants of the algorithm might be helpful. There is a lot of freedom in the choice of the unfaithful classes to be learnt. Our choice has been to learn first the smallest classes, but we have not investigated other choices. A possible variant, which we have not tried, would be to introduce a 'sleep' phase (Rujan and Marchand 1988): once a layer is achieved, one could see whether some units can be removed so that the classes remain faithful.

We have tested a variant of the algorithm where the master unit is generated differently. In that variant, it is the standard pocket algorithm which is run on the set of prototypes of the preceding layer—i.e. each prototype ν has weight 1 instead of its volume V_ν. In that case, the convergence is no longer guaranteed, and we have not been able so far to prove that this strategy will always converge. However we found in all our simulations that this strategy performed almost as well as—and some times better than—the tiling algorithm detailed above.

Apart from the above variants, there is only one adjustable parameter in the algorithm. This is the maximum number of iterations which are allowed before one decides that the perceptron algorithm has not converged. If this parameter is too large, there is a waste of CPU time. If it is too small, the non-convergence decision will sometimes be wrongly taken, and the size of the net may grow unnecessarily. A better theoretical understanding of the time of convergence of the perceptron algorithm would be very helpful in order to optimally control this parameter.

Since we have not yet tried to optimise the algorithm, computer times can only be given as a rough estimate of the performances of the algorithm. For example, the parity problem for $N_0 = 6$ and $N_0 = 8$ took respetively about 1 and 5 CPU seconds on a Convex C1.

4. Concluding remarks

We have presented a new strategy for building a feedforward layered network for any given Boolean function. With respect to previous approaches to learning in layered systems, such as backpropagation, our approach presents a completely new way of addressing the problem. The geometry, including the number of units involved and the connections, is not fixed in advance, but generated by a growth process. We have identified some possible roles of the hidden units: the master units drive the growth and ensure a number of errors which decreases strictly from one layer to the next; the ancillary units complete each layer in order to get faithful internal representations. In practice, numerical tests of the tiling algorithm on model examples are very promising. Clearly many things remain to be done. One can try to improve the algorithm in particular by investigating variants of the perceptron algorithm used to build the hidden units. It would be interesting to systematically compare the performances (efficiency,

computer time, size of the architecture...) of the tiling algorithm with those of other algorithms.

Generalisation to neurons with continuous values is worth studying. A relatively simple case is that of continuous inputs and binary outputs, for which the algorithm should work without any modification. The only new feature is a problem of precision: it may happen that the generation of ancillary units in the first hidden layer require much computer time whenever there exist two patterns which are very close to one another and have opposite outputs.

In this respect, it is also useful to mention a possible generalisation to the case of conflicting data. This is the case where identical patterns have opposite outputs. Obviously it is not possible to learn exactly all the patterns, the minimum number of errors, $e_{\min}$, being simply determined by the number of conflicting inputs. Then it is not possible to get a faithful representation in any layer (since the representation in the input layer is itself unfaithful). One must stop adding ancillary units as soon as the maximal set of unconflicting patterns has been represented faithfully. In this way one can reach an error level equal to $e_{\min}$.

The theorem of § 2.2 also suggests other strategies. For example one can obtain faithful classes by simply connecting the master unit of layer L to the master unit of layer $L-1$ and to all the units of the input layer. Then the internal representation of a pattern in layer L is the union of the pattern itself and of the associated state of the master unit of this layer, and there are p_0 distinct internal representations. We have not yet investigated the performances of this algorithm.

To generalise these algorithms to several output units, one can generate in each layer as many master units as there are output units. After this has been done one generates the ancillary units which build up faithful classes. One interesting perspective is to find a strategy which limits as much as possible the number of units in each layer.

Acknowledgments

We are grateful to G Toulouse for very useful suggestions. We thank P Rujan for communicating his results prior to publication. Numerical simulations were performed partly on a SUN workstation provided by a DRET contract, and partly on a Convex C1 thanks to a contract with the European Communities—program BRAIN, contract number ST2J-0422-C (EDB).

References

Binder K 1979 *Monte Carlo Methods in Statistical Physics* (*Topics in Current Physics* **7**) (Berlin: Springer)

Carnevali P and Patarnello S 1987 Exhaustive thermodynamical analysis of Boolean networks *Europhys. Lett.* **4** 1199

Denker J, Schwartz D, Wittner B, Solla S, Hopfield J J, Howard R and Jackel L 1987 Large automatic learning, rule extraction, and generalization *Complex Systems* **1** 877

Gallant S I 1986 *Optimal linear discriminants IEEE Proc. 8th Conf. on Pattern Recognition, Paris*

Grossman T, Meir R and Domany E 1988 Learning by choice of internal representations *Complex Systems* **2** 555

Krauth W and Mézard M 1987 Learning algorithms with optimal stability in neural networks *J. Phys. A: Math. Gen.* **20** L745

Le Cun Y 1985 A learning scheme for asymmetric threshold network CESTA-AFCET *Cognitiva* pp 599-604; 1987 Modéles connexionistes de l'apprentissage *Thesis* Paris

Minsky M and Papert S 1969 *Perceptrons* (Cambridge, MA: MIT Press)

Rosenblatt F 1962 *Principles of Neurodynamics* (New York)

Rujan P and Marchand M 1988 Learning by activating neurons: a new approach to learning in neural networks *Preprint* IFK Jülich (submitted to *Complex Systems*)

Rumelhart D E, Hinton G E and Williams R J 1986 Learning internal representations by error propagation *Parallel Distributed Processing* vol 1, ed D E Rumelhart and J L McClelland (Cambridge, MA: Bradford)

Rumelhart D E and McClelland J L (ed) 1986 *Parallel Distributed Processing* vols 1, 2 (Cambridge, MA: Bradford)

Sejnowski T J and Rosenberg C R 1987 Parallel networks that learn to pronounce English text *Complex Systems* **1** 145

Werbos P J 1974 Beyond regression: new tools for prediction and analysis in the behavioral sciences *PhD thesis* Harvard University