LEARNING IN MULTILAYER NETWORKS:
A GEOMETRIC COMPUTATIONAL APPROACH

Pál Ruján

KOSY AG, Fachbereich 8 Physik, Oldenburg Universität
Postfach 2503, D-2900 Oldenburg, Federal Republic of Germany

to appear in Proc. IX Sitges Conf. on Neuronal Networks, Lecture Notes in Physics, Springer Verlag

Fachbereich Physik, Universität Oldenburg
D-2900 Oldenburg, FRG
LEARNING IN MULTILAYER NETWORKS:  
A GEOMETRIC COMPUTATIONAL APPROACH

Pál Ruján

KOSY AG, Fachbereich 8 Physik, Oldenburg Universität
Postfach 2503, D-2900 Oldenburg, Federal Republic of Germany

1 PERCEPTRONS AND FEEDFORWARD NETWORKS

Artificial neural networks are simple models attempting to capture some essential features of the central nervous system. The arguments on exactly what are these essential features and to what extent they are missing from the fashionable theoretical models will certainly continue to generate heated debates in the foreseeable future. Here we will discuss several properties important for possible technological applications, namely learning and generalization abilities of multilayer feedforward networks. Our approach is based on a geometric picture of how feedforward networks process information. This forms the basis for constructing 'good' networks. Several network growing algorithms are discussed.

Until recently, the attention of physicists has focused on simple Perceptrons like models, which can be treated by methods borrowed from the statistical mechanics of random systems. These methods are best applicable when the set of examples consists of uncorrelated bit strings. This is the average 'worst case' for such networks, since a random boolean is by definition an unpredictable function. On the other hand, the overwhelming majority of boolean functions are random ones and these results provide strict upper bounds on the information capacity, for example. Our own experience shows that a geometric view is a helpful alternative for understanding neural networks[1,2], as suggested long ago by Pellionisz and Llinás[3] in a different context. This geometric approach is strongly linked to combinatorial optimization theory [4] and shares many of its concepts and difficulties.

Our basic processing unit is a McCulloch-Pitts [8] binary unit (or linear threshold gate) which is activated only when the weighed sum of its inputs
is larger than some threshold:

$$\sigma = \text{sgn} \left( \sum_i w_i s_i - \Theta \right)$$

(1)

where the vector $\vec{w} = (w_1, \ldots, w_N)$ is a vector of weights representing the strengths of $N$ impinging synapses from $N$ other neurons with activations $(s_1, \ldots, s_N)$, $s_i = \pm 1$. $\Theta$ is a threshold value. The output of the processing unit is $\sigma = 1$ if the ‘neuron’ fires a high frequency train of spikes or $\sigma = -1$ if it remains inactivated. From a geometric point of view Eq. (1) says that every elementary processor of our network corresponds to a $N$-dimensional hyperplane

$$\vec{w} \vec{s} = \Theta$$

(2)

and that the points of the $N$-dimensional configuration space are partitioned in two classes, which are called black for $\sigma = 1$ and white for $\sigma = -1$. In what follows the terms ‘hyperplane’ and ‘unit’ are used interchangeably, associating the vector $\vec{w}$ and the threshold $\Theta$ with unit $\sigma$. In many models the activation values $\sigma$, $s_i \in (0,1)$ are continuous variables and the step-like activation function (1) is a sigmoid. Physically, such situations correspond to a probabilistic updating scheme and the output variable $\sigma$ is then essentially a time or ensemble average activation [5]. The geometric picture remains valid, provided that one works with slabs of finite width instead of geometrical hyperplanes.

If the input units are the output of some feature detectors, $s_i = \pm 1$, the resulting one-processor network is a simple classifier better known under the name of Perceptron [8] see Fig. 1. Note that the Hopfield model, for example, is an ensemble of $N$ Perceptrons, one for each unit. As discussed by Minsky and Papert [7], Perceptrons can represent only the class of ‘linearly separable’ boolean functions (LSF) and many practical tasks fall outside this class. Adding one or more layers of so-called hidden units but without allowing feedback connections leads to a multilayer feedforward architecture. Such devices can implement arbitrary boolean functions at the price of using most of the times an exponentially large number of hidden units. Only connections between nearest neighbour layers will be allowed. Fig. 2
Figure 2: A feedforward network with three input units, two hidden units and one output unit. The numbers inside the circles represent threshold values and those on connections connection strengths, respectively.

shows an example of a simple feedforward network.

Unlike attractor networks, which relax to their low energy states by a Monte Carlo-like dynamics, feedforward networks run only for a number of steps limited by the number of layers, very much like a fast arithmetic unit or a Programmable Array Logic chip. This is an attractive feature from a technological point of view and makes such networks a viable model of parallel computation.

Assume now that a (predictable) source is generating a sequence (set) of digitized examples and that a 'teacher' provides information on the class to which each particular example belongs. In our case every example (object) is represented by an input bit string and may belong to one of two classes. Following some learning algorithm the network (machine) adjusts its internal parameters (connections and thresholds) so as to commit a minimal error on the presented training set. This task is called supervised learning or learning from examples. When the example set is loaded (learnt) into an appropriate architecture the training process leads usually to a good representation of the example generating function. This means that further examples from the same source are with high probability correctly classified, provided the function is predictable at all. In contrast, a look-up (or hash) table does not provide any information on new, unknown examples.

2 GEOMETRIC REPRESENTATION

A feedforward network can be represented as follows:

$$\sigma = f(s_1, \ldots, s_N; \{w_{ij}\}, \{\Theta_i\})$$  

(3)

where the dependence on the adjustable connection weights \{w_{ij}\} and the set of thresholds \{\Theta_i\} has been explicitly displayed. The boolean function ($\sigma = \pm 1$) satisfies a set of input-output (IO) examples $\xi^{(\nu)} = (s_1^{(\nu)}, \ldots, s_N^{(\nu)}; \sigma^{(\nu)})$ without any error. Note that the last element of the
vector $\boldsymbol{z}^{(v)}$ is the desired output value and $\xi_i^{(v)} = \pm 1$. The \textit{configuration input space} is defined as the set of vertices in a $N$-dimensional unit hypercube. All possible input configurations are corners of this hypercube. Only some of these corners correspond to given examples: we will colour them according to the desired output value. For the feedforward network shown in Fig. 2 this is shown in Fig. 3a. Any such function can be rotated, reflected, etc. according to the symmetries of the $N$-dimensional unit hypercube. Other, gauge type symmetries are related to changing signs of connections, thresholds and of some input units but leaving $\sigma$ invariant. A good network function (3) should capture all these symmetries.

Consider once again the feedforward architecture shown in Fig. 2: it has three input units $N = 3$ and two hidden units, $N_h = 2$. Depending on the connections between the input and the middle layer and on the threshold values for the hidden units each input string will induce a given configuration of hidden units. An $N_h$ dimensional unit hypercube represents now the \textit{hidden configuration space}. The set of coloured (marked) corners of this cube is the image of the original problem on the hidden layer. The relation between the configuration cube representations and the feedforward architecture of Fig. 2 is explicitly shown in Fig. 3. Every set of connections to a hidden unit corresponds to a hyperplane partitioning the input configuration hypercube in two. The connections between the output and the hidden units are represented by a single hyperplane (here a line) lying in the \textit{hidden} unit hypercube. Each layer of a multilayer network forms its own image (or internal representation) of the original function.

A convex combination of $M$ vectors $\{\xi^{(v)}\}_{v=1}^M$ is defined as

$$S = \{ \bar{z} \in S \mid \bar{z} = \sum_{\nu=1}^M a_{\nu} \xi^{(v)} ; \sum_{\nu} a_{\nu} = 1, a_{\nu} \geq 0 \}$$

An extremal point of a convex combination cannot be expressed as a linear combination of other points in the set. Given a set of points on the configuration hypercube one can construct the convex hull of this set as a convex combination of the form (4). The convex hull forms an $N$-dimensional con-

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3.png}
\caption{The hypercube representation of the network shown in Fig. 2. a) The input configuration cube b) The hidden configuration square. Note the one-to-one correspondence between the network form and the components of the normal vectors of the planes shown here.}
\end{figure}
A schematic representation of a minimal connector in 2 and 3 dimensions.

The vertices, edges, ..., faces of this polytope have dimensions 0, 1, ..., \( n \) and are themselves convex polytopes of lower dimensionality. The facets are faces of \( n = N \) dimensionality.

The intersection of two convex polytopes is also a convex polytope. The minimal distance between two nonintersecting convex polytopes is called the minimal connector. A schematic view of a minimal connector is shown in Fig. 4.

Consider now a vector \( \vec{w} \) in the configuration space. The map associated with this direction is the set \( \{ h^{(\nu)} \} \),

\[
h^{(\nu)} = \vec{w}^T \xi^{(\nu)}
\]

and consists of the set of projections (internal fields) of the marked unit cube corners on this direction.

Every learning algorithm for multilayer networks incorporates the following two subtasks:

- Problem 1. Is the problem linearly separable or not? If the problem is linearly separable no (further) hidden units are necessary and one can go to

- Problem 2. Find a hyperplane correctly classifying the set of examples of a linearly separable problem.

Both problems are soluble in a polynomial number of steps (as a function of the number of examples and dimension \( N \)). The link to the geometrical picture presented above is very simple. First, construct the convex hull \( S_1 \) corresponding to the \( M_1 \) black points and the polytope \( S_0 \) corresponding to the \( M_0 \) white points (\( M = M_0 + M_1 \)):

\[
S_0 = \{ \vec{X}_0 \in S_0 | \vec{z} = \sum_{\nu=1}^{M_0} a_{\nu} \xi^{(\nu)} ; \sum_{\nu} a_{\nu} = 1, a_{\nu} \geq 0 \}
\]

\[
S_1 = \{ \vec{X}_1 \in S_1 | \vec{z} = \sum_{\nu=1}^{M_1} b_{\nu} \xi^{(\nu)} ; \sum_{\nu} b_{\nu} = 1, b_{\nu} \geq 0 \}
\]

Here \( I_1 \) and \( I_0 \) is the set of indices corresponding to black and white points, respectively. If the intersection of \( S_0 \) and \( S_1 \) \( P = S_0 \cap S_1 = \emptyset \) the problem is linearly separable, otherwise it is not. Actually, if any two faces of the
convex hulls $S_0$ and $S_1$ are not disjoint the problem is not linearly separable. In terms of maps the same question can be formulated as following: is there a direction $\vec{w}$ in the $N$ dimensional space such that

\begin{align*}
h_\nu & \geq \alpha \forall \nu \in I_1 \\
h_\nu & \leq \beta \forall \nu \in I_0 \\
\alpha & > \beta
\end{align*}

(\text{8})

Assume now that we have found an architecture which reproduces exactly the whole training set. What are the geometric properties of such a solution? The hidden units are partitioning (or tiling) the space of the input configuration cube with hyperplanes, each hyperplane corresponding to the synaptic strengths afferent to a hidden unit. The first observation is that an exact representation of the example function is possible only if in every box, part of space bounded by hyperplanes and by the facets of the unit hypercube, one finds only corners of the same colour. In fact, when considering the image of the function on the first layer for every input cube box there will be only one representative corner of the first layer configuration cube. In other words: every marked corner within a given box is mapped into a single configuration of the hidden units. This is evident since the activation patterns on the first hidden layer can change only when one moves across the boundary of a box - and then only one of the coordinates will change. Thus the number of distinct activation patterns for the hidden units equals the number of non empty boxes.

The following construction will always reproduce correctly all IO examples. Separate every black corner by a hyperplane whose normal vector $\vec{w} = \xi^{(\nu)}$, $\nu \in I_1$ and the threshold is such that the plane ‘cuts out’ (separates) only the point in question. This kind of solution is of ‘grandmother’ type, because every black input pattern corresponds now to an uniquely specified hidden configuration, very much like grandmother neurons (if they exist) should be activated only on recognition of your grandmother. One can show (see the regular partition theorem below) that this construction leads to a linearly separable hidden configuration space, thus only one layer of hidden units suffices. Grandmother type solutions are extremely localized representations without any generalization ability: they are the neural network counterparts of usual storage devices. Many theories of neural information processing are based explicitly or implicitly on this type of knowledge representation (including Grossberg's ART networks) and - except perhaps for existence proofs - should be avoided. Since physicists have a lot of experience with cooperative phenomena we perhaps do not need here a long discussion on the advantages of distributed representations[9].

The geometric picture suggests that the right thing to do is try packing as many marked corners in a box as possible. In this case the number of distinct hidden configurations will be a small fraction of the original number of examples, implying a strong contraction of the ‘excitable’ configuration space from one layer to the next. This contraction is quite natural, since in output the configuration space will consist of only one output variable. This heuristic principle of maximal contraction is a good guide in constructing algorithms. If one can prove that in a concrete procedure the image of the original function is contracting from layer to layer, then this algorithm will eventually find an architecture representing correctly all presented examples. Such a strategy has been used by Mézard and Nadal [10] to prove that their tiling algorithm is converging.

In conclusion: a good architecture is obtained by a partition of the unit
hypercube with hyperplanes so that the formed boxes contain as many corners of the same colour as possible. The next question then arises naturally:

- **Problem 3.** Under what conditions is the configuration space of the last hidden layer linearly separable?

This question is again related to linear separability. The hidden unit configurations, however, have some special properties leading to tight sufficient conditions so that the image of a function on a hidden layer is linearly separable. We give here two such conditions. One is restricting the way the space is partitioned in boxes, namely requiring that the hyperplanes corresponding to the hidden unit connections be the boundary of exactly two boxes, one containing black and one containing white corners. We called such tilings regular partitions [1, 2]. Two such cases are shown in Fig. 5a-b for the parity-3 problem (Fig. 4b shows the grandmother solution). Figs. 5d and 5e are schematic representations of these regular partitionings. The center points of each box represent a whole cluster of corners and are connected into a bipartite tree. Figs. 5f, 5g display the same solutions in the more conventional network form.

The schematic section of the configuration space (Figs. 5f, 5g) shows the main idea of the proof: construct the dual graph of the tiled configuration space. The vertices of this graph are representative internal points of the non empty boxes and are coloured accordingly, similar to Fig. 3b. These points are connected by straight edges. By the definition of regular partitions, every edge of the graph is intersecting only one hyperplane boundary. From the conditions of the tiling it is easy to see that the dual graph must be a bipartite tree and that every edge points on a different direction. A separating plane can be constrained to pass through the median point of each edge by solving a sparse set of linear equations. It is easy to show that all black (white) vertices of the graph will fall (at the same distance) in one side of this plane. In this respect the regular partitions are basically generalizations of the Adaline method at the hidden layer level.

The second formulation is related to a sequential learning algorithm which requires only that the hidden units are set successively and at every step only a box containing corners of the same colour is separated from the input configuration space (sequential learning procedure [11]). A proof
similar to the stratification method of Minsky and Papert [7] shows that the resulting tiling is linearly separable and is given in Appendix A.

These two sufficient conditions ensure that a wide class of feedforward networks with one single hidden layer can perfectly represent an arbitrary set of IO examples. A simple criterion for 'good' architectures in terms of random boolean functions, constituting the average worst case situation, is the following. As explained above, the main obstacle to linear separability is the presence of a non void intersection polytope, \( P = S_0 \cap S_1 \). Every new hyperplane (hidden unit) cuts a chunk out of \( P \) and a sequential learning procedure, for example, ends when \( P \) has been 'eaten' up. It is obvious that the facets of \( P \) are parallel to either a facet of \( S_0 \) or a facet of \( S_1 \) and can be identified by solving a linear programming problem. Looking at the map in the direction normal to the facet, one can define two parallel planes cutting off at least the \( N \) linearly independent corners defining the facet. On the other hand, it is obvious that the number of example patterns per hidden unit cannot be larger than \( 2N \), the maximal storage capacity of the Perceptron. Thus the number of hidden units \( N_h \) needed to represent a random boolean function in a one internal layer network is bounded by

\[
\frac{M}{2N} < N_h < \frac{2M}{N}
\]  

where \( M \) is the number of examples and \( N \) the number of input units. For more hidden layers, including tree like structures, \( N_h \) should be smaller. This particularly simple result shows that in average \( 2M/N \) hidden units are enough to ensure a solution for a typical set of examples. Although this bound does not tell us anything about classes of functions with particular symmetries, it suggests that studying the facet structure of the convex polytope \( P \) might help in finding bounds on the sufficient number of hidden units. Thus, it seems plausible that the structure of \( P \) facets is also determining to what extent is a function learnable in Valiant's sense [13].

3 LEARNING ALGORITHMS AND THEIR COMPLEXITY

Learning is the search in the space of connections and thresholds (fixed architecture) or in more general, the search in the space of all possible architectures as to reproduce without errors a given set of IO examples. The problem of noisy inputs is not discussed here, nor are learning algorithms leading to a minimal error probability for further examples sampled from the same source. Before discussing algorithms for multilayer networks we shortly summarize the methods known for the Perceptron (for more details see [12]), in particular, the Fisher discriminant method[14], the least mean square error (Adaline) method [15] and the optimal Perceptron method[16,17].

The Fisher discriminant method

This is a standard method in pattern recognition and is based in the simple statistical picture shown schematically in Fig. 6. Assume that the classes of white and black points follow a distribution similar to the normal distribution. Consider a unit vector \( \mathbf{w} \) and the projection of the center of
mass and of the variance of these distributions into direction $\bar{w}$ ($\alpha = 0, 1$):

$$\mu_\alpha = \frac{1}{M_\alpha} \sum_{\nu \in I_\alpha} \bar{w}^{(\nu)}_\xi$$
$$\sigma_\alpha = \frac{1}{M_\alpha} \sum_{\nu \in I_\alpha} (\bar{w}^{(\nu)}_\xi - M_\alpha \mu_\alpha)^2$$

(10)

The Fisher discriminant method is based on the idea that a good classifying hyperplane should maximize the distance between the average position of the two distributions and minimizes the sum of the two variances.

$$\max_{\bar{w}} F = \frac{(\mu_1 - \mu_0)^2}{\sigma_1^2 + \sigma_0^2}$$

(11)

Note that the direction pointing from the center of mass of the distribution of white points to the center of mass of the black points,

$$\bar{w} = \frac{1}{M_1} \sum_{\nu \in I_1} \xi^{(\nu)} - \frac{1}{M_0} \sum_{\nu \in I_0} \xi^{(\nu)}$$

(12)

corresponds to the Hopfield learning rule and maximizes the distance between the two centers of mass.

The linear discriminant method implies the inversion of an $N \times N$ matrix or can be iteratively obtained by a gradient descent method.

**The least mean square error (Adaline) method**

The Adaline method[15] is a similar method, based on the observation that Eq. (9) can be rewritten as

$$\sigma h^\nu - \sigma \alpha \geq 0 \text{ for } \nu \in I_1, \sigma = 1$$
$$-\sigma h^\nu - \sigma \beta \geq 0 \text{ for } \nu \in I_0, \sigma = -1$$

(13)

(14)

Since this set of inequalities is homogeneous, the scale can be fixed by one of the two constants $\alpha, \beta$ or by their difference. Using $\alpha = \theta + 1$, $\beta = \theta - 1$ the squared error cost function is

$$E_{Adaline} = \sum_{\nu \in I_1} (h_\nu - \theta - 1)^2 + \sum_{\nu \in I_0} (\theta + 1 - h_\nu)^2$$

(15)

The Adaline method is basically constructing a hyperplane passing through $M$ linearly independent points, thus the storage capacity—the number of examples which can be exactly stored—is $N$. Recalling Eq.(7) one obtains that in this case the direction of the plane $\bar{w}$ is a linear combination of only $M$ linearly independent set of examples:

$$\bar{w} = \sum_{\mu=1}^{M} x_\mu \xi^{(\mu)}$$

(16)

When the set of examples is larger than $N$, Adaline makes a least mean squared fit, so as to minimize the average squared distance of black (white) points from the corresponding planes. As shown in Fig. 6 for a special (schematic) distribution of example configurations, this separating plane is not particularly well suited for generalizations, a fact supported by recent analytic calculations[18]. The minimization implies the calculation of the generalized inverse of a $M \times N$ matrix or can be obtained by the fa-
Figure 6: A schematic representation of different methods for solving linearly separable problems. a) Hopfield-rule, b) Fisher determinant, c) Adaline, d) optimal Perceptron.

The optimal Perceptron

Originally, the optimal Perceptron method has been introduced by P. Lambert [16] in a form which slightly differs from the definition used nowadays in the physics literature. Consider once again the geometric picture of the convex hull of black and white examples and assume for a moment that their intersection is void, so the problem is linearly separable. The 'best' separating plane is the one which is most robust to changes in connection strengths and thresholds, corresponding to rotations and translations, respectively. Treating rotations in the $N$ dimensional hypercube is awkward. Translations are easier to treat: one looks for two parallel planes Eq. (9) such that the gap $(\alpha - \beta)/\sqrt{\bar{w} \bar{w}}$ is maximal

$$\max_{\bar{w}} G = \frac{\alpha - \beta}{\gamma}, \text{ where } \gamma^2 = \bar{w} \bar{w}$$

(17)

This is the primal quadratic programming problem, subject to the linear set of inequalities (9). In the geometric language one can alternatively ask for the minimal connector, defined as the minimal distance between any point of $S_1$ from any point of $S_0$:

$$\min_{\alpha, \beta} L^2 = (\bar{X}_1 - \bar{X}_0)^2$$

(18)
where the vectors $\vec{x}_i$ are defined in (7) and the coefficients $b_\nu$ and $a_\nu$ are convex combination coefficients. This equation is called the dual quadratic programming problem and one can show [33] that at optimality $G = L$. Obviously, if the set of examples is not linearly separable then $G$ is not defined but $L = 0$. This allows us to use the dual formulation of the problem to decide (in a polynomial number of steps) whether a set of examples (or its image) is linearly separable or not.

The optimal Perceptron used by physicists is slightly different. It is defined by maximizing the gap between the origin and the union $S_i \cup S_0$ - that is the gap between the origin and the set $S_i (i = 0, 1)$ closest to the origin. For large systems and random examples the two definitions agree in average. We prefer to work with Lambert's definition because it leads to a possibly larger gap and works for any distribution of examples.

Quadratic programming algorithms are basic tools of nonlinear optimization and fast implementations are commonly available. One example is the routine QPROG of IMSL version 10.0 implemented by Powell. Two iterative methods have been also developed recently, the MinOver algorithm [17] and the AdaTron algorithm [19]. Both can be generalized to solve the Eqs. (17,18). In our variable step implementation the MinOver is numerically more stable and only slightly slower than the AdaTron (see Appendix B for details).

The optimal Perceptron plays an important role in our discussion for several reasons. First, it makes clear that from all presented examples only the so-called active set of $N$ linearly independent edges (difference vectors) is instrumental in determining the optimal hyperplane. This fact is also emphasized by the so-called active set algorithms of quadratic optimization. This allows to define an optimal set of examples as the minimal IO set from which the optimal Perceptron can be uniquely reconstructed. Obviously, the optimal IO set contains only the active extremal points. Hence, any linearly separable function can be stored in $(N+1) \times (N+1)$ bits, independently of the precision of the $N+1$ real components of $(\vec{w}; \Theta)$. Secondly, the generalization properties of the optimal Perceptron have been calculated analytically[18] and they are superior to any other known learning method (the Fisher discriminant has not been considered yet). This hints at the following intuitive situation: the more robust against rotations and translations a hyperplane is, the better its generalization abilities. This simple intuitive principle is very helpful when dealing with more complex networks, where analytical calculations are difficult to perform.

**Multilayer feedforward networks**

For fixed architecture feedforward network the training process uses some gradient descent method, for example back-propagation [20], to minimize an error function. Unfortunately, this learning process is quite slow and even if simulated annealing is used the running time increases in a prohibitive way with the network size. This is explained theoretically by the fact that, as shown by Judd[21], the simpler problem of deciding whether a set of examples can be represented by a given feedforward network with hidden units or not is $NP$-complete. Later Blum and Rivest [22] have mapped the learning problem for a network with only two hidden units into the set splitting problem, which is known to be $NP$-complete. This means that even for such a simple architecture there are sets of examples which are very difficult to learn. $NP$-complete problems are characterized
by an exponential number of metastable states, leading usually to a slow and erratic dynamics. One should, however, keep in mind that these negative results apply to the worst case of general algorithms: for special cases one might develop fast special algorithms.

According to Valiant’s definition[13] a set of examples (or boolean functions) is learnable only if there is a polynomial time algorithm which can approximate the function from a polynomial set of examples within a controllable error, that is when the error can be made arbitrarily small. A necessary condition for learnability in feedforward networks is thus that the minimal unit representation of the function has a number of hidden units which is a power of the number of the input units and of the required accuracy. Otherwise the polynomial set of examples cannot ‘cover’ densely enough the function. Such architectures are possible only if the set of examples has enough symmetries and correlations. Assume that one tries to model these correlations by an energy functional including some interactions between the different corners of the input hypercube (the IO strings). In other words, define an energy for each boolean function which depends, for instance, on the Hamming distance of the different input strings. Intuitively, learnable functions are such that in the in $N$ dimensional input space black and white points form clusters separated by planar interfaces. This restricts the class of interactions to Ising type models with smooth (not rough) planar interfaces.

By solving Problems $1-3$ we have now the tools for a class of algorithms constructing an appropriate network from the set of examples. The main idea is to add sequentially new hidden units until the problem is linearly separable by the output unit. This type of algorithms are called growing algorithms and so far two main approaches have been used. They differ mainly in the way a new hidden unit is set up given the yet unclassified IO set. The first method is to partition in two parts the example set by a hyperplane which optimizes some appropriately defined cost function. This strategy has been used by Mázard and Nadal[10] in the tiling algorithm and by Nadal [23] and Marchand and Golea[24] for constructing tree-like architectures.

The second approach is using the theorems of the previous Section by separating in every step a box containing as many points of the same colors as possible. This separation step is repeated on the remaining example set until the linear separability test is positive. Then one uses either the solution suggested by the theorems or the (optimal) Perceptron method to set up the output unit.

As long as the boxes contain only points of the same colour, any construction scheme will lead to a good representation of the function, even when several layers are constructed in this way. The number of network architectures implementing a large set of examples is not countable (like the number of functions defined on integer points). Which networks are good and which ones present no interest? Wilhelm Occam’s razor says that among different hypotheses explaining the same facts the simpler ones should be preferred. The hint that networks of minimal size have the best generalization properties is also implied by some mathematical work[25]. This should not be surprising since minimal networks must fully incorporate all the correlations and the symmetries of the input set. In our heuristic algorithms we try thus to minimize the number of hidden units by using the maximal contraction principle. It is important to remark that finding the minimal network representing a set of examples must be at least as difficult as deciding whether a given architecture can or cannot implement without
error a set of examples. Thus the minimal architecture problem is also 
\( \mathcal{NP} \)-hard and no polynomial time general algorithm can decide whether an
architecture is minimal or not (unless \( \mathcal{P} = \mathcal{NP} \)).

The sequential learning algorithm [11] has found the known minimal
architectures for the parity problem, the mirror symmetry problem and very
good solutions (closed to the optimal bound) for random boolean functions.
A similar algorithm has been recently applied to the handwritten numeral
recognition problem with very impressive results [26].

The main problem remains to put these ideas in a mathematically more
rigorous form. Steps in the direction of optimal predicting strategies have
already been made recently [27] and, interesting enough, are also based on
a combinatorial geometrical description similar to the one used here.

In conclusion: a very nice feature of the growing network algorithms is
that they use only the information contained in the set of examples but
do not need task-dependent fine tuning of many parameters, like learning
rates, acceleration terms, starting conditions, noise levels in simulated
annealing, which is the case for back-propagation, competitive learning or
Boltzmann machines [9]. In addition, we have been able to give a simple
example where the combination of the sequential learning procedure linked
to a facet identification routine leads to a procedure to construct in worst
case only 4 times more hidden units than the best possible case for ran-
dom IO examples. The random IO problem can be considered as a test
characterizing the average worst case for any network growing algorithm.
According to this test the Perceptron based sequential learning algorithm of
Marchand, Golea and Rujan [11] is performing near optimality and is con-
sistently better than the tiling algorithm, the regular partition algorithm
or the recently suggested tree generating algorithms.

4 DATA REPRESENTATION AND OPTIMAL
ARCHITECTURES

Have we thus solved the boolean function representation problem? Not
really: more questions have been actually raised than answered. In what
follows some of these questions will be discussed, other ones will be only
stated and left at the Reader's discretion.

What is the optimal architecture if several technological or biological
constraints have to be taken into account? Consider the mirror-symmetry
problem: output a 1 only if the input string is mirror symmetric about its
center. This problem can be solved with two hidden units obtained with
the help of the stratification method [7] (Fig. 7a). However, this implies
very oblique planes, with some connection strength ratios growing expo-
entially with size. Another realization, which follows the disjunctive form
of this function and involves \( N \) hidden units of unit strength seems more
useful from a practical point of view (Fig. 7b). Yet another realization,
which uses exact identities to eliminate a hidden units by allowing for direct
connections between the input and output units is shown in Fig. 7c. This
is the minimal architecture, since the problem is not linearly separable.
Which architecture is preferable? This obviously depends on the price of
connections, processing units, etc. and cannot be answered in general.

How does the complexity of the network change with the degree of
parallelism (number of hidden layers)? Fig. 8a-c shows three possible
realizations of the parity-$N$ (here $N = 7$) problem: the fastest architecture has $N$ hidden units ($N/2$ if input-output connections are allowed), a tree-like 'canonical' realization of the parity function [28] with an $\ln N$ number of layers and hidden units and the two processor network with feedback implementing directly the recursion form of the parity function and running in $2N$ steps. The execution time can be a very important factor in the choice of the network architecture, as every VLSI designer knows. More speed means usually more transistors and here more hidden units. In biological systems one might expect the same effect when and where fast reflexes are crucial.

What classes of simple recursive functions are learnable by feedforward architectures? An interesting question is whether simple arithmetic operations, like addition or multiplication are learnable functions in Valiant's sense. Our numerical results indicate that indeed addition may require only a polynomial number of hidden units. For boolean functions symmetric under the permutation of all input variables the same results apply as above for the parity function. A systematic analysis along these lines seems most desirable.

What happens if we allow the elementary processors to be more complex? Assume for example, that the input and output units can have more colours than two. The basic strategy of the growing algorithms can be easily adapted to such cases provided the hidden units remain hyperplanes, (binary units). Several combinations of binary units have also simple geometric representations. Two parallel planes, for instance, cut the space into three regions and will correspond to a three state unit. Obviously, these cases are related to the data representation problem.

What is the fastest learning algorithm leading to a still acceptably performing network? Can one give some bounds on the expected probability of error for new examples? Can the above learning algorithms be implemented locally and iteratively? How to generalize our strategies when the problem to represent is a probability distribution and the set of examples contains also contradictory examples? These question pertain to a statistical theory of such networks, which is still in its infancy[29].

And above all, there hovers the problem which we avoided from the
Figure 8: Three architectures for the parity problem. a) The regular partition. The number of processing units is linear in $N$, running time is minimal $(2)$. b) The 'canonical' form with a logarithmic number of hidden units. The number of processing units and of hidden layers is $\log N$. Running time is also of the same order. c) The feedback network corresponding to a sequential program: $f^N_{\text{parity}} = \text{XOR} (f^{N-1}_{\text{parity}}, n_N)$. It requires only two delays and three processing units but execution time is $T \sim 2N$.

beginning: how are the feature detectors created and to what extent is their data representation a function of the classification process? This question goes deep into the problem of what the process of perception actually is. In general, the number of input units can be increased by a sparse representation of the data. An extreme example is an analogue representation: for example one can use $(1000)$ to represent 0, $(0100)$ to represent 1, etc., instead of the only 2 bits needed to store 4 values in digital code. More sophisticated external coding techniques\cite{30} can be also used to enhance the number of input units in such a way as to increase the probability that the function is linearly separable. A layer of hidden units with random couplings might play a similar role\cite{31}. The question is then: when is the addition of input units helping to minimize the total number of processors ? What is the impact on learning times, since it is always more expensive to search a large input configuration space than a smaller one. At some point these question should be faced in a global way: from the choice of 'sensorial' detectors up to the form of the classifier system and/or the autoassociative features, including their mutual interactions. Nature has solved this problem in its own inimitable way. With the introduction of adaptable Random Markov Field methods in the early stages of feature detection \cite{32} and a better understanding of the geometry of learning we are now closing up on a statistical mechanical information processing machine with exciting abilities.
References


[27] Haussler, D., Littlestone, N. and Warmuth M. K. Predicting \{0,1\}-Functions on Randomly Drawn Points to be published


[31] Gallant, S. L. Neural Networks. 3 (1990) 191-201


APPENDIX A: SEQUENTIAL LEARNING THEOREM

A sequential learning procedure is defined as follows[11]. Assume one has some method to find hyperplanes (here hidden units) such that on one side of the hyperplane there are only points of the same colour \( \sigma \). By adding such a plane to the set of hidden units we say that we have cut the later set of points (they are removed from the example list). Choose the sign of the hidden unit (this can be achieved by negating all connections and the threshold if necessary) so as to match that of \( \sigma \). This is not really necessary but will simplify our discussion. Assume now that after \( H \) such steps the remaining set of IO examples is found to be linearly separable. This means that we have created \( H \) hyperplanes (hidden units) which partition the unit input hypercube in at least \( H + 1 \) boxes. Each hidden unit separates a subset \( \Omega_i \) of \( M \) points from the input IO set and assumes the value \( s_i = \sigma \) when a member of the subset is processed. The subset \( \Omega_{H+1} \) is defined to be the second set of points obtained by linear separation in the last step of the algorithm and the colour of its output is \( s_{H+1} = -s_H \). We give now a recursive algorithm to calculate the connection strengths \( u_1, \ldots, u_H \) of the hidden units to the output unit \( \sigma \) and the threshold \( \Theta \).

When presenting the whole IO set to the input one can easily see that the form of the activations \( \zeta = (\zeta_1, \ldots, \zeta_H) \) on the hidden layer will have the following form

\[
\begin{align*}
\zeta^{(1)} &= (s_1,*,*,\ldots,*) & \text{if } \xi \in \Omega_1 \\
\zeta^{(2)} &= (-s_1,s_2,*,\ldots,*) & \text{if } \xi \in \Omega_2 \\
& \vdots \\
\zeta^{(k)} &= (-s_1,-s_2,\ldots,s_k,*,\ldots,*) & \text{if } \xi \in \Omega_k \\
& \vdots \\
\zeta^{(H)} &= (-s_1,-s_2,\ldots,-s_k,*,\ldots,s_H) & \text{if } \xi \in \Omega_H \\
\zeta^{(H+1)} &= (-s_1,-s_2,\ldots,-s_k,\ldots,-s_H) & \text{if } \xi \in \Omega_{H+1}
\end{align*}
\]

where \( \xi \) are the input examples and \( \zeta \) their image on the first layer. The wild card * signals that the corresponding component can have either a +1 or a -1 value.

Choose the threshold in the following form:

\[
\Theta = -\sum_{i=1}^{H+1} u_is_i
\]  

(19)

where \( u_{H+1} \equiv 1 \) by definition. In order to prove that the space of hidden unit activations is linearly separable one has to satisfy

\[
\sigma^{(i)} = s_i = \text{sgn} \left( \sum_{j=1}^{H} u_j \zeta_j^{(i)} - \Theta \right)
\]  

(20)

for all classes of inputs \( \in \Omega_i, i = 1, 2, \ldots, H \). These equations are equivalent
to the set of inequalities

\[ s_i \sum_{j=1}^{H} u_j \zeta_j^{(i)} - s_i \Theta \geq 0 \]  

(21)

Start with class \( \Omega_{H+1} \). Using the general form of \( \zeta^{(H+1)} \) and the definition of the threshold \( \Theta \) one can see that the choice \( u_{H+1} = 1 \) satisfies this inequality. Now repeat the procedure for the class \( \Omega_H \). A simple calculations give an inequality involving only \( u_H \) (as a function of \( u_{H+1} \)). Continue this procedure. In step \( k \) one recovers a set of \( M_{H+1-k} \) inequalities for the component \( u_{H+1-k} \). Choosing as solution the most stringent inequality, the procedure is continued until the whole vector \( \bar{u} \) is obtained. A solution always exists: the choice

\[ u_k = 2^{k-H-1} \]  

(22)

provides a sufficient condition due to the property

\[ 2^N > \sum_{i=1}^{N-1} s_i 2^i, s_i = \pm 1 \]  

(23)

Whenever the actual condition for \( u_k \) has the form \( u_k > -1 \) the choice \( u_k = 0 \) will imply that the \( k^{th} \) hidden unit is not connected to the output unit and will result in a simplification of the network. For regular partitions one can show that this choice is the algebraic correspondent to the 'sleep' phase[1,2].

A better solution than the one obtained by stratification can be obtained by using the optimal Perceptron learning algorithm discussed in Appendix B. However, the 'stratification' solution is a constructive one and thus runs much faster.

APPENDIX B: A VARIABLE-STEP MINOVER ALGORITHM

In this Appendix we discuss a variant of the MinOver algorithm [17] for the optimal Perceptron defined in Section 3. Recall the definition of the direction \( \bar{w} \) connecting a point from \( S_1 \) to a point from \( S_0 \):

\[ \bar{w} = \bar{X}_1 - \bar{X}_0 \]  

\[ \bar{X}_1 = \frac{\sum_{\nu \in \nu_1} b_\nu \zeta^{(\nu)}}{\sum_{\nu \in \nu_1} b_\nu} \]  

\[ \bar{X}_0 = \frac{\sum_{\nu \in \nu_0} a_\nu \zeta^{(\nu)}}{\sum_{\nu \in \nu_0} a_\nu} \]

where the coefficients \( a_\nu \) and \( b_\nu \) are nonnegative. The map corresponding to the direction \( \bar{w} \) is then

\[ h^{(\nu)} = \bar{w} \zeta^{(\nu)} = \frac{\sum_{\mu \in \nu_1} b_\mu Q_{\mu,\nu}}{\sum_{\mu \in \nu_1} b_\mu} - \frac{\sum_{\mu \in \nu_0} a_\mu Q_{\mu,\nu}}{\sum_{\mu \in \nu_0} a_\mu} \]  

(24)

where \( Q_{\nu,\mu} \) is the correlation (overlap) matrix

\[ Q_{\nu,\mu} = \zeta^{(\nu)} \zeta^{(\mu)} \]  

(25)
The length of the \( \vec{w} \) is given by
\[
L^2 = \vec{w} \cdot \vec{w} = \frac{\sum_{\nu \in I_0} b_\nu h^{(\nu)}}{\sum_{\nu \in I_1} b_\nu} - \frac{\sum_{\nu \in I_0} a_\nu h^{(\nu)}}{\sum_{\nu \in I_0} a_\nu}
\] (26)

It is this length that we want to minimize. In our definition of the map there are two parts, one contains the projections of positive examples, while the other consists of the projections of the negative example points on the direction \( \vec{w} \). In the MinOver algorithm one shall update only the coefficients of the black point corresponding to the minimal field \( h^{(\min)}_i \) of all positive examples and of the white point corresponding to the maximal field \( h^{(\max)}_0 \) among all negative examples. In other words, we push the white points on the negative direction from below and the black points on the positive direction from the top. Therefore, after finding the two indices \( (\min) \) and \( (\max) \) we update the vectors of coefficients \( a_\nu \) and \( b_\nu \) as following:
\[
\begin{align*}
\tilde{a}_\nu &= a_\nu + \eta \delta_{\min, \nu}, & \nu \in I_0 \\
\tilde{b}_\nu &= b_\nu + \eta \delta_{\max, \nu}, & \nu \in I_1
\end{align*}
\] (27)

Using the definitions above we first calculate in second order the new values of the map:
\[
(h^{(\nu)})' = h^{(\nu)} + y(1 - y)[Q_{\min, \nu} - Q_{\max, \nu} - h^{(\nu)}] 
\] (28)

where we assumed that initially \( \sum_{\nu \in I_0} a_\nu^0 = \sum_{\nu \in I_1} b_\nu^0 = S^0 \) and according to our updating rule the norm factor of \( a_\nu's \) and \( b_\nu's \) remains equal during the iteration. \( y = \eta / S \), \( S \) being the value of the normalization factor before updating. Iterating this procedure but now for \( L^2 \) one obtains after some simple algebra
\[
L'^2 = L^2 + 2y(h^{(\min)} - h^{(\max)} - L^2) + y^2 L^2 
\] (29)

Since the gap is defined as
\[
G = \frac{h^{(\min)} - h^{(\max)}}{\sqrt{\vec{w} \cdot \vec{w}}} \Theta(h^{(\min)} - h^{(\max)}) 
\] (30)

where \( \Theta(x) \) is the step function, the previous equation has the form
\[
L'^2 = L^2 - 2y L(L - G) + y^2 L^2 
\] (31)

From the duality theorem of quadratic programming[33] \( G = L \) only at optimality, which proves that for small enough \( \eta \) this procedure converges to the optimal solution. The scale of the optimal step size can be also calculated and reads
\[
\eta \sim S(L - G) 
\] (32)

which is typical for a quadratic problem.