TWO SUFFICIENT CONDITIONS FOR CONVERGENCE IN MULTILAYER PERCEPTRONS

Pál Ruján

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

1 LEARNING FROM EXAMPLES

The actual interest in neural networks is motivated partly by developments in neurobiology, partly by the need for theoretical models of parallel computation. No doubt, the neurobiological approach is a strategic, long term basic reasearch program. Engineering applications, on the other hand, raise also interesting questions which are easily cast in a mathematical framework. In this talk I am going to present some results obtained by M. Marchand, M. Golea (Department of Physics, Ottawa University), and myself, acknowledging that this contribution might have little to say to those interested mainly on the human brain. I was asked by the organizers of this conference to be as 'mathematical' as possible. Being a physicist, I can only guarantee that our results are true — but not necessarily rigorous[1]

So what are those famous "intelligent machines"? I asked many times this question but never got a straight answer. Here is an educated guesss: Consider the system shown schematically in Fig. 1. A physical signal (electromagnetic, sound wave, etc.) is detected by some receptors (the preprocessor box) and is coded by reducing redundance and noise. We assume that on output the preprocessor provides a series of binary signals: each output bit is said to be a 'feature' detector. This binary form is not a strong restriction but helps in formalization. Preprocessing is in itself a difficult task and is solved here simply by assuming that a solution exists and has been provided by the preprocessing department.

The next stage is represented by the "Adaptive System" box. The input is a $\vec{s} \in \{0,1\}^N$ binary vector, while the output is a $q-$dimensional vector $\vec{\sigma} \in \{0,1\}^q$, usually $q << N$. While the input units are activated on the presence of some characteristic features (color, form, texture, etc.) the output units represent the class to which such an input pattern most probably belongs. The adaptive system has internal degrees of freedom represented by the parameter set $p \in \mathcal{P}$ and a *learning algorithm* which is an operator $\mathcal{L}: \mathcal{P} \to \mathcal{P}$. The learning operator is activated during the training process, when examples are presented at the input of the adaptive box and a 'teacher' corrects the output classification associated to each input signal. To every binary output unit σ_i one can associate a boolean function $f_i(\vec{s}) = \sigma_i = 0, 1$. Without loss of generality, one can consider thus only the case of a single output unit. The protocol of learning from examples can be formalized as follows [1].

[1] Paraphrasing R. Baxter's definition of 'exact but not rigorous' results

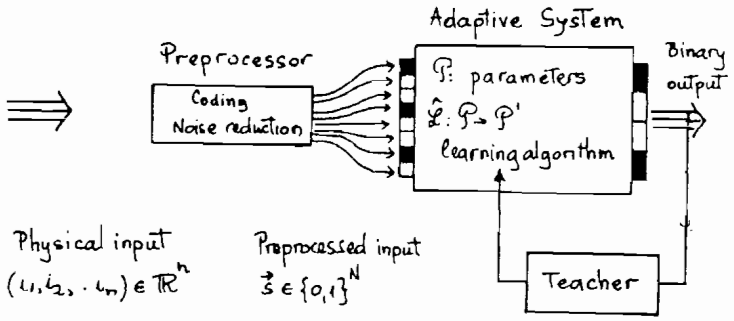

Figure 1: *A schematic picture of a learning machine*

Consider a class of boolean functions defined on $\{0,1\}^N$, $f \in \mathcal{F}$, $f(\vec{s}) = \{0,1\}$.

1. Generate at random a (typical) target function $f_T(\vec{s}) \in \mathcal{F}$.
2. Sample from some fixed probability distribution $P(\vec{s})$ at random and independently M input vectors, $\left\{\vec{\xi}_\nu\right\}_{\nu=1}^{M}$. For simplicity, in what follows we shall use always binary strings as input vectors, although all mathematical statements made later on can be easily generalized to real vectors.
3. Provide the correct output for each input example, $\sigma^{(\nu)} = f_T(\vec{\xi}^\nu)$.
4. Assume that the training algorithm $\mathcal{L}: \mathcal{P} \to \mathcal{P}'$ will succeed with probability $1 - \delta$ to provide an approximation $f_A(\vec{s})$ in the sense that
5. The error probability defined as

$$P_E = \sum_{\{\vec{s}:\ f_A \neq f_T(\vec{s})\}} P(\vec{s}) < \epsilon \qquad (1)$$

6. A class of functions $\mathcal{F}$ is *learnable* if there exists a learning algorithm $\mathcal{L}$ plynomial in $1/\delta$, $1/\epsilon$, N and $size(f_T)$ satisfying the above conditions, where $size(f_T)$ is the size of encoding f_T in binary code.

Recently, interesting results have been obtained regarding the minimal size of networks giving a controllable generalization ability, or vice versa, on

the number of training examples needed to achieve a desired accuracy of prediction[2]. We will return to this question later on.

2 FEEDFORWARD NETWORKS

Figure 2. shows schematically the neuronal structure of the retina in the macaque monkey. Without going into details, it is clear that the architecture is designed in such a way as to permit a fast information flow from the receptors towards the fovea. Perhaps not incidentally, fast numerical processors share the same property, namely a massive parallel, feedforward architecture without feedback loops. Such systems are optimized for processing speed and their dynamics is very simple. In what follows, we shall work with McCullogh - Pitts[3] binary neurons, which are activated only when the weighed sum of their inputs is larger than some threshold:

$$\sigma = \mathrm{sgn}\left(\sum_i w_i s_i - \Theta\right) \tag{2}$$

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 $(s_1, \ldots, s_N)$. $s_i = \pm 1$. Θ 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 an engineering point of view our basic elements are thus threshold gates. The geometric interpretation of Eq. (2) is that every elementary processor corresponds to a N-dimensional hyperplane

$$\vec{w}\vec{s} = \Theta \tag{3}$$

partitioning the $N-$dimensional configuration space into two classes: black points for $\sigma = 1$ and white ones for $\sigma = -1$. In what follows the terms 'hyperplane' and 'unit' are used interchangeably, associating the vector $\vec{w}$ and the threshold Θ with unit σ. In many models the activation values σ, $s_i \in (0, 1)$ are continuous variables and the step-like activation function (2) is a sigmoid. Physically, such situations correspond to a probabilistic updating scheme and the output variable σ is then essentially a time or ensemble averaged activation [4]. The geometric picture remains valid, provided that one works with slabs of finite width instead of geometrical hyperplanes.

An architecture consisting of a single output processor unit is a simple classifier better known under the name of *Perceptron* [5]. The class of linearly separable functions $f_T \in \mathcal{LS}$ is defined as follows:
there $\exists\ \vec{w}$ and threshold Θ such that

$$f_T(\vec{\xi}^\nu) = \mathrm{sgn}(\vec{w}\vec{\xi}^\nu - \Theta),\ \forall \nu = 1, 2, \ldots M \tag{4}$$

Obviously, simple Perceptrons can represent only the class of 'linearly separable' boolean functions $\mathcal{LS}$ and many practical tasks fall outside this class [6]. Adding one or more layers of so-called hidden units 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. In what follows only connections between nearest neighbour layers will be allowed.

Most of the actual interest on such neural networks is due to the discovery

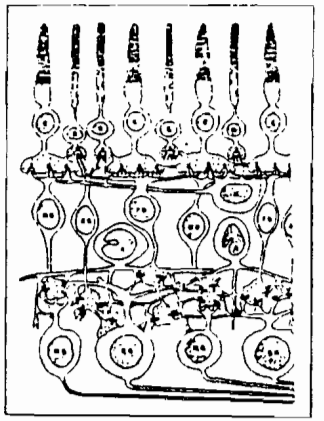

Figure 2: *The neuronal structure in the retina of the macaque monkey (from 'Vision', P. Buser and M. Imbert, Herrmann, Paris, 1987)*

of the backpropagation algorithm [7]. This approach attempts to solve the learning task using a gradient descent type algorithm on the error function

$$\mathrm{Error} = \sum_{\nu=1}^{M} \left[f_T(\vec{\xi}^\nu) - f_A(\vec{\xi}^\nu)\right]^2 \tag{5}$$

by evaluating the gradient vector

$$(\vec{\nabla}\mathrm{Error})_{ij} = \frac{\partial \mathrm{Error}}{\partial w_{ij}} \tag{6}$$

with a neat technical trick. When applied to small size and appropriately structured networks, the backpropagation algorithm delivered some spectacular network solutions and induced scientific and technological mass-enthusiasm. Unfortunately, it became subsequently clear that the efficiency of this approach could not be sustained as the size of the network (and examples) increased to levels needed in most practical applications. However, there is nothing particularly wrong with backpropagation. The root of the problem is that deciding whether the boolean function f_T can be represented by an a priori fixed (but otherwise arbitrary) network is $\mathcal{NP}$-complete [8] even if the number of hidden units is as small as 2[9]. From a 'physical' point of view this means that the problem is as difficult as finding the ground state of a three dimensional spin glass. The search is hindered in both cases by an exponential number of local (metastable) minima and consequently a very slow dynamics. Any attempt at improving the backpropagation algorithm would only change a prefactor in front of an exponential and is thus futile. Claims like 'our algorithm runs 100 times as fast as backpropagation' have in themselves no value, except when is interested on upscaling the size of a small network by a factor of two.

The only way to 'escape' the unpitiful laws of algorithmic complexity is to change the rules of the game. As we shall see, this does not mean necessarily cheating. One way of proceeding is to start without any preconception about the structure of the network and to construct it as to best fit the set of examples. The mathematical basis for this approach forms the bulk of my talk. An additional possibility is to allow the network to ask questions.

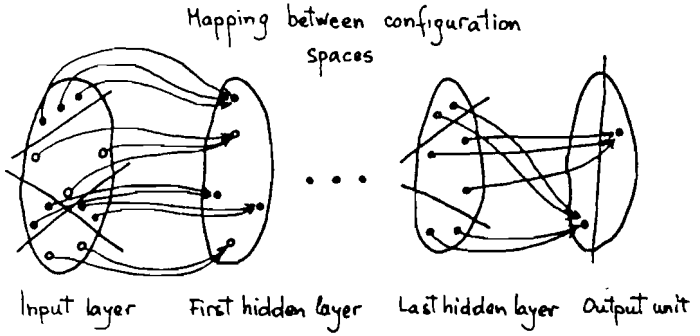

Figure 3: *Information processing in feedforward networks a) The input configuration space is partitioned in several regions by the hidden units of the first layer b) The first hidden unit configuration space: every region of the previous input space is mapped into a single point c) The last hidden layer defines a linearly separable function. The black and white points are separated by the output unit*

thatis to query the value of f_T on any possible configuration of the input space. This gives the additional advantage of allowing for the optimization of the generalization ability of the network. An example in this direction has been added at the end of this article.

3 INFORMATION PROCESSING

In order to explain our network growing algorithms we must first understand how information is processed in feedforward networks. A (hyper)plane divides the N-dimensional space (unit cube) into two regions. Two planes divide it into four (the planes intersect each other) or three regions (the planes do not intersect). Note that when the set of example vectors defines a compact subspace of $\mathcal{R}^N$ not intersecting planes are not necessarily parallel.

Consider now the situation shown in Fig. 3. Points represent (example)vectors in some configurational space and their color denotes the desired output value (black for +1 and white for -1). The first layer of hidden units will partition the configuration space of input vectors into several regions.

A moment of thought reveals that all points lying into a given region will be mapped into a single point (vector) on the configuration space of the first hidden units. This map is a many-to-one map and therefore contracts. Another moment of reflection will make clear that this contraction leads to errors when points of different colors are mapped into the same point. Sometimes we might WANT to include a few errors. This might be the case when the examples have strong uncertainties either on their positions or on their classification. Another case is when one (or few) points hinder(s) the coalescence of two large regions. Putting aside these special considerations, a necessary condition for an error free training is that the partition of the configuration space with hyperplanes results on regions populated by the

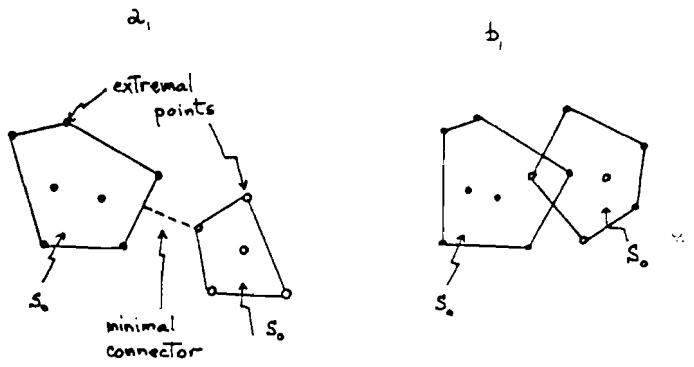

Figure 4: *a) Linearly separable case. Note the minimal connector. b) The two convex hulls overlap and the problem is not linearly separable.*

same kind (class) of examples. An additional benefit of this simple picture is that it hints at the following trivial theorem: if a constructive algorithm defines in every step (from one layer to another) a contractive map, then it will necessarily converge. This is the basic idea for proof of convergence in most growth algorithms.

Let us now ask a little bit more ambitious question, namely: how to partition the *last* hidden layer so as to *guarantee* linear separability? After answering this question one can restrict oneself without loss of generality to architectures with one single hidden layer. From the point of view of execution time this would be optimal. Other considerations, like a lack of enough processing units, might favor a many-layer solution.

Our main results are two sufficient conditions for linear separability of a *hidden* layer, one is based on a geometrical construction, the other one is an algebraic proof called stratification[6]. Before going into details, let me first shortly discuss how to decide when is a set of examples NOT linear separable.

4 DECIDING NON LINEAR SEPARABILITY

Definition The *convex hull* S of the set of examples $\vec{\xi}^\nu \in X \in \mathcal{R}^N$ is defined as a *convex combination* of the vectors $\{\vec{\xi}^\nu\}$:

$$S: \left\{ \vec{x} \in S \mid \vec{x} = \sum_\nu \alpha_\nu \vec{\xi}^\nu;\ \sum_\nu \alpha_\nu = 1;\ \alpha_\nu \geq 0 \right\} \tag{7}$$

Definition Let $I_\bullet$ define the set of indices ν such that $f_T(\vec{\xi}^\nu) = 1$ and $I_\circ$ the set of indices pertaining to the white examples. Let $S_\bullet$ be the convex hull of all black vectors $\nu \in I_\bullet$ and similarly let $S_\circ$ be the convex set of white exampless. Then we have the following obvious

Theorem $f_T \in \mathcal{FS}$ iff $S_\bullet \cap S_\circ = \emptyset$. See Fig. 4 for a graphic explanation.

Definition The *minimal connector* of $S_\bullet$ and $S_\circ$ is the minimal distance between the two sets and is defined as

$$\min_{\alpha,\beta} \quad (\vec{m}\vec{m}) \quad , \tag{8}$$
$$\vec{m} \quad = \quad \vec{r}_\bullet - \vec{r}_\circ, \vec{r}_\bullet \in S_\bullet, \ \vec{r}_\circ \in S_\circ$$

This minimization problem is a ***convex quadratic optimization*** problem with linear constraints on the convex coefficients α_ν and β_ν defining the vectors $\vec{r}_\bullet$ and $\vec{r}_\circ$, respectively. Obviously, if the minimal connector has length nill, the problem is not linearly separable. Since the convex quadratic problem $\in \mathcal{P}$[10], this is a reasonable way to decide linear separability. In practice, non linear separability is always decided faster than the optimum of(8) because the points $\vec{r}_\bullet$ and $\vec{r}_\circ$ fall rather freely towards each other and the constraints do not interfere.

It seems worthwhile to note that the *dual* of the problem Eq.(8) corresponds to positioning two parallel planes inbetween $S_\bullet$ and $S_\circ$ so that their distance is maximal. This 'optimal' Perceptron has been first introduced to my knowledge by Lambert[11] and a possible implementation algorithm can be found in Vapnik [12] and [13]. Physicists are using as 'optimal' Perceptron (or Perceptron with maximal stability) the minimal distance between origin (bitstrings have in this formulation coordinates $\xi_i = \pm 1$) and the convex set of *all* examples. For random patterns and $N \to \infty$ these definitions agree on the average but in general the physicist's definition is more restrictive.

5 REGULAR PARTITIONS

Definition Two hyperplanes do not intersect on the unit hypercube iff they partition the set $\{\vec{\xi}^\nu\}$ in three classes only.

Definition A *box* is a convex polytope whose boundaries are either hidden unit hyperplanes or face(t)s of the unit hypercube AND contains only examples of the same kind (class).

Lemma No mapping error is introduced by tiling with boxes only.

Definition *Regular partitions* are tilings of the unit hypercube with hyperplanes such that

1. Only boxes result from tiling,
2. The hyperplanes do not intersect pairwise. As a result, every hyperplane is at the boundary of exactly two boxes.
3. These two boxes contain input vectors of different color (classes).

Now we are in the position of stating our first sufficient condition:

Theorem [14]: The configuration space of hidden units partitioning the input space as a regular partition is linearly separable.

Proof Consider Fig. (5) for a schematic explanation. The circle represents the compact subspace of $\mathcal{R}^N$. The lines represent projections of the hyperplanes and the little arrow at the end the direction of the normal.

The corresponding neuron is active for every point falling in that side of the plane and inactive otherwise. Points represent examples colored according to their desired output. On the right side we see the so-called internal representation, thatis the map of the input configuration into the first hidden

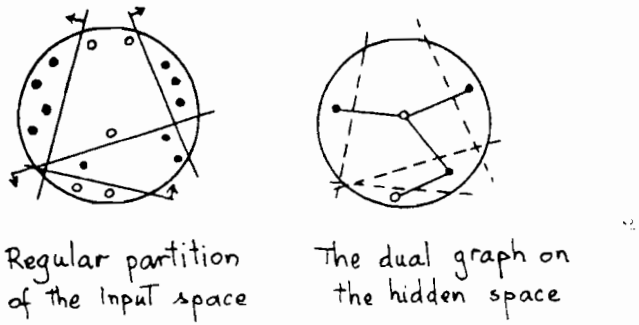

Figure 5: *a) The input configuration space is partitioned regularly. The hyperplanes do not intersect within the convex hull of all examples. The direction of each normal vector is indicated by a small arrow. b) The hidden configuration space. The dashed lines are a reminder of the box boundaries in a). The nearest neighbor points define the bipartite graph use in the proof.*

unit configuration space. Note that in our example the number of hidden units is 4. The position of the hidden planes is shown by dashed lines in the right hand side. The proof folows now the following steps:

1. Construct a graph in the hidden layer space by connecting points (vertices) originating from boxes with common boundaries with an edge. This graph has the following properties:

   1a The graph is a tree. This follows from the fact that the planes do not intersect pairwise. The edges of the graph must intersect at least one boundary of the corresponding box. Since the boxes are convex, the edges will intersect exactly one hyperplane. Since the planes do not intersect within the input space, no loop is possible.

   1b The graph is a bipartite tree. This means that the color (desired output) associated with the vertices of the tree alternate along any path on the graph. This property folows from the third condition imposed on regular partitions, namely that the color of the points on the two sides of any box facet corresponding to a hyperplane is not the same.

2. This can be also easily visualized on the hidden configuration space as follows. Associate the origin (0,0,0,0) with the position of the central white point (any other point will do as well). Since all points connected by edges were in neighboring boxes and every edge crosses only one hyperplane, moving along and edge will result into the change of only one coordinate (in our example the three black points connected to the point at origin will have the coordinates (1000), (0100), and, say, (0001) in the hidden configuration space. Different edges cross different hyperplanes, thus in the above coordinate system the edges point all in different directions. The last white point must then have the coordinates (0011).

3. Now it is easy to complete the proof: consider the middle points of these edges. The graph has in total $H + 1$ edges (H is the number of

hidden units) and the median points of all edges define H non coplanar points, which, in turn, define uniquely an H dimensional hyperplane in the hidden unit configuration space (the output unit). The only thing which remains to be proved is that in one side of the plane all points are black and on the other side they are white. This follows from the bipartite property. Assume we choose the direction of the normal vector so that the (0000) point is correctly classified. Moving along any edge we will cross the output unit hyperplane (exactly at the midpoint of the edge). Since the graph is a bipartite graph the whole set of vertices will be correctly classified. Q.E.D..

This type of solution has some very appealing properties: the output hyperplane can be found by solving a very sparse linear system of equations with a particularly simple tree like structure. The solution is such that with an appropriate scaling (remember that the equation defining a hyperplane is homogeneous) the $\vec{u}\vec{\zeta} = \pm 1$, where ζ_i is the activation of the i^{th} hidden unit and u_i its connection strength to the output unit. Regular partitions produce therefore a minimal entropy solution for the output unit, as follows from the Fisher discriminant or the least square error method (see [15]).

Although attractive, this method has a snag regarding algorithmic implementation: it is rather cumbersome to keep track of the pairwise non-intersection of planes. We thus looked for a more general sufficient condition.

6 CONVERGENCE OF SEQUENTIAL LEARNING

A sequential learning procedure is defined as following[16]. 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 color σ. By adding such a plane to the set of hidden units the later set of points is cut (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 σ. 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 Ω_i of M_i points from the input IO set and assumes the value $s_i = \sigma$ when a member of the subset is processed. The subset Ω_{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 σ and the threshold Θ.

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

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

where $\vec{\xi}$ are the input examples and $\vec{\zeta}$ 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:

$$\Theta = -\sum_{i=1}^{H+1} u_i s_i \tag{9}$$

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

$$\sigma^{(i)} = s_i = \text{sgn}\left(\sum_{j=1}^{H} u_j \zeta_j^{(i)} - \Theta\right) \tag{10}$$

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

$$s_i \sum_{j=1}^{H} u_j \zeta_j^{(i)} - s_i \Theta \geq 0 \tag{11}$$

Start with class Ω_{H+1}. Using the general form of $\vec{\zeta}^{(H+1)}$ and the definition of the threshold Θ one can see that the choice $u_{H+1} = 1$ satisfies this inequality. Now repeat the procedure for the class Ω_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 $\vec{u}$ is obtained. A solution always exists: the choice

$$u_k = 2^{k-H-1} \tag{12}$$

provides a weak sufficient condition due to the property

$$2^N > \sum_{i=1}^{N-1} s_i 2^i, \; s_i = \pm 1 \tag{13}$$

A better solution is obtained by following more closely the actual recursion. 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 the network can be simplified.

This theorem is very useful for setting up algorithms constructing a network from the given set of examples. These algorithms have been described elsewhere[16, 13] and can be considered as good heuristic methods for finding a subminimal architecture (finding THE minimal architecture is obviously $\in \mathcal{NP}$-hard class).

It is perhaps worthwhile mentioning that similar constructive algorithms have been used recently for the problem of recognizing handwritten numbers[18].

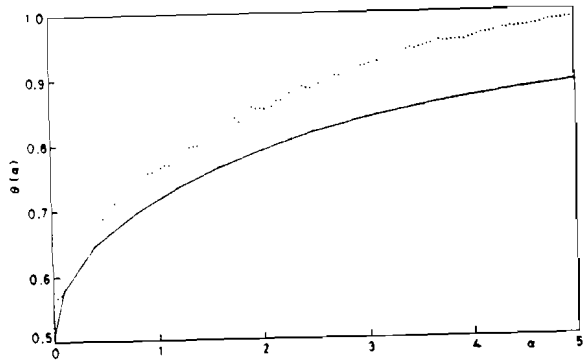

Figure 6: *Generalization probability for the 'optimal' Perceptron learning rule as a function of* $\alpha = \frac{M}{N}$. *The lower curve are analytical results for randomly generated examples, the upper curve is a numeric simulation for the case when the network is allowed to ask questions with a maximal information content.*

The results are of the same quality level as the much more elaborate AT&T network, **designed** by first class experts and *trained* by backpropagation.

7 OPTIMIZING GENERALIZATION ABILITY

The reason behind searching for minimal architectures is that the generalization probability of a minimal architecture is better. This follows from the intuitive argument that a minimal architecture must represent in the most compact form the symmetries and correlations between the different input and output vectors. Mathematical results confirm these expectations[2].

On the other hand, the generalization ability depends on the set of examples. One can consider the examples as constraints on a minimization problem (remember the discussion of the minimal connector problem). However, not every constraint is really important: only the so-called active (extremal) constraints play a role in determining the neuronal architecture. The *minimal example set* is the set of active examples allowing the full reconstruction of the network given a specific learning algorithm.

One can expect, therefore, that allowing the algorithm to query the value of the target function f_T on specific input examples can speed up a lot the correct construction of the network. I add here a little example which W. Kinzel and myself have published in the time between the summer school and the writting of this manuscript [17]. Consider a function $f_T \in \mathcal{LS}$ and allow the network to ask questions. Now, under the assumption that f_T is linearly separable, the network can obtain a maximal amount of information by asking the output value of some input vector lying exactly (or as near as possible) to the border of the actual classification. This simple idea has proved to be extremely efficient: see Fig. (6) for the generalization probability obtained for the optimal Perceptron method described in this article obtained from randomly generated examples and from optimal queries.

It seems to me that in the near future we shall see a much awaited breakthrough: polynomial algorithms creating network architectures with mathematically guaranteed generalization probabilities.

References

[1] Valiant L. G. *Comm. of the ACM* **27** pp. 1134-1142 (1984)

[2] Baum, E. B., Haussler, D. *Neural Computation* **1** (1989) 151-160

[3] McCullogh W. S. and Pitts W. (1943) *Bull. Math. Biophys.* **5** 115-133

[4] Burnod, Y. and Korn, H. *Proc. Natl. Acad. Sci. USA* **86** (1989) 352-356

[5] Rosenblatt, F. *Psychoanalytic Review* **65** (1958) 386

[6] M. L. Minsky, S. Papert: *Perceptrons: An Introduction to Computational Geometry* Cambridge Ma., MIT Press, 1969 and 1988

[7] D. E. Rumelhart, G. E. Hinton, R. J. Williams: *Nature* **323**, 533-536 (1986)

[8] Judd S. *Proc. IEEE First Conference on Neural Networks San Diego 1987* Vol. II pp. 685-692 (IEEE Cat. No. 87TH0191-7)

[9] Blum A. and Rivest R. *Advances in neural information processing systems* pp. 494-501 Touretsky D. (Ed.) Morgan Kaufman, San Mateo, Ca., 1988

[10] Kozlov, M. K., Tarasov, S. P., and Khachian, L. G. *Soviet Math. Dokl.* **20** (1979) 1108-1111

[11] Lambert, P. F. *Methodologies of Pattern Recognition* Watanabe M. S. (Ed.) 359-381 Academic Press, New York, 1969

[12] Vapnik, D. . *Estimation of Dependencies from Empirical Data* Springer Verlag, 1982

[13] Ruján, P. in "Neural Networks", Proceedings of the IX. Sitges Summer School, L. Garrido (Ed.), Springer Verlag, 1990

[14] Ruján P. and Marchand M. *Complex Systems* **3** (1989) 229-242 , *Proceedings of IJCNN 1989, Washington* Vol. II 105-110

[15] S. Watanabe *Pattern Recognition: Human and Mechanical* John Wiley & Sons, New York, 1985

[16] Marchand, M., Golea, M. and P. Ruján *Europhys. Lett.* **11** (1990) 487-492

[17] Kinzel, W. and Ruján P., *Europhys. Lett.* **13** (1990) 473-477

[18] Knerr, S., Personnaz, L. and Dreyfus, G. to appear in *Neurocomputing: Algorithms, Architectures and Applications* NATO ASI Series, Springer Verlag 1990