A Geometric Approach to Learning in Neural Networks

Pál Ruján

Institut für Festkörperforschung der Kernforschungsanlage

Jülich, Postfach 1913, D-5170 Jülich, Federal Republic of Germany

Mario Marchand

Department of Physics, University of Ottawa

34 G. Glinsky, Ottawa, Canada K1N 6N5

Abstract

We present a geometric view of how information is processed in feedforward networks of linear threshold units. The role played by the hidden units is described in terms of the complementary notions of *contraction* and *dimension expansion*. A tight sufficient condition for the representation of an arbitrary Boolean function is given by introducing *regular partitionings*. Learning is interpreted as a general search procedure seeking a custom made minimal architecture for a given but otherwise arbitrary function or set of examples. A new class of learning algorithms is introduced, which provide a suboptimal solution in a *polynomial* number of steps. The results of several experiments on the storage and on the rule extraction abilities of three-layer Perceptrons are presented. When the input patterns are strongly correlated, simple neuronal structures with good generalization properties emerge.

1 Introduction

Connectionist models of cognition attempt to give a microscopic description of how ensembles of neurone-like units process distributed information in parallel[1]. Simple processing units mimicking neurons are connected into a multilayered network with *visible* input and output layers and one or more internal layers of *hidden units*. The presence of hidden units is essential if the network is to perform complicated tasks, since single layer *Perceptron-like* devices without hidden units have only limited abilities[7]. Feedforward networks are useful because they have a simple and fast dynamics and do not require special synchronization.

The actual interest on such models stem from two of their inherent properties: execution speed and learning ability. *Grosso modo*, a computing machine is calculating several, many variable Boolean functions defined by the program. The question is how and to what extent can be a recursively defined function executed in parallel.

Learning is the ability to correctly represent and solve a task without following any particular program except a general learning strategy. We distinguish three classes of learning processes, corresponding to different levels of cognition development. During *unsupervised* learning a self-organization process takes place so as to create a correct internal representation of the input vector distribution. Usually this process leads to the creation of feature detectors and keeps the information loss minimal[4]. After the creation of feature detectors the network learns (from examples) how to categorizes a large amount of input signals into a small number of classes. This *supervised learning* process implies a strong data compression by capturing the *symmetries* present in the patterns to be learned. When new, unlearned patterns are processed, the output should be consistent with the observed symmetries. This *rule extraction* ability is required for pattern recognition and inductive generalizations. Finally, the categories (abstract mental concepts) emerging from the supervised learning process form the basis for *problem solving*. In what follows we consider only the problem of supervised learning and use always the word learning in this sense.

The main problem with the known learning algorithms[9,2] is that they are defined for a *fixed* architecture. Indeed, it is the human designer of such a network who must guess the minimal number of hidden units and who imposes certain regularities on the connections. However, a given task cannot be solved by just any network. Deciding this *satisfiability* question is at the core of the difficulties encountered in different approaches to the learning problem and leads to the recent result that learning is an $\mathcal{NP}$-complete problem [11]. This implies that the learning time increases at least exponentially with the system size. In addition, algorithms like back-propagation are not guaranteed to converge at all and do get stuck quite often in local minima, especially when a minimal network is used[9,13].

Following the path taken by Minsky and Papert [7] we took a geometric view at the multilayer Perceptron [6]. A simple picture of the possible architectures realizing a given Boolean function is given in Section 2. Accordingly, learning is defined as the search for a minimal architecture given some technological or biological constraints. This approach is close to the theory of learnable by "arbitrary circuit design" advocated by Valiant [18] and is explained in detail in Section 3. We introduce a simple "greedy" algorithm which finds good solutions in a *polynomial* number of steps. This method allows us to make a realistic assessment of data storage and rule extraction properties of three-layer feedforward networks and to discuss how they depend on the correlations between the different input patterns (Section 4).

Our procedure does not require any human interference during learning. In contrast, back-propagation, competitive learning or Boltzmann machines[1], all need a *task-dependent* fine tuning of many parameters, like learning rates, acceleration terms, starting conditions, noise levels in simulated annealing, etc.

2 The geometry of multilayer networks

Consider a feedforward network of simple logical threshold McCulloch-Pitts units connected to each other by inhibitory or excitatory connections. Each unit calculates its output as

$$n^{out} = \lim_{m\to\infty} \frac{1}{2}[1 + \tanh mx], \qquad x = \sum_{\text{all inputs } i} w_i n_i - \Theta \quad (1)$$

where the output and input activations are binary variables, $n^{out}, n_i = \{0, 1\}$. The limit $m \to \infty$ corresponds to the usual step function. Then Eq. 1 has a simple geometrical interpretation: n^{out} is 1 on one side of the plane $\vec{w}\vec{n} = \Theta$ and 0 on the other side. The components of the normal vector $\vec{w}$ are the incident connection weights w_i and the threshold of the unit is Θ.

If the network has several output units each output unit can be considered as a Boolean function of N_i variables, where N_i is the number of input units. We call such a function *incomplete* if there are input configurations for which the desired output is not specified. The input (or switching) hypercube has dimension N_i and every corner corresponds to a possible input configuration. Its color is black if the corresponding output value is 1, white for output values 0, and is not colored at all (unmarked) when no output value is given.

The input cube for the XOR and the parity-3 functions are shown in Fig. 1. From the simple geometric interpretation of Eq. 1 it follows that a problem can be solved with only one output unit *iff* the set of white points can be separated from the set of black points by a (hyper)plane. A simple geometric way to decide linear separability is to construct the convex hull (the smallest convex polytope incorporating all points in

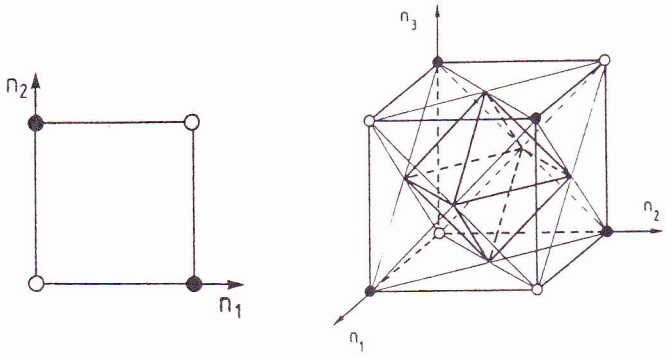

Figure 1: *The unit cube representation of the XOR and parity-3 problem. For the XOR S_0, S_1 are the two diagonals, P is their intersection point. In the parity-3 cube the two tetrahedrons formed by the white and black set of points intersect on the octahedron displayed in heavy lines.*

question) of the two sets of vertices, S_0 and S_1. Next construct the intersection $P = S_0 \cap S_1$, which is also a convex polytope.

Boolean functions for which P is empty are called linear separable functions (l.s.f.). The number of incomplete linear separable functions of N variables and M specified output values $B_M(N) \leq 2\sum_{k=0}^{N} \binom{M-1}{k}$ [3]. For complete functions $B_{2^N}(N) \leq 2^{N^2}/N!$, compared to the total number of 2^{2^N} functions. However, for large N and small M the probability that a function is l.s.f. is 1 if $M < 2N$ and 0 if $M > 2N$ [3].

For l.f.s. we define the "best separating plane" by the following geometric construction: take two parallel hyperplanes whose distance can be varied. Fit the two planes between S_0 and S_1 and rotate-translate them simultaneously as to maximize the distance between them. A third parallel plane halving this distance is most robust against noise affecting the connections and the threshold. Let us also remark that if a function (input hypercube) is linearly separable, so are all its faces. Conversely, it is necessary but not sufficient that all faces must be

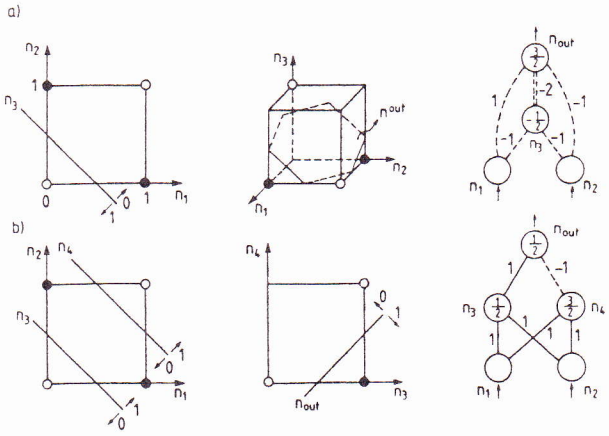

Figure 2: *Two ways of removing the intersection P. a) Adding one more dimension to the input square by adding a hidden unit . The cube shows all possible configurations of the three units and the function is separable. The network has direct connections from the input to the output layer. b) Partition the input square into boxes containing only points of the same color. Only one corner per box will be activated (marked) in the hidden layer. The network does not have direct links between input and output units.*

l.s.f. for the hypercube to be l.s.f.

On the contrary, when the intersection $P = S_1 \cap S_2$ is not empty, the space is not linearly separable. In such cases we must introduce new internal (hidden) units, so as to remove the intersection P. Two ways of doing this are illustrated in Fig. 2 for the XOR function. Let us call *box* a polytope whose boundaries are hyperplanes or facets of the original input hypercube and which contains at least one corner. We consider here only architectures whith no direct connections between the input and the output unit. This is not a strong restriction since using the last hidden unit also as output (which is always possible) one regains an architecture of the type shown in Fig. 2a.

The input hypercube must be fully partitioned into boxes containing corners of the same color (unmarked corners do not count). On the configurational space of the next layer every box will be represented by a single configuration. We call this property *contraction*, since the total number of possible configurations will shrink from layer to layer. It is not allowed to imbed corners of different colors into the same box because they will be projected onto the same output value and the network will fail.

At this stage one possible strategy to follow is the *maximal contraction* principle: search for partitionings with allowed maximal contraction. The configuration space of a hidden layer is an incomplete Boolean function and after a large contraction it may happen that $M_h < 2N_h$, where N_h is the number of hidden units and M_h the number of distinct hidden configurations. Hence, with a large probability the hidden space is linear separable. Otherwise stated, every contracting procedure is guaranteed to converge into a possible network solution. This strategy can be implemented in different ways. A recent independent work using as ingredient the usual Perceptron learning rule leads to networks with only a few layers[8].

We have carried this analysis one step further by giving a

tight sufficient condition for a partitioning to be linearly separable (we call such partitionings *legal partitionings*). Fig. 3a shows a minimal partitioning with parallel (1,1,1) planes of the parity-3 problem, while Fig. 3b is a "grand mother" type partitioning, obtained by constructing a box for every black point. The grand mother partitioning solves always the *memorization* problem[10] but requires an excessively (exponentially) large amount of resources. Fig. 3c is the configuration cube of the hidden units (internal layer). We use the same notation as in Fig. 3a, except that unmarked corners can never be reached with the chosen allocation of hidden units. The corresponding picture for the grand mother solution involves a four dimensional cube not shown here. Figs. 3d and 3e are schematic representations of these regular partitionings. The points, each representing a whole cluster of corners, are connected by a bi-

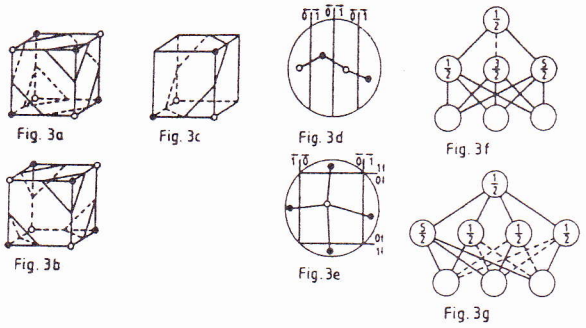

Figure 3: *Processing information in feedforward networks (see text). Full (broken) lines have strength +1 (-1).*

partite tree. Finally, in Figs. 3f, 3g, these partitions are expressed as conventional networks solving the parity-3 problem.

What are the common properties of these two solutions? Let us call a *regular partitioning* of the N_i-dimensional hypercube a partitioning with hyperplanes (hidden units) such that:

1. Every region contains vertices of the same color
2. Every hyperplane separates corners of different colors
3. The hyperplanes do not "intersect" inside the hypercube

(An "intersection" is void if it does not contain corners).

We now show that every regular partitioning is separable by a single output unit (hyperplane of the hidden space). The proof is quite simple: imagine a point inside every region, representing a given activation pattern of the hidden units. Construct a graph connecting all nearest neighbour points (see Fig. 3d and Fig. 3e). The N_h edges of this graph must pass through a single plane delimiting that region and no other edge can cross this plane (from property 3 and convexity). All nodes of the graph must have neighboring nodes of the other color (from property 2). Hence, the graph is a bipartite tree. Each edge points in a different direction, because only the corresponding hidden unit changes its activation along that edge. The median points of the edges form a set of noncolinear N_h points, defining uniquely a hyperplane in the N_h-dimensional hidden space. In addition, this plane cannot have 0 connections (every hidden unit must be connected to the output). An example of an output plane is shown in Fig. 3c for the parity-3 problem.

Although the regular partitionings are not the only way hidden units may form a separable space (the class of *legal partitionings*), we have not found yet examples of legal partitionings which solve a given problem with less planes than a regular one. In fact, among all legal partitionings with the same number of hidden units regular partitionings have the maximal contraction. However, there are cases (*e.g.* the mirror symmetry problem - [7]) where the regular partitioning requires hyperplanes whose coefficients increase exponentially with the size ('oblique planes'). Such hyperplanes are hard to learn and unfeasible from a technological or biological point of view.

An *optimal partitioning* is a legal partitioning which minimizes a cost function assigning some price tag to every unit and every connection and defines the minimal architecture needed to implement the predicate under consideration. Thus, the problem of supervised learning amounts to constructing a good, possibly optimal architecture. A learning process which allocates resources from a pool of neurons is biologically quite plausible. In the "attention state" some groups of neurons might have their strongly inhibitory thresholds increased and further tuned during the learning process, constructing a kind of temporary, virtual architecture. The fact that the cortical responses are highly enhanced in the state of "attention" is well established experimentally[12].

3 The learning procedure

This simple geometric interpretation of how three-layer Perceptrons process information suggests a class of algorithms for constructing regular partitionings. An obvious choice is a greedy algorithm which searches for planes having corners of only one color on one of its sides. Cutting the largest such cluster out of the hypercube will leave behind a complicated but still convex body. One continues this "chop-off" procedure until all vertices of the hypercube are separated. Obviously, this implies an enumeration of all planes. We decided to restrict ourselves to a "minimal set", where the connections (weights) can have only the values $w_i = \{-1, 0, +1\}$ and the thresholds are $\Theta = -\frac{2N_- - 1}{2}, -\frac{2N_- - 1}{2} + 1, \ldots, \frac{2N_+ - 1}{2}$ where $N_{-(+)}$ counts the number of -1's (+1's) in $\vec{w}$. All such planes intersect the unit hypercube but do not contain any vertex. Their total number is easily calculated by observing that all normal vectors point from a vertex to the diagonally opposite vertex, from an edge to the diagonally opposite edge, etc., for all faces of the hypercube. Counting this number with the weight given by the possible number of thresholds one finally obtains $K3^{K-1}$ (the number of possible inputs is 2^K). This set is nevertheless able to represent any Boolean function as a grand mother type regular partitioning or—since multientry AND and OR gates are *special cases* of our set—as a minimal disjunctive representation[14,15].

Thus in our three-layer architecture the units are connected layer wise with all other units through inhibitory and excitatory connections of unit strength (0 corresponds to a missing connection). This seems reasonable from a biological point of view and is advantageous also for digital implementations of such networks. Although one is not guaranteed to find any longer the optimal partitioning, we still expect solutions with good data compression properties.

The formal description of the greedy algorithm is the following:

1. Construct a list with all allowed planes (list 1)
2. Using all input/output (IO) examples, calculate the number of black and white points on both sides of all such planes (4 classes of points in list 2)
3. Start greedy procedure: search for the plane with a maximal number of black (white) points on one side of the plane, such that no points of opposite color are present
4. Once a plane has been chosen, delete all planes intersecting it from the list 1 and update list 2 by excluding the separated points. Keep one representative configuration for each separated cluster of corners (list 3).
5. Check for separability: if not separable, go to step 3.
6. Rerun the set of representative IO (list 3) and delete all spurious hidden units.
7. Construct the output unit and check the function.

Step 6. is needed because the greedy algorithm constructs a tree, but not necessarely a bipartite one. In fact, with a limited lookahead it is not possible to foresee situations when the separation of a small cluster of black points, for example, will allow two clusters of white points to merge together.

The main computational load of the greedy algorithm is the classification of planes according to all possible inputs and is of order $3^{N_i} N_{patt}$, where $\max N_{patt} = 2^N$. Note that backpropagation[9] requires at least an exponential number of steps in N_{patt}. We have developed also a full enumeration algorithm in order to find the optimal solution for this set of planes. However, since the number of regular partitionings is extremely large, exact enumeration can be performed at the moment only for small networks. A variant of the greedy algorithm constructing networks with multiple units is already operational. By the time of the publication of this article, we hope to have finished our work on an algorithm whose running time is minimal $\sim O(N_{patt})$.

4 Results

When running the parity-K problem on the full input/output set we obtain the optimal solution of K, parallel (1,1, ...,1) (or rotated) planes (see Fig. 3). The number C of bits needed to store a network with K hidden units is $C \leq K(K+1)\log_2 3 + (K+1)\log_2 K$. Compared to the 2^K bits needed to tabulate the function this shows a tremendous data compression. One of the most attractive features of our algorithm is this ability of constructing simple architectures for symmetric tasks. For functions symmetric in all their variables $N_h \sim N_i^2$ is an upper bound in architectures with one hidden layer[5].

In the next set of experiments, we used our learning procedure to find networks with $N_{inp} = 6$ which execute a randomly chosen predicate. The average storage requirement of 200 different random predicates was 226 bits with the variance of 36 bits. In average 15.8 hidden units were needed to learn the tasks. Compared to the 64 bits needed for tabulation this result is disappointing but probably there are no evolutionary advantages related to learning random functions. As shown below, when the input patterns are correlated, the storage requirement decreases drastically.

Another much emphasized feature of neural networks is the ability of rule extraction. After training feedforward networks with an incomplete set of input/output data, new input sets are presented. In many cases the network has good generalization properties by correctly classifying the new inputs. In this form, however, the question of rule extraction ability is an ill-posed problem, since its answer depends on the choice and the size of the training set inasmuch as on the nature of the correlations between different input patterns. For a random Boolean function, for example, it is absurd to talk about rule extraction. What is usually meant by generalizations is in fact a kind of *continuity hypothesis* regarding the function to be learned. In order to gain a better understanding of this problem, we have performed the following experiment. Consider the set of all Boolean functions. Each function can be viewed as a possible configuration of the $2^{N_{inp}}$ binary variables $\{n_i\}$ (colors, spins) defined on the vertices of the unit hypercube of dimension N_i. We want to control to what extent the color of a vertex depends on the color of the nearest neighbour vertices. This can be achieved by introducing an *energy* for each Boolean function as

$$E(\{n_i\}) = -J \sum_{<i,j>} (2n_i - 1)(2n_j - 1) \qquad (2)$$

which is the Ising Hamiltonian. The sum runs over all $<i,j>$ pairs of vertices connected by an edge of the hypercube. The probability of finding a given Boolean function with energy E is given by the Boltzmann distribution

$$\text{Prob}(\{n_i\}) \sim \exp(-E(\{n_i\}) \qquad (3)$$

J plays the rôle of an inverse temperature: when $J > 0$ the interaction is attractive, like in a ferromagnet, while for $J < 0$ it is repulsive (antiferromagnetic). For $|J| = 0$ all Boolean functions have an equal weight, while for $|J| \to \infty$ one obtains a ferromagnetic (all vertices have the same color, separable case) or an antiferromagnetic (parity function) structure on the hypercube. At low temperature the regular structures break into some large clusters requiring only a few additional hidden planes. Presenting a few points from each cluster will create the desired architecture and the system will be 'predictable'. We illustrate this fact in Fig. 4 by plotting the average ratio of correct classifications as a function of the size of the training set at several temperatures. It is clear that the generalization ability decreases drastically at higher temperature (random patterns). In large networks this sudden deterioration is triggered by a second order phase transition of the underlying Ising model. Not surprisingly, a reduction in the average number of hidden units $< N_h >$ needed to learn the task is also observed when J is increased. Indeed, for $J = 0.2,\ 0.4,\ 0.6,\ 0.8$, one obtains $< N_h >= 8.46 \pm 1.91,\ 6.93 \pm 1.92,\ 2.34 \pm 1.72,\ 1.56 \pm 1.23$, respectively, for $N_p = 40$.

Robustness is another useful property of neural networks. This means that the performance of the net is not impaired by random changes affecting locally connections, thresholds, or whole units. Our architectures work also with a smooth activation function Eq. 1, where now $m \sim 1$. Since the solution depends only on the sign of the connection, they will be unperturbed by small local noise. Our approach suggests also a new repairing mechanism by reallocation of tasks to other, healthy units.

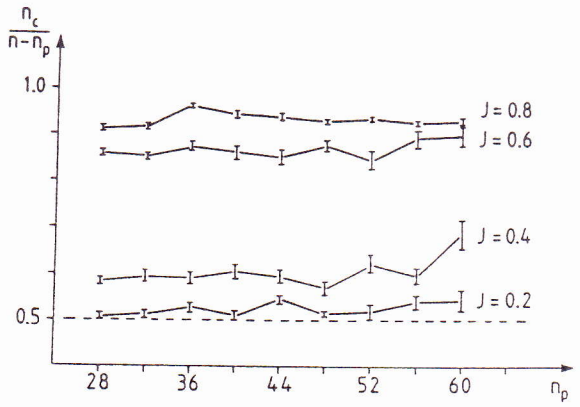

Figure 4: *Generalization ability in networks with 6 input units. At each temperature we have generated by a Monte Carlo method 100 different Boolean functions and applied the greedy algorithm to each. From the total of $n = 64$ patterns n_p randomly chosen patterns have been learned. From all the remaining patterns n_c are correctly classified. The error bars represent the variance around the average value of $n_c/(n - n_p)$ as a function of n_p and J. A random guess corresponds to $n_c/(n - n_p) = 0.5$.*

5 Discussion

During our development of the greedy algorithm we have ourselves made a few mistakes. First, once a cutting plane has been chosen, all other planes intersecting it within the unit hypercube must be removed from the set. When this is not done properly such intersections form loops in the graph of hidden units (Figs. 3d-e) and can be corrected only by additional layers of internal units. An interesting feature of the greedy algorithm is the "sleepphase when the units not satisfying property 2 are removed by rerunning all representative configurations, followed by the disconnection of a rather large set of hidden units (in typical cases this means a 30-40% reduction). Although fully aware of the oversimplifications inherent in our model, we cannot refrain from speculating that the multilayered structure of the neocortex originated perhaps as a correction mechanism of topological mistakes in poorly learned tasks. Similarly to the correction phase of the greedy algorithm, REM sleep[16,17] might be the way simple errors are eliminated from the memory by reviewing a seemingly uncorrelated and distorted subset of the learned set of patterns (dreams), used to identify and disconnect the spurious internal units.

Although we are confident that our method can be used for many practical problems up to medium size, we acknowledge that this line of research has still a long way to go. The principle of maximal contraction leads sometimes to solutions of no practical interest, like in the example of mirror symmetry [7]. From this point of view our main problem is the partitioning of a task into a hierarchy of well separated subtasks. Given some primary network structures there is no problem to combine them into more complex ones, as probably happens in evolutionary processes. Looking from the top (output) level, however, there is a vast number of possible partitionings of the given task into subtasks and the search seems hopeless.

References

[1] D. E. Rumelhart, J. L. McClelland *Parallel Distributed Processing* Vol. 1-2 , Cambridge Ma., Bradford Books, MIT Press, 1986

[2] T. Grossman, R. Meir and E. Domany: "Learning by Choice of Internal Representations", Weizmann Institute preprint 1988

[3] R. O. Winder: "Bounds on Threshold gate Realizability" *IRE Trans. Electron. Comp.* **EC-12** (5) pp. 561-564 (1963)

[4] R. Linsker: "From Basic Network Principles to Neural Architecture - I—III" *Proc. Natl. Acad. Sci. USA* **83** I: pp. 7508-7512, II: pp. 8390-8394, III: pp. 8779-8783 (1986)

[5] P. M. Lewis II and C. L. Coates: *Threshold Logic* John Wiley & Sons, New York, 1967

[6] P. Ruján and M. Marchand: "Learning by Activating Neurons: A New Approach to Learning in Neural Networks" submitted to *Complex Systems* 1988

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

[8] M. Mézard and J.P. Nadal: "Learning in Feedforward Layered Networks: The Tiling Algorithm", L.P.S.E.N.S. preprint 1989

[9] D. E. Rumelhart, G. E. Hinton, R. J. Williams: "Learning Representations by Back-propagating Errors" *Nature* **323**, 533-536 (1986)

[10] J. Denker, D. Schwartz, B. Wittner, S. Solla, J. Hopfield, R. Howard, L. Jackel: "Automatic Learning, Rule Extraction, and Generalization" *Complex System* **1**, pp. 877-922 (1987)

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

[12] S. Hillyard, T. Picton and D. Regan: "Sensation, Perception and Attention: Analysis using ERPs" in *Event-related brain potentials in man* pp. 223-231 E. Callaway *et al* (Eds.), New York, Academic Press, 1978

[13] G. Tesauro, B. Janssens: "Scaling Relathionships in Backpropagation Learning" *Complex System* **2** pp. 39-44 (1988)

[14] W. V. Quine: " A Way of Simplyfying Truth Functions" *The American Mathematical Monthly* **62**, pp. 627-631 (1955)

[15] E. J. McCluskey Jr.: " Minimization of Boolean Functions" *Bell System Technical Journal* **35** 1417-1444 (1956)

[16] F. Crick, G. Mitchinson: "The Function of Dream Sleep" *Nature* **304**, pp. 111-114 (1983)

[17] J. J. Hopfield, D. I. Feinstein, R. G. Palmer: "Unlearning"has a Stabilizing Effect in Collective Memories" *Nature* **304** pp. 158-159 (1983)

[18] L. G. Valiant: "A Theory of the Learnable" *Comm. of the ACM* **27** pp. 1134-1142 (1984)