Complex Systems **6** (1992) 301–314

Learning from Examples and Generalization

J. Dente
Laboratório de Mecatrónica, Instituto Superior Técnico,
Av. Rovisco Pais, 1096 Lisboa Codex, Portugal

R. Vilela Mendes
CERN, Theoretical Division,
CH-1211, Genève 23, Switzerland

Abstract. We analyze the issue of generalization in systems that learn from examples as a problem of representation of functions in finite fields. It is shown that it is not possible to design algorithms with uniformly good generalization properties in the space of all functions. Therefore the problem of achieving good generalization properties becomes meaningful only if the functions being studied belong to restricted classes.

We then propose the implementation of systems endowed with several distinct (biased) strategies, allowing the exploration and identification of the functional classes of learning problems. Two such strategies (polynomial learning and weighed majority rule) are developed and tested on the problems of even-odd and two-or-more clusters.

1. Introduction

Whenever the algorithm required to program a specific task is not known or the task itself involves uncertain features, learning from examples seems to be a sensible approach. In contrast to the construction of approximate representations, which remain forever committed to a set of simplifying assumptions, learning from examples allows for systematic improvement. It suffices to increase the size of the training set, and the more examples are presented the wiser the system becomes.

In concept learning [1], the generalization capability of a learning system is judged by its ability to take into account a number of specific observations and then extract and retain their most important common features. When learning from the examples in a training set, the issue of generalization therefore concerns the problem of how well the system reacts to situations that are not presented during the training period. This problem has been addressed by many authors [2–13] in the last few years. The work has been carried out mostly in the context of specific architectures—neural networks or classifier

systems, for example—or assuming some properties of the functions to be learned. Wolpert [4, 5], for example, has developed a theory of generalization that takes the point of view of generalization as surface fitting. This is also the point of view of the regularization theory [14]. However, the notion that the process being learned can somehow be guessed or anticipated, as opposed to being completely random, does not necessarily imply that it fits in the class of smooth functions.

In this paper we attempt a discussion of this issue in a general setting. In section 2 we develop a few concepts to study generalization as a problem of representation of functions in a finite algebraic field. This is done in a way that is independent of particular implementations or assumptions about the classes of functions being studied. The main issues to be addressed concern the questions of whether it is possible to classify learning algorithms on the basis of their generalization performances and whether a general-purpose algorithm can be constructed, that is, an algorithm with good generalization capabilities for arbitrary functions. The results concerning these issues are mostly negative in the sense that we conclude that, in the space of all functions, all algorithms are essentially equivalent, and a truly general-purpose algorithm is bound to have an average generalization rate of zero. Therefore, good generalization performances are to be expected only if the functions being studied belong to particular classes.

As a consequence of the general study we were led to suggest that, instead of looking for generic well-performing algorithms, one should equip the learning systems with several distinct learning strategies, each one biased toward a particular class of functions. Allowing the system to switch between these distinct strategies in the course of the learning process might optimize the performance and identify the functional class of the problem. In sections 3, 4, and 5 two such strategies are described and tested on two classical problems.

2. Learning from examples and generalization

Learning is a term that is used in a variety of contexts and applied to many different processes. The general notion we want to retain here is the one of learning as the capacity to adapt, through interaction with the environment, to perform a specific task. More precisely we characterize a *learning system* as having a certain number of *inputs* through which it receives information from the external world and a certain number of *outputs* through which it acts on the environment. By coding the appropriate transducers, the inputs and outputs may always be considered numbers from some algebraic field.

The systems that we will be concerned with are not those that *learn by being taught* what rules to follow, but those that learn by being presented with a certain number of correct input–output pairs (I_i, O_i). The set of input–output pairs (I_i, O_i) presented during the training process is called the *training set.* If all that is required from the system is an accurate reproduction of the learned examples, then the system is called a *data basis.* If the system is provided with some a priori knowledge (rules of inference) in such a way

that any input, even if not contained in the training set, uniquely provides the desired output, then it is called an *expert system*. If, however, no a priori rules are supplied and one nevertheless expects the system to react meaningfully to inputs not contained in the training set, we say that the system possesses *generalization capacity*.

We now quantify the notion of generalization of a learning system. Let there be N inputs and M outputs that may take p different states, p being a prime number. This implies no loss of generality because, for any other number q of possible states, we may always assign a subset of input terminals to represent the digits of the number q in the p-basis. A system with no a priori constraints should be able to represent any $F_p^N \to F_p^M$ function $f_i(x)$, F_p being the prime field of characteristic p [15]. Considering each of the components f_i separately, all one really has to deal with is the representation problem for an $F_p^N \to F_p$ function. There are $(p)^{p^N}$ such functions.

A *learning algorithm* associates with each training set T of K $(K \leq p^N)$ input–output pairs $\{(x_i, y_i)\}$, where $x_i \in F_p^N$ and $y_i \in F_p$, a unique function $g_T(x)$ such that $g_T(x_i) = y_i$ for all $(x_i, y_i) \in T$. The function $g_T(x)$ may be one among $(p)^{p^N - K}$ functions.

For each function f to be learned and for each training set T, we define the *global generalization rate* $t(T, f)$ by

$$t(T,f) = \frac{1}{p^N - K} \sum_{x \notin T} \gamma\left(g_T(x) - f(x)\right) - \frac{1}{p} \tag{2.1}$$

where $\gamma(0) = 1$ and $\gamma(x) = 0$ for $x \neq 0$. $t(T, f)$ measures the rate of correct predictions $g_T(x) = f(x)$ for inputs not contained in the training set subtracted from the probability of being correct by chance. If $t(T, f) > 0$ we say that there is *global generalization* for the pair (T, f).

However, in a situation of continuous learning, it is not very important to know how the system will react to all possible inputs not contained in the training set. The important issue is to know how it will react to the next input x' because, given the chance of comparing the proposed output $g_T(x')$ with the actual value $f(x')$, the system may incorporate this new piece of information and readapt itself to the enlarged training set. This leads to the notions of *sequential learning* and *sequential generalization*. Each ordered sequence $S = \{(x_i, y_i = f(x_i)),\ i = 1, \ldots, K\}$ of input–output pairs is called a *learning instance* for the function f. The point of insisting on sequences is that the reaction and performance of the learning algorithm may change with the order in which the training pairs are presented. For each learning instance a *sequential learning* (SL) *algorithm* generates a sequence of functions g_i such that $g_i(x_i) = f(x_i)$ for $i \leq K$.

Consider a chain of learning instances $S_j = \{(x_i, y_i = f(x_i)),\ i \leq j\}$. The *sequential generalization rate* is defined by

$$t_S(S_K, f) = \frac{1}{K - 1} \sum_{i=1}^{K-1} \gamma\left(g_i(x_{i+1}) - f(x_{i+1})\right) - \frac{1}{p} \tag{2.2}$$

Defining an SL algorithm corresponds to specifying the function g_i chosen in response to all possible pairs (x_i, y_i) at each level of a sequence of length p^N. At level K there is still a choice of $(p)^{p^N-K}$ functions compatible with the chain $S_K = \{(x_i, y_i),\ i \le K\}$, and there are $\prod_{j=0}^{K-1} p(p^N - j)$ branches when all possible sequences are taken into account. Therefore there are

$$N_{\mathrm{SL}} = \prod_{j=1}^{p^N} \left((p)^{p^N-j}\right)^{p^j p^N (p^N-1)\cdots(p^N-j+1)} \tag{2.3}$$

different sequential learning algorithms. This is, of course, a very large number even for relatively small N and p. It would seem desirable to characterize a small set of "efficient" algorithms, if they exist, or at least to divide them into equivalence classes according to some performance criterion.

If $t_S(S_K, f) = -1/p$ for all K—that is, if the algorithm A fails at each step for the complete chain S and the function f—we say that f is a *complete failure* (CF) *function* for (A, S). Conversely, if the prediction is correct at every step, f is called a *complete success* (CS) *function* for (A, S).

For a chain $S = \{(x_i, y_i)\}$, denote by $S^{(1)} = \{x_i\}$ the sequence of input values and by $S^{(2)} = \{y_i\}$ the sequence of output values. Then *there are* $p(p-1)^{p^N-1}$ *CF functions and* p *CS functions for each pair* $(A, S^{(1)})$. Given an algorithm, the choice of the functions g_i that are generated at each step is completely determined by the pairs (x_j, y_j) with $j < i$. To construct a CF function it suffices to define $f(x_{i+1}) \ne g_i(x_{i+1})$, and for a CS function $f(x_{i+1}) = g_i(x_{i+1})$. Taking into account the p-multiplicity in the first step and possible function choices in the others, the result follows.

The conclusion is that all algorithms are CS- and CF-equivalent and therefore, in the space of all functions, there is no absolute criterion for selecting a particular class of algorithms as having better performance than any others. Similar conclusions are obtained if instead of CF and CS functions one considers the classes of *well-represented* and *badly represented* functions, defined as those for which $t_S(S, f) > 0$ and $t_S(S, f) < 0$, respectively.

Another notion that, again in the space of all functions, is self-defeating is the notion of an *unbiased general-purpose algorithm.* This would be an algorithm that is equally efficient for all possible functions. On the average any such algorithm would be equivalent to a stochastic algorithm that at each step K of the learning process chooses at random one among the $(p)^{p^N-K}$ functions compatible with the training set. But for such an algorithm the average generalization rate is zero, and the algorithm has no generalization capacity.

In conclusion, the only possibility for obtaining meaningful generalization performance in practical algorithms lies in the hope that somehow the functions that one encounters in the natural sciences belong to restricted classes as opposed to being completely generic in the whole function space. If that is the case it makes sense to endow the system with several distinct biased learning strategies and eventually provide for a switch of strategies on the basis of the results obtained during the learning process. If one of

the alternative strategies works successfully, then not only have we obtained a performing system, but at the same time we have identified the class of functions that is at play in the studied process. If, conversely, neither of the biased strategies works, we at least gain the information that the process is of a new kind and might be used to characterize a new functional class.

In the following sections we explore two (biased) learning strategies that rely on different assumptions concerning the class of functions being learned. In defining the generalization criteria we concern ourselves merely with the algorithms that generate the functions at each step of the learning process. Concrete hardware implementations of the learning concepts, through neural networks and/or logical arrays, are possible but will not be discussed in this paper. The following learning criteria are considered:

1. The system reproduces correctly the input–output patterns of the training set and configures itself to represent the "simplest" function that is compatible with the learned patterns.
2. The system reproduces the input–output patterns of the training set and, when presented with a new input, makes a "smooth" interpolation between the learned output patterns.

Of course, as emphasized by the quotation marks, the notions of simple function and smoothness have to be defined precisely. In (1) and (2) the criteria are similar to those used in experimental science where the data is fitted using a low-degree polynomial, a trigonometric function, or a minimal curvature constraint. In the case of real-valued functions, fitting data to a simple function leads to a solution very similar to the minimal curvature constraint. In finite fields, however, the corresponding requirements produce different results.

An $F_p^N \rightarrow F_p$ function is a linear combination of polynomials, namely [15]

$$f(\vec{x}) = \sum_{\vec{y} \in F_p^N} f(\vec{y}) \left(1 - (y_1 - x_1)^{p-1}\right) \cdots \left(1 - (y_N - x_N)^{p-1}\right) \qquad (2.4)$$

where $\vec{x} = (x_1, \ldots, x_N)$, $\vec{y} = (y_1, \ldots, y_N)$, and the sum runs over all p^N elements of F_p^N. The functional simplicity criterion used in section 3 states that, at each stage of the learning process, the system should represent the polynomial using the smallest degree and smaller number of monomials that accurately reproduce the learned examples. If, however, the smoothness assumption (2) is preferred, we have the requirement that the represented function, for an input pattern not belonging to the training set, interpolate between the values taken for nearby input patterns. This requires the introduction of the notion of distance between input patterns. Notice, however, that not all input terminals may be equally significant. For example, the patterns 01111 and 10000 have a Hamming distance [16] of 5 whereas 10101 and 00101 have a Hamming distance of 1. However, if the patterns are considered binary representations of real numbers, the former are much closer to each other than the latter. For simplicity, in our algebraic treatment we

will consider the characteristic of the finite field to be sufficiently large for the input terminals to be independent; for example, each input refers to a different physical variable, as opposed to being digits in the coding of the same variable. Hence all inputs are equally significant. This notion is made precise by requiring that the definition of distance be symmetric in the input variables. Alternatively we might include in the distance definition different weights for each input variable. The symmetric approach of enlarging the finite field is simpler for the theoretical discussion, although for concrete electronic implementations binary coding and asymmetrically weighed distance functions are more convenient.

3. Functional simplicity and polynomial learning

The criterion used in this section for the construction of the learning algorithm requires that, at each step of the learning process, the simplest polynomial be represented that is compatible with the input–output patterns of the current training set. By simplest polynomial we mean a polynomial with the smallest number of monomials of lowest degree.

If the training set has K patterns, a straightforward although not very efficient method is to consider a polynomial with K monomials and obtain its coefficients from the solution of a $K \times K$ linear system in F_p. The polynomial is obtained by the following recursive process: Each time a new pattern is added to the training set, a monomial

$$x_{i_1}^{\alpha_1} x_{i_2}^{\alpha_2} \cdots x_{i_k}^{\alpha_k}$$

is added, where $\{i_1, i_2, \ldots, i_k\}$ is the set of inputs where the new pattern is non-zero, and the exponents $\alpha_1, \alpha_2, \ldots, \alpha_k$ are as small as possible, compatible with non-duplication of monomials already included. For example, if no pattern has previously appeared in which $\{i_1, i_2, \ldots, i_k\}$ is the non-zero set, then $\alpha_1 = \alpha_2 = \cdots = \alpha_k = 1$. Otherwise we examine the powers α_i in the corresponding monomials and add one unit to one of the α's.

It would be convenient to develop algorithms such that, instead of having to compute anew all of the coefficients, one uses the previously polynomial, adds a new term, and possibly changes a few coefficients. Below we develop one such algorithm for the F_2^N case.

Consider then the binary case, $\vec{x} \in F_2^N$. Define $Z_{\{i_1,\ldots,i_k\}}^N$ to be the set of N-patterns that have zeros in the complement of the positions $i_1, \ldots, i_k$:

$$Z_{\{i_1,\ldots,i_k\}}^N = \{\vec{x} : i_n \notin \{i_1, \ldots, i_k\} \Rightarrow x_{i_n} = 0\} \tag{3.1}$$

We may now rewrite (2.4) in monomial form:

$$f(\vec{x}) = \sum_{\{i_1,\ldots,i_k\}} x_{i_1} \cdots x_{i_k} \left(\sum_{\vec{y} \in Z_{\{i_1,\ldots,i_k\}}^N} f(\vec{y}) \right) \tag{3.2}$$

In the set of N-patterns a partial order relation is defined by

$$\vec{x} \leq \vec{y} \qquad \text{iff} \qquad x_i = 1 \Rightarrow y_i = 1 \tag{3.3}$$

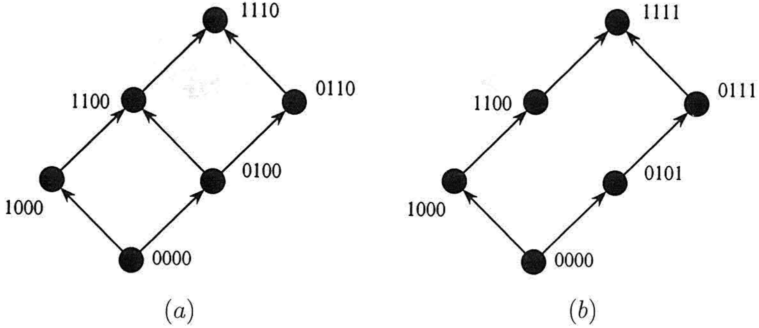


Figure 1: Sets of patterns connected by the order relation (3.3). The arrows point toward larger patterns.

For example, if $N = 4$, $a = 1100$, $b = 0110$, and $c = 1110$, then $a \leq c$ and $b \leq c$, but a and b are not comparable by this order relation.

For an element $\vec{y}$ of a given (training) subset T of F_2^N, we define $\text{sub}_T(\vec{y})$ to be the set of all least upper bounds of elements smaller than $\vec{y}$ in T:

$$\text{sub}_T(\vec{y}) = \{\vec{\xi}(i) : \vec{\xi}(i) \neq \vec{y};\ \vec{x} \leq \vec{y} \Rightarrow \vec{x} \leq \vec{\xi}(k) \text{ for some } k\}$$

For example, if

$$T = \{1110, 1100, 0110, 0100, 1000, 0000\} \tag{3.4}$$

then

$$\text{sub}_T(1110) = \{1100, 0110\}$$

For each element $\vec{y}$ in the training set T, we also define the subset $E(\vec{y})$ of points smaller than $\vec{y}$ that can be reached from $\vec{y}$ by an even number of paths forming unbroken loops. For example, in figure 1(a)

$$E(1110) = \{0100\} \qquad \text{and} \qquad E(1100) = \{0000\}$$

whereas in figure 1(b)

$$E(1111) = \{0000\}$$

For all the other patterns $\vec{z}$ in the figures, $E(\vec{z})$ is empty. Notice that in figure 1(a) the pattern 0000 does not belong to $E(1110)$ because the loop is broken by the line from 1100 to 0100.

The learning algorithm is now defined requiring the system, after a training set T, to represent the following function:

$$f(\vec{x}) = \sum_{\vec{y} \in T} x_1^{y_1} \cdots x_N^{y_N} \left(f(\vec{y}) + \sum_{\vec{\xi} \in \text{sub}_T(\vec{y})} f(\vec{\xi}) + \sum_{\vec{\eta} \in E(\vec{y})} f(\vec{\eta}) \right) \tag{3.5}$$

For example, for the training set in figure 1(a) the coefficient of $x_1x_2x_3$ is $\{f(1110)+f(1100)+f(0110)+f(0100)\}$, and in figure 1($b$) the coefficient of $x_1x_2x_3x_4$ is $\{f(1111)+f(1100)+f(0111)+f(0000)\}$.

For $N=4$, if the learning sequence is

$$1110 \quad 0100 \quad 1100 \quad 0110 \quad 0000 \quad 1000 \quad \ldots \tag{3.6}$$

the sequence of polynomials generated by the system would be:

1. $f(1110)x_1x_2x_3$
2. $f(0100)x_2+\{f(1110)+f(0100)\}x_1x_2x_3$
3. $f(0100)x_2+\{f(1100)+f(0100)\}x_1x_2+\{f(1110)+f(1100)\}x_1x_2x_3$
4. $f(0100)x_2+\{f(1100)+f(0100)\}x_1x_2+\{f(0110)+f(0100)\}x_2x_3$
   $+\{f(1110)+f(1100)+f(0110)+f(0100)\}x_1x_2x_3$
5. $f(0000)+\{f(0100)+f(0000)\}x_2+\{f(1100)+f(0100)\}x_1x_2$
   $+\{f(0110)+f(0100)\}x_2x_3$
   $+\{f(1110)+f(1100)+f(0110)+f(0100)\}x_1x_2x_3$
6. $f(0000)+\{f(1000)+f(0000)\}x_1+\{f(0100)+f(0000)\}x_2$
   $+\{f(1100)+f(0100)+f(1000)+f(0000)\}x_1x_2$
   $+\{f(0110)+f(0100)\}x_2x_3$
   $+\{f(1110)+f(1100)+f(0110)+f(0100)\}x_1x_2x_3$

The changes in the coefficients of some of the monomials, introduced in previous learning steps, result from the changes in the topology of the training set each time a new pattern is added.

The learning algorithm defined by equation (3.5) guarantees that, for a training set of K elements, the learned examples are correctly reproduced and the function represented by the system has at most K monomials of degree less than or equal to the maximum number of ones in the input patterns of the training set.

For hardware implementations, the following network-like approach to computing the coefficients of the polynomial is probably useful. For a training set T of K patterns we write the corresponding polynomial, computed at the pattern $\underset{\sim}{\alpha}(j)$, as follows:

$$P(\underset{\sim}{\alpha}(j))=f(\underset{\sim}{\alpha}(j))=\sum_{n=1}^{K}c_nI_n(\underset{\sim}{\alpha}(j)) \tag{3.7}$$

The monomial I_n that corresponds to the pattern $\underset{\sim}{\alpha}(n)$ is

$$I_n=x_1^{\alpha_1(n)}x_2^{\alpha_2(n)}\cdots x_N^{\alpha_N(n)}$$

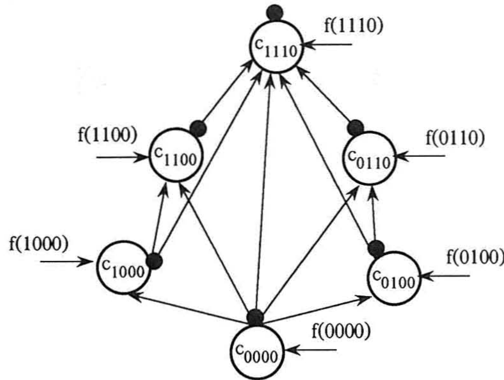


Figure 2: Equivalent network that computes the polynomial coefficients for the training set in figure 1(a).

Then $I_n(\underset{\sim}{\alpha}(j)) \neq 0$ if and only if $\underset{\sim}{\alpha}(n) \leq \underset{\sim}{\alpha}(j)$, and we may rewrite (3.7) as

$$\begin{aligned} c_j &= f(\underset{\sim}{\alpha}(j)) + \sum_{\underset{\sim}{\alpha}(n)<\underset{\sim}{\alpha}(j)} c_n \\ &= f(\underset{\sim}{\alpha}(j)) + \sum_n W_{jn} c_n \end{aligned} \tag{3.8}$$

with $W_{jn} = 1$ if $\underset{\sim}{\alpha}(n) < \underset{\sim}{\alpha}(j)$ and $W_{jn} = 0$ otherwise.

The coefficients of the polynomial are the solutions of the linear system (3.8), and may be obtained by the following iterative process:

$$c_j(t+1) = f(\underset{\sim}{\alpha}(j)) + \sum_n W_{jn} c_n(t) \tag{3.9}$$

Under iteration the coefficients become fixed, starting from the lowest elements and moving up along the chains. After a finite number of steps, equal to the size of the largest chain, the dynamical system (3.9) converges to the solution of (3.8). Therefore the calculation of the coefficients of the polynomial is equivalent to the evolution of a network with synaptic strengths one or zero, as defined by the order relation. Figure 2 shows the network corresponding to the 6-element training set in (3.6). The stationary output values of the nodes after the time evolution are the coefficients of the polynomial.

4. Smoothness and weighed majority rule

In finite fields (and in contrast to what one is familiar with for real-valued functions), fitting a set of data points to a polynomial of low degree does not guarantee a smooth interpolation of the data. For example, in F_5 the simplest polynomial that fits the points $f(0) = 0$ and $f(2) = 1$ is $f(x) = 3x$.

Computing $f(1)$ one now obtains 3, instead of 0 or 1 as smooth extrapolation would suggest.

To carry to finite fields a notion analogous to smooth extrapolation in the reals, one wants the value of the function at a point to be close to the value at neighboring points. To define what a neighbor is requires a notion of distance. In coding theory the notion of Hamming distance is used, this being the number of positions where two patterns differ from each other. This is the natural notion for coding because what one is concerned with is the number of errors in a message. Here, however, where the value of the variables are associated with the intensity values of some physical quantity, it seems more reasonable to use the vector distance

$$d(\vec{x},\vec{y}) = \left\{ \sum_i (x_i - y_i)^2 \right\}^{1/2} \tag{4.1}$$

For a pattern $\vec{x} \in F_p^N$ and a subset $T \subset F_p^N$, we define the set of neighbors of $\vec{x}$ in T, $\mathrm{nei}_T(\vec{x})$, as the set of T-patterns that lie on the boundary of the largest empty hypercube around $\vec{x}$.

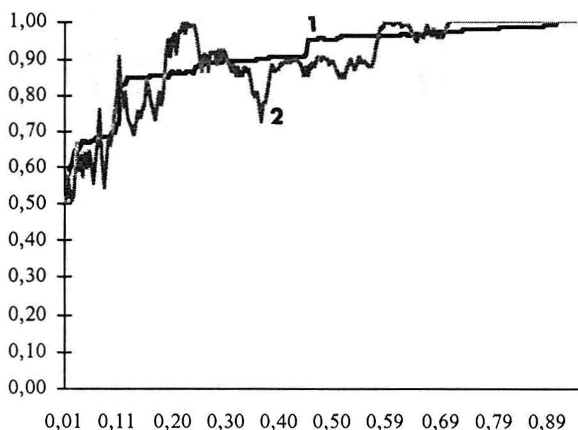
The training algorithm is now defined by requiring that after the training set T has been presented to the learning system, this one must represent the function

$$\begin{aligned} f(\vec{x}) = {} & \sum_{\vec{y}\in T} f(\vec{y}) \left(1 - (y_1 - x_1)^{p-1}\right) \cdots \left(1 - (y_N - x_N)^{p-1}\right) \\ & + \sum_{\vec{y}\notin T} \mathrm{Int} \left\{ \frac{1}{2} + \left(\sum_{\vec{\xi}\in \mathrm{nei}_T(\vec{y})} \frac{1}{d(\vec{y},\vec{\xi})} \right)^{-1} \sum_{\vec{\xi}\in \mathrm{nei}_T(\vec{y})} \frac{f(\vec{\xi})}{d(\vec{y},\vec{\xi})} \right\} \\ & \times \left(1 - (y_1 - x_1)^{p-1}\right) \cdots \left(1 - (y_N - x_N)^{p-1}\right) \end{aligned} \tag{4.2}$$

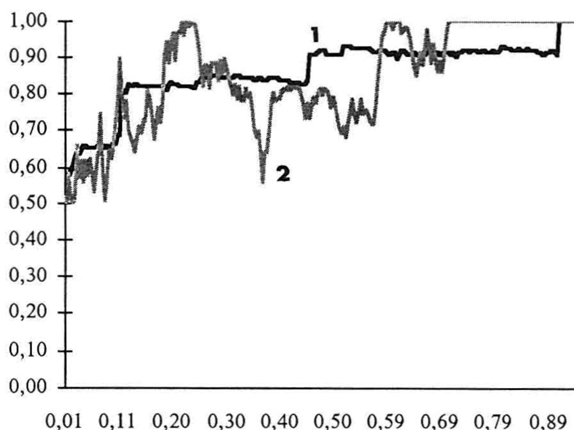
That is, the system reproduces the examples already learned, and for the new patterns it uses the values at neighboring points weighed by the distances.

5. Examples

The algorithms for polynomial learning (section 3) and weighed majority rule (section 4) were tested in two problems with $N = 8$ and $p = 2$. The first is the "even-odd" problem, in which the function to be learned is zero when the binary number represented by the input pattern is even, and one when the binary number is odd. The second is the "two-or-more clusters" or "contiguity" function. This function is one when there are two or more clusters of ones, and zero otherwise. The results are shown in figures 3 and 4. In figures 3(a) and 4(a) we plot the fraction of accurate outputs as a function of the relative size of the training set.



(*a*)



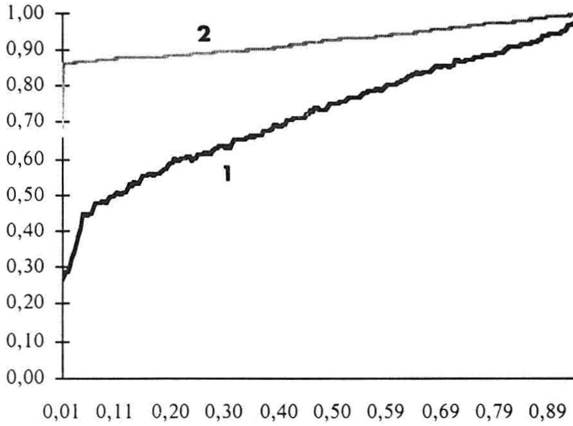
(*b*)

Figure 3: Even-odd problem (1. Polynomial learning, 2. Weighed majority rule). (*a*) Fraction of accurate outputs versus relative size of the training set. (*b*) Efficiency factor.

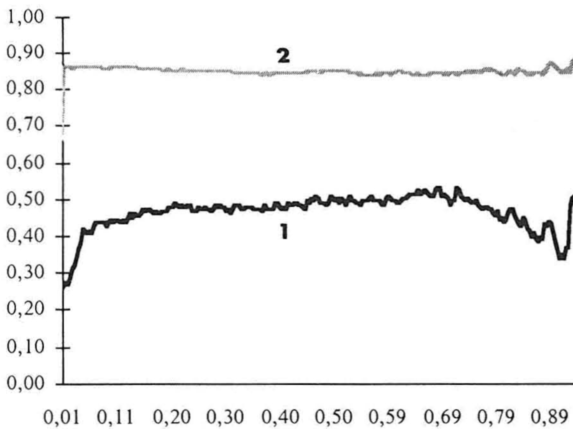
Let K be the size of the training set T, $n_t = 2^N$ be the total number of distinct input patterns, and n_e be the number of generalization errors at a given step of the learning process. We define the efficiency factor

$$\eta = 1 - \frac{n_e}{n_t - K}$$

The efficiency factor is related to the global generalization rate, defined in section 2, by $\eta = t(T, f)+(1/p)$. One expects that, by pure chance, half of the



(*a*)



(*b*)

Figure 4: Two or more clusters problem (1. Polynomial learning, 2. Weighed majority rule). (*a*) Fraction of accurate outputs versus relative size of the training set. (*b*) Efficiency factor.

outputs will be the correct ones (for $p = 2$), hence there is generalization only if $\eta > 0.5$. The efficiency factor is plotted in figures 3(*b*) and 4(*b*).

The algorithms were tested in uncorrelated trial runs, the patterns to be included in the evolving training set being chosen at random. The plots in figures 3 and 4 show the average over a typical set of three trials.

For the even-odd problem, both algorithms show generalization capabilities and their average performance is similar, although the "weighed majority rule" has larger fluctuations than "polynomial learning," which therefore

seems more appropriate for this problem. For the two-or-more clusters problem, "polynomial learning" has no generalization capacity, the number of accurate outputs (beyond those of the training set) being consistent with pure chance ($\eta \sim 0.5$). The "weighed majority rule" algorithm, however, is extremely efficient in this problem.

These results illustrate what we have stated before, that there is no universal generalization strategy. A general learning module would thus be better equipped with several complementary strategies and with the ability to switch between the strategies to optimize its performance. A difficulty arises, however, from the fact that learning algorithms and hardware architectures are related, in the sense that some algorithms are more suited than others for a given architecture.

References

[1] T. M. Mitchell, "Generalization as Search," *Artificial Intelligence*, **18** (1982) 203–226.

[2] J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, L. Jackel, and J. Hopfield, "Large Automatic Learning, Rule Extraction, and Generalization," *Complex Systems*, **1** (1987) 877–922.

[3] E. B. Baum and D. Haussler, "What Size Net Gives Valid Generalization?" *Neural Computation*, **1** (1989) 151–160.

[4] D. H. Wolpert, "A Benchmark for How Well Neural Nets Generalize," *Biological Cybernetics*, **61** (1989) 303–313.

[5] D. H. Wolpert, "A Mathematical Theory of Generalization," *Complex Systems*, **4** (1990) 151–249.

[6] A. Lapedes and R. Farber, "Non-linear Signal Processing Using Neural Nets: Prediction and System Modelling," Los Alamos preprint LA-UR-87-2662 (1987).

[7] E. Mjolsness, D. H. Sharp, and B. K. Alpert, "Scaling, Machine Learning, and Genetic Neural Nets," Los Alamos preprint LA-UR-88-142 (1988).

[8] D. Psaltis and M. Neifeld, "The Emergence of Generalization in Networks with Constrained Representations," pages 371–381 in *Proceedings of the 1988 IEEE International Conference on Neural Networks*, volume 1, San Diego (1988).

[9] F. Vallet and J.-G. Cailton, "Recognition Rates of the Hebb Rule for Learning Boolean Functions," *Physical Review A*, **41** (1990) 3059–3065.

[10] S. I. Gallant, "A Connectionist Learning Algorithm with Provable Generalization and Scaling Bounds," *Neural Networks*, **3** (1990) 191–201.

[11] H. Sompolinsky, N. Tishby, and H. S. Seung, "Learning from Examples in Large Neural Networks," *Physical Review Letters*, **65** (1990) 1683–1686.

[12] C. Bishop, "Improving the Generalization Properties of Radial Basis Function Neural Networks," Harwell preprint AEA FUS 94 (1991).

[13] J. Sietsma and R. J. F. Dow, "Creating Artificial Neural Networks That Generalize," *Neural Networks*, **4** (1991) 67–79.

[14] T. Poggio, V. Torre, and C. Koch, "Computer Vision and Regularization Theory," *Nature*, **317** (1985) 314–319.

[15] R. Lidl and G. Pilz, *Applied Abstract Algebra* (Berlin, Springer-Verlag, 1984).

[16] D. Welsh, *Codes and Cryptography* (Oxford, Oxford University Press, 1988).