OHIO STATE

High Level Connectionist Models

Jordan B. Pollack, Principal Investigator
Laboratory for Artificial Intelligence Research
Ohio State University
Columbus Ohio, 43210

Department of the Navy
Office of Naval Research
Arlington, Virginia 22217

Grant N00014-89-J1200
Semiannual Report
Nov 1989 - April 1990

1.0 Technical Progress.

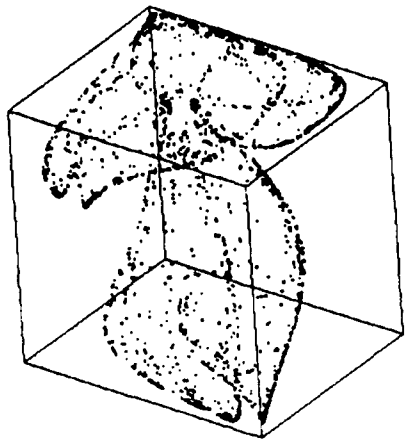


FIGURE 1 Image of a "strange" automaton, the result of a language acquisition device which was not satisfied with a finite number of states.

1.1 Sequential Cascaded Networks

Major work this semi-annum has been the further development and documentation of sequential cascaded networks. We resolved a difficulty with our initial work[1] applied to the acquisition of formal languages, which was the necessity of specifying intermediate states of a machine. Through a simple modification, our networks can now reliably learn languages from a finite set of positive and negative examples of strings. This has lead to two important discoveries. One is that by underconstraining the network for the task of inducing finite state machines, we ended up with a network which could induce infinite state machines whose states fall on an underlying "strange attractor" (Figure 1). This could be the basis for a language acquisition device without an inductive bias towards a fixed class of grammars. Secondly, we have discovered an elegant new form of induction, where the difference between finite and infinite perfor-

1. Pollack, J. B. (1987) Cascaded Back-Propagation on dynamic connectionist networks *Proceedings of the Ninth Conference of the Cognitive Science Society. Seattle.*

mance depends on the reshaping of an attractive landscape with arbitrarily small weight changes. This work is reported in the appendix, "The Induction of Dynamical Recognizers."

1.2 Schema Selection/Case Retrieval

We have been developing a new approach to case-based memory retrieval (CBMR) based upon RAAMs. A set of symbolic structures are first encoded into vectors in a high-dimensional space such that similar structures map to nearby vectors. A query is also converted to a vector in the same space, and similar structures are retrieved using Euclidean Nearest Neighbor search (a form of content-addressable memory) on the vectors. Initial studies show that the proposal is sound, but that similarity across the entire vector space is too imprecise a form of retrieval. We developed a greedy algorithm for selecting subsets of dimensions to craft specific indexes to the memory, and these results show promise for a new method of CBMR. A working paper is contained in the appendix.

1.3 Sensitivity Studies

We have completed the study of back-propagation's sensitivity to initial conditions, and include the final technical report in the appendix. Based on peer review feedback, the work looks as simpler functions, lower learning parameters, and views sensitivity to the learning parameters as well as the initial weights. This work was very important for two reasons. First, because we discovered this chaotic behavior in learning rather than in performance dynamics, and because this method has been widely used for 4 years and thought to be well-behaved.

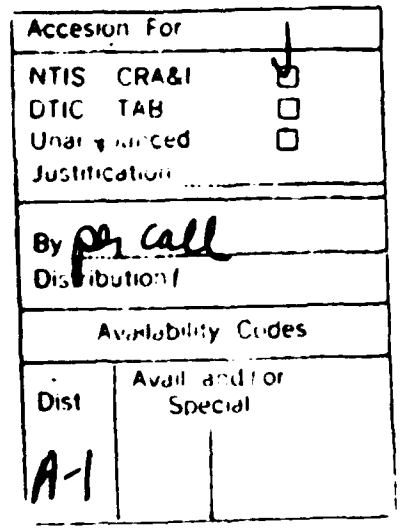
1.4 Floating Symbol Systems

We have started research and implementation on building systems where modular networks agree on shared representations. This is important for a major problem in AI which is the integration of symbolic processing with perceptual input. For example, a module which converts from pixel

arrays to letters must agree on the representation of a letter with the module that puts the letters into words. The first step in this process is to relax the assumption that the input (terminal) symbols to a module are fixed. When we tried a published method[2] of "floating" input symbols, all symbols became identical. Further study showed that this symbol fusion problem afflicts all current connectionist representation research, generating various work-arounds. Our solution, which involves feedback to push symbols apart, is described in a working paper in the appendix.

1.5 Allegro Common Lisp on the Cray Y-MP

The Ohio Super Computer Center (OSC) recently received an evaluation copy of Cray Common Lisp and Allegro Matrix for the Cray Y-MP. We have been granted an account and some computer time to evaluate and benchmark this software. While preliminary Gabriel benchmarks show that the Cray Lisp is only slightly faster than a Sun4, we believe that with access to efficient vector code, Lisp on the state-sponsored supercomputer may an appropriate engine for future applications of our research.



2. Miikkulainen & Dyer (1988). Forming global representations with extended back-propagation. *Proceedings of the IEEE second annual conference on Neural Networks*, San Diego.

2.0 Publication Activity

2.1 Papers Appeared

1. J. B. Pollack. (1989) No Harm Intended: A review of the *Perceptrons* Expanded Edition. *Journal of Mathematical Psychology*, 33, 3, 358-365.
2. Pollack. J. B. (1989). Structured Symbolic Representations in Neural State Vectors. *Fifth Annual Aerospace Applications of Artificial Intelligence Conference*, Dayton, OH.

2.2 Papers In Press

1. J. B. Pollack. Recursive Distributed Representations, *Artificial Intelligence*
2. J. Kolen and A. Goel. Learning in Parallel Distributed Processing Networks: Computational Complexity and Information Content. *IEEE Transactions on Systems, Man, and Cybernetics.*
3. J. B. Pollack. Language Acquisition via Strange Automata. *Proceedings of the 12th Annual Conference of the Cognitive Science Society.*
4. J. F. Kolen & J. B. Pollack. Back Propagation is Sensitive to Initial Conditions. *Proceedings of the 12th Annual Conference of the Cognitive Science Society.*
5. J. B. Pollack & J. A. Barnden. Conclusion. In Barnden, J. A. & Pollack, J. B. (Eds.) *Advances in Connectionist and Neural Computational Theory.* Norwood, NJ: Ablex.
6. J. A. Barnden & J. B. Pollack. Problems for High-Level Connectionism. In Barnden, J. A. & Pollack, J. B. (Eds.) *Advances in Connectionist and Neural Computational Theory.* Norwood, NJ: Ablex.

2.3 Papers Submitted

7. J. B. Pollack. The Induction of Dynamical Recognizers. *Machine Learning.*
8. J. F.Kolen & J. B. Pollack. Back Propagation is Sensitive to Initial Conditions. *Complex Systems.*
9. J. F.Kolen & J. B. Pollack. Back Propagation is Sensitive to Initial Conditions. *IEEE Conference on Neural Information Processing Systems.*
10. J. B. Pollack. Language Acquisition via Strange Automata. J. B. Pollack. *IEEE Conference on Neural Information Processing Systems.*
11. P. J. Angeline & J. B. Pollack. Avoiding Fusion in Floating Symbol Systems.*IEEE Conference on Neural Information Processing Systems.*

3.0 Appendices

The Ohio State University
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

April 1990

The Induction of Dynamical Recognizers

Jordan B. Pollack
Laboratory for Artificial Intelligence Research
228 Bolz Hall, 2036 Neil Avenue
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

Note: Submitted to *Machine Learning*.

The Induction of Dynamical Recognizers

Jordan B. Pollack

Laboratory for AI Research &
Computer & Information Science Department
The Ohio State University
2036 Neil Avenue
Columbus, OH 43210
(614) 292-4890
pollack@cis.ohio-state.edu

ABSTRACT

Sequential Cascaded Networks are recurrent higher order connectionist networks which are used, like finite state automata, to recognize languages. Such networks may be viewed as discrete dynamical systems (Dynamical Recognizers) whose states are points inside a multi-dimensional hypercube, whose transitions are defined not by a list of rules, but by a parameterized non-linear function, and whose acceptance decision is defined by a threshold applied to one dimension. Learning proceeds by the adaptation of weight parameters under error-driven feedback from performance on a teacher-supplied set of exemplars. The weights give rise to a landscape where input tokens cause transitions between attractive points or regions, and induction in this framework corresponds to the clustering, splitting and joining of these regions. Usually, the resulting landscape settles into a finite set of attractive regions, and is isomorphic to a classical finite-state automaton. Occasionally, however, the landscape contains a "Strange Attractor," to which there is no direct analogy in finite automata theory.

1. Introduction & Background

Recently, J. Feldman (personal communication) posed the language acquisition problem, as a challenge to connectionist networks. The problem is long-standing, and related to his own early work in finite-state automata induction (Feldman, 1972). In its general form, it can be stated quite simply, although it has many specialized variants.

Given a language, specified by example, find a machine which can recognize (or generate) that language.

In this paper I expose a recurrent high-order back-propagation network to both positive and negative examples of boolean strings, and report that although the network does not find the minimal-description finite state automata for the languages, it does induction in a novel and interesting fashion, and searches through a hypothesis space which, theoretically, is not constrained to machines of finite state.

This interpretation is dependent on an analogy among automata, neural networks, and non-linear dynamical systems, which I will try to set out below. After the background, I will describe the architecture, a set of experiments, some analyses of the results, and conclusions.

1.1. The Language Acquisition Problem

The language acquisition problem stated above can be specialized by identifying the *complexity* of the language and thus the class of machine necessary to recognize it. For example, most work has attacked only regular languages and assumed a hypothesis space of finite-state automata.

The problem can also be specialized by constraining the *presentation* of language data to the learning system. For example, are the examples selected from a finite or infinite subset of the target language, and are they presented once or many times? Are negative examples included and identified as such, included as detractors, or excluded? It is a widely-held view that children learn their native language from positive examples only.

Finally, the problem can be specialized by the *paradigmatic goals* of various disciplines. On the one hand mathematical and computational theorists might be concerned with the basic questions and definitions of learning, or with optimal algorithms (Angluin, 1982; Gold, 1967; Rivest & Schapire, 1987). On another hand, linguists may be concerned with how the question of learnability discriminates among grammatical frameworks and specifies necessarily innate properties of mind. On the third hand, psychologists might be concerned, in detail, with how a computational model actually matches up to the empirical data on child language acquisition. Rather than attempting to survey these areas, I point to the excellent theoretical review by (Angluin & Smith, 1983) and the books by (Wexler & Culicover, 1980) and by (MacWhinney, 1987) covering the linguistic and psychological approaches.

1.2. The Generative Capacity Problem for Connectionism

Before one can start talking about language acquisition, it is important to keep in mind the question of generative capacity, the standard measure of the complexity of infinite language structures. A connectionist model which could only induce bounded-length or regular languages might be interesting, but would not contribute at all to resolving the debate on the appropriateness of the connectionist approach to human cognition (Fodor & Pylyshyn, 1988). As Minsky and Papert (1988, p. x) recently put it, "Before a system can learn to recognize X, it must have the potential ability to represent X."

This was also the crux of the intellectual issue that made generative theories of language structure so prevalent. Within the framework of *computable* languages, Chomsky showed that there was a hierarchy of formal languages, defined by their ability to generate natural language phenomena, which were associated with differently powered machines. A machine based solely on "reflexes" or "associations" just is not playing the language game.

The simplest languages are *regular* languages, which are associated with Finite State Automata (FSA). Certain linguistic phenomena, such as center-embedded structures, could not be described by such automata without massive duplication. The next class of languages are *Context-free* languages, which can be generated by Push-Down Automata. Other linguistic phenomena such as feature agreement or crossed serial dependencies, could not be handled by such automata without massive duplication. The next class of languages are called *Context Sensitive* languages, and can only be generated by Turing Machines.[1]

Most connectionist models use fixed-structure networks, in which the weights are either set programmatically or adjusted slowly by a learning technique. The resultant networks are essentially "hard-wired" special-purpose computers that perform some application, like a 10-city Traveling Salesman problem (Hopfield & Tank, 1985) or a text-to-speech processor (Sejnowski & Rosenberg, 1987).

---

[1] The terms context-free versus context-sensitive are technical terms which apply to phrase-structure rules, not to whether adjacent words affect each other. And to retort that every physical machine is finite state anyway (so who cares), misses the central idea that massive duplication fails the scientific goal of parsimony.

When a special-purpose fixed-resource parallel machine is naively applied to language processing, the results have uniformly been parsers which can only deal with bounded-length sentences (Cottrell, 1985; Fanty, 1985; Hanson & Kegl, 1987; McClelland & Kawamoto, 1986; Selman, 1985), or regular languages.

On the one hand, it is quite clear that human languages are *not* formal, and thus are only weakly related to these mathematical syntactic structures. This might lead connectionists to claim that recursive computational power is not of the "essence of human computation "[2] On the other hand, it is also quite clear that without understanding the complexity issues, connectionists can stumble again and again into the trap that naive neural network models cannot account for the *data* of linguistics, which unambiguously requires complex representations and processes (Pinker & Prince, 1988).

There are several potential routes out of this generative capacity limitation, besides creating hybrid models which use normal programs to correct the limitations of neurally-inspired models (Berg, 1987; Lehnert, In Press; Waltz & Pollack, 1985). One general path involves discovering powerful distributed representations (Hinton, 1984; Pollack, 1988; Pollack, To Appear; Smolensky, 1987; Touretzky, 1986), which would allow static networks to process complex and novel structures. Two other routes, which are jointly exploited by the work herein, involve the sequential or recurrent use of adaptive networks, and the rapid modification of network structure through high-order (multiplicative) connections.

1.3. Recurrent and Higher-Order Networks

Despite early connectionist rhetoric about massive parallelism (Feldman & Ballard, 1982; Pollack & Waltz, 1982) the generative capacity and representational adequacy problems (in various guises) has led many researchers to investigate the use of recurrent or sequential networks for variable-length problems. An architecture could built of units computing in parallel, but the inputs might arrive sequentially. Jordan's (1986) work was perhaps the earliest indication that back-propagation could be used successfully to train recurrent networks. Lapedes & Farber (1987) demonstrated how a recurrent network could be trained to predict various non-linear sequences. Elman (1988) devised a very

[2] (Rumelhart & McClelland, 1986), p. 119.

simple and elegant approach to language processing by prediction. The papers in this volume by Elman and by Servan-Schreiber et al. explore this architecture in great detail, and I will return to a comparison near the end of this paper.

Secondly, various researchers have seen the need for rapid modification of network structure (Feldman, 1982; McClelland, 1985). The solutions have been less than widely applied, because the resulting systems are either neurally implausible, resource intensive, or quite unstable and difficult to control (Maxwell et al., 1986).

To understand how higher-order networks lead, in effect, to rapidly changing network structure, a network's state may be viewed as a vector $V_i(t)$ which evolves over time as an some function of the weights in the system, W_{ij}, which are fixed after learning:

$$V_i(t+1) = f(W_{ij}V_j(t))$$

A higher-order, or multiplicative system uses a three-dimensional array of weights, and involves multiplying it by the state vector twice:

$$V_i(t+1) = f(W_{ijk}V_k(t)V_j(t))$$

Therefore this can be viewed as a system where there is a weight associated with each product of outputs (hence the name multiplicative connection), or a system whose configuration (as defined by the 2D weights) changes at every instant:

$$W_{ij}(t) = W_{ijk}V_k(t)$$

$$V_i(t+1) = f(W_{ij}(t)V_j(t))$$

Thus, in the simple case of a system using multiplicative connections, each of the virtual $O(n^2)$ weights in a system of n units is a very sensitive function of the current activities of all the units, leading to a system with $O(n^3)$ "parameters" instead of $O(n^2)$.[3]

1.4. Finite State Automata, Neural Networks, Dynamical Systems

Finally, as a bit of background, I would like the reader to be able simultaneously consider a neural network as an automaton and as a dynamical system. This is not entirely new, as the analogies of automata-to-neural network, automata-to-dynamical

[3] An even more unconstrained approach are general Sigma-Pi networks (Williams, 1986), in which a weight can exist between each unit and the products of every possible subset of units, or $O(n2^n)$.

system, and neural net-to-dynamical system have been made *at least* by (VonNeumann, 1958), (Wolfram, 1984), and (Ashby, 1960), respectively.

A Finite State Automaton is a quadruple (Q, Σ, λ, F), where Q is a set of states (q_0 denotes the initial state), Σ is a finite input alphabet, λ is a transition function from $Q \times \Sigma \Rightarrow Q$ and F is a set of final (accepting) states, a subset of Q.

For small machines, λ is specified as a table, which lists a new state for each state and input. As an example, a machine which accepts boolean strings of odd parity can be specified as $Q=\{q_0, q_1\}$, $\Sigma=\{0, 1\}$, $F=\{q_1\}$, and λ, the table:

	Input	
State	0	1
q0	q0	q1
q1	q1	q0

After specifying the set of symbols, learning such an automaton involves the construction of the table of state-transitions. And, although we usually think in terms of fully specified tables or graphs, transition functions for an automaton can also be specified as a mathematical function of the current state and the input. For example, to get a machine to recognize boolean strings of odd parity, one merely has to specify that the next state is the exclusive-or of the current state and the input.

Generalizing from a multilayer networks' ability to perform exclusive-or to the various constructive and existential proofs of the functional/interpolative power of such networks (Hornik et al., To Appear; Lapedes & Farber, 1988a; Lippman, 1987), it is pretty obvious that recurrent neural networks can work at least as finite state automata, where the transition table is folded up into some moderately complex boolean function of the previous state and current input.

From another point of view, a recurrent network with feedback from k units can also be considered a k-dimensional discrete-time dynamical system, with a precise initial condition, $z_k(0)$ and a state space in a bounded subspace of $\mathbf{R}^k$ (i.e., "in-a-box" (Anderson et al., 1977)). The input string, $y_j(t)$, is merely considered "noise" from the environment which may or may not affect the systems evolution, and the governing function, F, is parameterized by weights, W:

$$z_k(t+1) = F_W(z_k(t), y_j(t)) \qquad \text{i}$$

If we view one of the dimensions of this system, say x_0 as an "acceptance" dimension, we can define the language accepted by such a *Dynamical Recognizer* as all strings of input tokens evolved from the precise initial state for which the accepting dimension of the state is above a certain threshold.

The first question to ask is how can such a dynamical system be constructed, or taught, to accept a particular language? The weights in the network, individually, do not correspond directly to graph transitions or to phrase structure rules. The second question to ask is what sort of generative power can be achieved by such systems?

2. The Model

To begin to answer the question of learning, I now present and elaborate upon my earlier work on Cascaded Networks (Pollack, 1987), which were used in a recurrent fashion to learn parity, depth-limited parenthesis balancing, and to map between word sequences and proposition representations (Pollack, To Appear).

2.1. Cascaded Networks

A Cascaded Network is a well-controlled higher-order connectionist architecture to which the back-propagation technique of weight adjustment (Rumelhart et al., 1986) can be applied. Basically, it consists of two subnetworks: The *function network* is a standard feed-forward network, with or without hidden layers. However, the weights are dynamically computed by the linear *context network*, whose outputs are mapped in a 1:1 fashion to the weights of the function net. Thus the input pattern to the context network is used to "multiplex" the the function computed, which can result in simpler learning tasks. For example, the famous Exclusive-or function of two inputs can be decomposed into two simpler functions of one input which are selected by the other:

$$XOR(y) = \begin{cases} y & \text{if } x = 0 \\ \neg y & \text{if } x = 1 \end{cases}$$

Thus, a simple 1-1 feedforward network is used to compute either the *identity* or *inverter* function, and a simple linear combination is used to set these weights. This network is is shown in Figure 1, uses only 4 weights instead of the 7 required by the smallest multi-layer perceptron. The large diamonds indicate units which compute linear combinations; the small diamonds indicate that the value computed is used as a weight;

Figure 1 near here

Figure 1. *Cascaded network for the XOR problem. The function network acts as either an inverter or non-inverting buffer depending on the context bit.*

and the black circles are bias units, whose outputs are always 1. Thus when x is 0, the function network computes g(3y - 1.5), and when x is 1 it computes g(1.5-3y), where $g(z) = 1/1+e^{-z}$, the usual sigmoidal squashing function.

Back-propagation is quite straightforward on a cascaded network. After determining the error terms for the weights of the function network, these are used as the error terms for the output units of the context network.

2.2. Sequential Cascaded Networks

When the outputs of the function network are used as inputs to context network, a system can be built which learns to produce specific outputs for variable-length sequences of inputs. Because of the multiplicative connections, each input is, in effect, processed by a different function.

Figure 2 near here

Figure 2. *A sequential cascaded network. The outputs of the function network are used as the next inputs to the context network, yielding a system whose function varies over time.*

Figure 2 shows a block diagram of a simple sequential cascaded network. Given an initial context, $z_k(0)$, and a sequence of inputs, $y_j(t)$, $t = 1...n$, the network computes a sequence of states vectors, $z_k(t)$, $t = 1...n$ by dynamically changing the set of weights, $w_{kj}(t)$:

$$w_{kj}(t) = w_{kjk}\, z_k(t-1)$$

$$z_k(t) = g(w_{kj}(t)\, y_j(t))$$

Assuming the desired output for the sequence, d_k, is known in advance, the error propagation phase can be applied just to the final step, generating terms for the context network, w_{kjk}:

$$\frac{\partial E}{\partial z_k(n)} = (z_k(n) - d_k)\, z_k(n)\, (1 - z_k(n))$$

$$\frac{\partial E}{\partial w_{kj}(n)} = \frac{\partial E}{\partial z_k(n)}\, y_j(n)$$

$$\frac{\partial E}{\partial w_{kjk}} = \frac{\partial E}{\partial w_{kj}(n)}\, z_k(n-1)$$

As in a standard batch or "epoch" organized back-propagation scheme, these errors are collected over a set of input-string, desired-output pairs. The termination condition for sequential cascaded back-propagation is the same as for the normal formulations, namely, when for each test-case and for each output value, the difference between desired and actual output is less than some threshold.

This can be used successfully when a teacher can supply a consistent and generalizable desired output for a large-enough set of strings.

2.3. The Backspace Trick

Unfortunately, this severely overconstrains the model. In learning parity, this doesn't matter, as the state fully determines the output. In the case of a higher-dimensional system, where we know whether or not a string is acceptable, but we *don't know* what the internal recurrent state of the system should be, there is a problem in generating the information necessary to modify the weights.

Jordan (1986) showed how recurrent back-propagation networks could be trained with "don't care" conditions. If there is no specific preference for the value of an output unit for a particular training example, simply consider the error term for that unit to be 0. This will work, *as long as that same unit receives feedback from other examples*. Otherwise the weights to that unit will never change.

For larger state vectors, the only prespecified output is the accept bit (1 for positive exemplars and 0 for negative ones). All of the "don't cares" line up and there is no information for modifying most of the weights in the network.

The first reaction, fully unrolling a recurrent network by maintaining vector histories (Rumelhart et al., 1986) has not lead to spectacular results (Mozer, 1988), the reason being that very tall networks with equivalence constraints between interdependent layers are unstable. My solution to this dilemma involves unrolling the loop only once: After propagating errors back from the final configuration, n, of the accept bit d through the one accepting "plane" of the weight "cube":

$$\frac{\partial E}{\partial z_a(n)} = (z_a(n) - d)\, z_a(n)\, (1 - z_a(n))$$

$$\frac{\partial E}{\partial w_{aj}(n)} = \frac{\partial E}{\partial z_a(n)}\, y_j(n)$$

$$\frac{\partial E}{\partial w_{ajk}} = \frac{\partial E}{\partial w_{aj}(n)}\, z_k(n-1)$$

We compute the error on the rest of the weights (e.g. $i = \{1 \cdots k \mid i \neq a\}$ by recycling the error on the accept plane with the network "backspaced" to its penultimate state:

$$\frac{\partial E}{\partial z_i(n-1)} = \sum_j \frac{\partial E}{\partial w_{ajk}}\, \frac{\partial}{\partial w_{aj}(n)}$$

$$\frac{\partial E}{\partial w_{ij}(n-1)} = \frac{\partial E}{\partial z_i(n-1)}\, y_j(n-1)$$

$$\frac{\partial E}{\partial w_{ijk}} = \frac{\partial E}{\partial w_{ij}(n-1)}\, z_k(n-2)$$

3. Experiments

Connectionist learning algorithms are very sensitive to the statistical properties of the set of exemplars which make up the learning environment. This has lead some psychological researchers to include the learning environment in the experimental parameters to manipulate (Plunkett & Marchman, 1989). Otherwise, it may not be clear if the results of a connectionist learning architecture are due to itself or due to skill or luck with setting up a collection of testcases. Therefore, rather than making up a set of languages and exemplars myself, I chose to work with test cases from the literature.

Tomita (1982) performed beautiful experiments in inducing finite automata from positive and negative examples. He used a genetically inspired two-step hill-climbing procedure, which manipulated 9-state machines by randomly adding, deleting or moving

transitions, or inverting the acceptability of a state. The current machine was compared to the mutated machine, and changed only when an improvement was made in the result of a global evaluation function, which tested the machine's performance on the training set. The first hill climbing used an evaluation function which subtracted the number of negative examples accepted from the number of positive examples rejected. The second step used an evaluation function which maintain correctness of the examples while minimizing the number of states.

Tomita ran his system on 7 cases and their complements. Each case was defined by two small sets of boolean strings, accepted by and rejected by the regular languages listed below.

1	1*
2	(1 0)*
3	no odd zero strings after odd 1 strings
4	no pairs of zeros
5	pairwise, an even sum of 01's and 10's.
6	number of 1's - number of 0's = 3n
7	0*1*0*1*

For uniformity, I ran all 7 cases on a sequential cascaded network of a 1-input 4-output function network (with bias, 8 weights to set) and a 3-input 8-output context network with bias. The total of 32 weights is essentially arranged as a 4 by 2 by 4 array. Only three of the output dimensions were fed back to the context network, along with a set of biases, and the 4th output unit was used as the acceptance dimension. The standard back-propagation learning rate was set to 0.3 and the momentum to 0.7. All 32 weights were reset to random numbers between ±0.5 for each run. Termination was when all accepted strings returned output bits above 0.8 and rejected strings below 0.2.

Of Tomita's 7 cases, all but cases #2 and #6 converged without a problem in several hundred epochs. Case 2 would not converge, and kept treating negative case 110101010 as correct; I had to modify the training set (by added reject strings 110 and 11010) in order to overcome this problem. Case 6 took several restarts and thousands of cycles to converge, cause unknown. I changed initial conditions during the period of experimentation, and used an initial state of (.2 .2 .2) for cases 1, 3, and 4, and (.5 .5 .5) for the rest.

Figure 3 near here

Figure 3. *(a) the ideal FSA Tomita had in mind for 1*, (b) the set of positive and negative exemplars (c) the language induced by the 3D dynamic recognizer. See text for detail*

Figure 4 near here

Figure 4. *(a) the ideal FSA Tomita had in mind for (10)*, (b) the set of positive and negative exemplars (c) the language induced by the 3D dynamic recognizer.*

Figure 5 near here

Figure 5. *(a) the ideal FSA Tomita had in mind for "no odd zero strings after odd 1 strings", (b) the set of positive and negative exemplars (c) the language induced by the 3D dynamic recognizer.*

Figure 6 near here

Figure 6. *(a) the ideal FSA Tomita had in mind for "no pairs of zeros", (b) the set of positive and negative exemplars (c) the language induced by the 3D dynamic recognizer.*

Figure 7 near here

Figure 7. *(a) the ideal FSA Tomita had in mind for "an even sum of 01's and 10's", (b) the set of positive and negative exemplars (c) the language induced by the 3D dynamic recognizer.*

The 7 figures, 3 - 9, contain all the data from these experiments, in a uniform fashion. Each figure is composed of three recursively shaded rectangles, each of which maps a number between 0 (white) and 1 (black) to all boolean strings up to length 10.

Starting at the top of each rectangle, each row r contains subrectangles for all the strings of length r in ascending order, so the subrectangle for each string is sitting right

---

Figure 8 near here

Figure 8. *(a) the ideal FSA Tomita had in mind for "number of 1's - number of 0's = 3n", (b) the set of positive and negative exemplars (c) the language induced by the 3D dynamic recognizer.*

Figure 9 near here

Figure 9. *(a) the ideal FSA Tomita had in mind for 0*1*0*1*, (b) the set of positive and negative exemplars (c) the language induced by the 3D dynamic recognizer.*

below its prefix. The top left subrectangle shows a number for the string 0, and the right shows a number for the string 1. Below the subrectangle for 0 are the ones for 00 and 01, and so on.

The top rectangle describes the "ideal" regular language Tomita was "aiming" to induce; the middle describes the set of exemplars (black indicating strings to accept, white indicating strings to reject, and with gray indicating unspecified strings), and the bottom shows the language induced by the sequential cascaded network, with strings classified into accepted and rejected by thresholding the acceptance dimension of the final state at 0.5.

There are a couple of things to note, the first being that none of the ideal languages were induced by the network. Even for the first language 1*, a 0 followed by a long string of 1's would be accepted by the network. If the network is not inducing the smallest consistent FSA, what is it doing? The constraint that a common set of weight parameters is used for all state transitions means that the network is not merely constructing massively duplicated states with an arbitrary transition function (which would render the problem trivial), but that there must be some geometric relationship among them.

4. Analysis

In my attempts at understanding the resultant networks, the first approach was to analyze what finite-state automata they were isomorphic to. The procedure was very simple. I ran the network as a generator, subjecting it to all possible boolean strings as input,

and collecting first, the set of strings for which the acceptance dimension was past threshold, and second, the set of states (points in 3-space) visited by the machine.

4.1. Limit behavior

Figure 10 near here

Figure 10. *Three stages in the adaptation of a network learning parity. (a) the test cases are separated, but there is a limit point for 1* at about 0.6. (b) after further training, the even and odd sequences are slightly separated. (c) after a little more training, the oscillating cycle is pronounced.*

Collecting the strings indicated one potential problem with the approach. After training the system can "fuzz out" for longer inputs than the ones given in the test cases. This can be examined for any particular recognizer. We simply observe the limit behavior on the accepting dimension for very long strings. For parity, since the string 1* requires an oscillation of states, we can examine the acceptance dimension as a function of the length. Figure shows three stages in the adaptation of a network for parity. At first, despite success at separating a small training set, a single attractor exists in the limit, so that long strings are indistinguishable. After a little further training, the even and odd strings are separated, and after still further training, the separation is enough to set a threshold easily.

Figure 11 near here

Figure 11. *The same three stages viewed figuratively as the splitting of a limit point into two points, and the amplification of their distinction.*

What initially appeared as a bug turns out to indicate a very interesting form of induction. Under feedback pressure to adapt, a slight change in weights leads to a point attractor being "bifurcated" into two. The result, in terms of performance, is significant! Before the split the network only worked correctly on short finite strings; afterwards, it worked on infinite strings. Figure 11 is an artists conception of the stages of this type of

induction.

4.2. Visualizing the Machines

Based upon preliminary studies of the parity example, my initial hypothesis that a set of clusters would be found, located in such a way that jumps between them could be organized geometrically by the quasi-linear combinations of state and input. Thus, after collecting the state information, it seemed that this would cluster into dense regions which would correspond to states in a FSA. A program could certainly explore this space automatically, by taking an unexplored state and combining it with both 0 and 1 inputs. To remove floating-point fuzz, it could use a parameter ε and throw out new states which were within ε euclidean distance from any state already known. Unfortunately, some of the machines seemed to grow drastically in size as ε was lowered!

Figure 12 near here

Figure 12. *Images of the attractors for the seven Tomita testcases. The points visited by all boolean input strings up to length ten are plotted.*

One reason for this seems to be that many "ravine" shaped clusters rather than point clusters are developed. Because the states are "in a box" of low dimension, we can view these machines graphically to gain some understanding of how the state space is being arranged. Graphs of the states visited by all possible inputs up to length 10, for the 7 test cases are shown in figure 12. Each figure contains 2048 points, but often they overlap.

The lack of closure under ε can now be seen as a completely different sort of attractor, making my earlier mapping attempt reminiscent of Mandelbrot's (1977, p. 25) essay about measuring the coastline of Britain. The variability in these structures certainly deserve further study, especially with regards to what types of landscapes are possible with different sized networks and alternative activation functions. The images (a) and (d) are what were expected, clumps of points which closely map to states of equivalent FSA's, although both the clusters seem to have well-defined similar substructures. Images (b) and (e) have simple ravines, which bleed into each other at their ends, probably indicating that longer strings will fuzz out.

Images (c), (f), and (g) are complex and quite unexpected, and will be further discussed below.

5. Related Work

The architecture and learning paradigm I used is closely related to the work of Elman and Servan-Schieber (this volume). Both networks rely on extending Michael Jordan's networks in a direction which separates visible output states from hidden recurrent states, without making the unstable "back-propagation through time" assumption. Besides our choice of language data to model, the two main differences are that

(1) They use a "predictive" paradigm, where error feedback is provided at every time step in the computation, and I used a "classification" paradigm, feeding back only at the end of the given examples. Certainly, the predictive paradigm is more psychologically plausible as a model of positive only presentation (c.f., Culicover & Wexler, pp 63-65).

I have no commitment to negative information; all that is required is some desired output which *discriminates* among the input strings in a generalizable way. Positive versus negative evidence is merely the simplest way (with 1 bit) to provide this discrimination. At no loss of generality, the desired output could be a representation or the necessary control input for a complex task.

(2) They use a single layer (quasi-linear) recurrence between states, whereas I use a higher-order (quadratic) recurrence. It is certainly plausible that this quadratic nature allows more "radical" non-linearities to blossom.

Besides continued analysis, scaling the network up beyond binary symbol alphabets, immediate followup work involves comparing and contrasting our respective models with the other two possible models, a higher-order network trained on prediction, and a quasi-linear model trained on classification.

Figure 13 near here

Figure 13. *Study of evolution of dynamics for a network learning parenthesis balancing, at epoch's 300, 350, 375, 400, 425, and 475.*

As further analysis goes, I have begun to study the evolutionary behavior of the sequential cascaded networks, to see how such complex structures could arise within a finite number of training cycles. I ran the same 32-weight model used on the Tomita examples on exemplars of balanced parenthesis strings (from (Pollack, 1987)), and saved the weights every 25 epochs. Figure 13 shows in 6 snapshots that the induction-as-attractor bifurcation method may operate quite in parallel, and might be closely related to the universal path to chaos. Along these lines, (Crutchfield & Young, 1989) has analyzed the computation underlying period-doubling in chaotic dynamical systems and has found power equivalent to indexed context-free grammars.

6. Discussion and Conclusion

The state spaces of the dynamical recognizers for Tomita cases 3, 6, and 7, are interesting, because, theoretically, they may be *infinite* state machines, where the states are not arbitrary or random, but are constrained in a powerful way by some mathematical principle. I believe that it is closely related to related to Barnsley's work on iterated systems, where affine "shrinking" transformations direct an infinite stream of random points onto a underlying fractal attractor. In the recurrent network case, the "shrinking" is accomplished via the sigmoidal function, and the stream of random points are all boolean strings.

My usage of the term "attractor" is not precisely the same sort as used in the "energy landscape" metaphor of optimization problems. (Ackley et al., 1985; Hopfield & Tank, 1985). Furthermore, because my state-space figures arise from subjecting a dynamical system with a precise initial condition to a combinatorial explosion of noise, the attractors are not precisely the same "strange attractors" which arise in deterministically chaotic systems, which are sensitive to initial conditions (Lorenz, 1963).

Certainly, the link between work in complex dynamical systems and neural networks is well-established both on the neurobiological level (Skarda & Freeman, 1987) and on the mathematical level (Derrida & Meir, 1988; Huberman & Hogg, 1987; Kurten, 1987). It is time that this link be further developed, especially as it applies to the question of the adequacy of connectionist, and other "emergent" approaches to high-level cognitive faculties, such as language (Pollack, 1989). The big question is whether any of the information structures which can be generated by complex dynamical systems can be

at all correlated with the structures arising in natural language.

In conclusion, I have by no means proven that a recurrent dynamical system can act as an efficient recognizer and generator for non-regular languages, though it does seem obvious.[4] But since Dynamical Recognizers are not organized as a PDA's or Turing Machines, it is not clear where the associated languages might fit inside the Chomsky rlierarchy.

Nevertheless, we can consider the implications for language (and language acquisition) of a family of automata which smoothly evolve between finite and infinite state machines without massively duplicated transition tables: It will give rise to an induction method which will apply without a priori specification of the grammatical framework of a language in question.

Generative capacity is neither natively assumed nor directly manipulated, but is an emergent property of the (fractal) geometry of a bounded non-linear system which arises in response to a specific learning task and is only revealed through performance.

Acknowledgements

This work is funded by Office of Naval Research Grant N00014-89-J-1200. I thank the colleagues who have directly or indirectly discussed various aspects of this research, including B. Chandrasekaren, J. Crutchfield, P. Culicover, W. Ogden, D. Touretzky, and my own student colleagues. 3D graphics were adapted from a program written by Tony Plate while we were both at NMSU.

7. References

Ackley, D. H., Hinton, G. E. & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann Machines. *Cognitive Science, 9*, 147-169.

Anderson, J. A., Silverstein, J. W., Ritz, S. A. & Jones, R. S. (1977). Distinctive Features, Categorical Perception, and Probability Learning: Some Applications of a Neural Model. *Psychological Review, 84*, 413-451.

---

[4] Assuming rational numbers for states, a recurrent multiplicative relationship would be enough to start counting, which is necessary for beginning to handle context-free embeddings, of the sort $a^n b^n$; For example, if $z(t+1) = .5a(t)z(t) + 2b(t)z(t)$. Assuming irrationals in the recurrence relationship, as physicists inadvertently do, the languages might not even be computable!

Angluin, D. (1982). *Journal of the Association for Computing Machinery, 29*, 741-765.

Angluin, D. & Smith, C. H. (1983). Inductive Inference: Theory and Methods. *Computing Surveys, 15*, 237-269.

Ashby, W. R. (1960). *Design for a Brain: The origin of adaptive behaviour (Second Edition)*. New York: John Wiley & Sons.

Berg, G. (1987). A Parallel Natural Language Processing Architecture with Distributed Control. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*. Seattle, 487-495.

Cottrell, G. W. (1985). Connectionist Parsing. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*. Irvine, CA.

Crutchfield, J. P & Young, K. (1989). Computation at the Onset of Chaos. In W. Zurek, (Ed.), *Complexity, Entropy and the Physics of INformation*. Reading, MA: Addison-Wesley.

Derrida, B. & Meir, R. (1988). Chaotic behavior of a layered neural network. *Phys. Rev. A, 38*.

Elman, J. L. (1988). Finding Structure in Time. Report 8801, San Diego: Center for Research in Language, UCSD.

Fanty, M. (1985). Context-free parsing in Connectionist Networks. TR174, Rochester, N.Y.: University of Rochester, Computer Science Department.

Feldman, J. A. (1972). Some Decidability Results in grammatical Inference. *Information & Control, 20*, 244-462.

Feldman, J. A. (1982). Dynamic Connections in Neural Networks. *Biological Cybernetics, 46*, 27-39.

Feldman, J. A. & Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science, 6*, 205-254.

Fodor, J. & Pylyshyn, A. (1988). Connectionism and Cognitive Architecture: A Critical Analysis. *Cognition, 28*, 3-71.

Gold, E. M. (1967). Language Identification in the Limit. *Information & Control, 10*, 447-474.

Hanson, S. J. & Kegl, J. (1987). PARSNIP: A connectionist network that learns natural language grammar from exposure to natural language sentences. In *Proceedings of the Ninth Conference of the Cognitive Science Society*. Seattle, 106-119.

Hinton, G. E. (1984). Distributed Representations. CMU-CS-84-157, Pittsburgh, PA: Carnegie-Mellon University, Computer Science Department.

Hopfield, J. J. & Tank, D. W. (1985). 'Neural' computation of decisions in optimization problems. *Biological Cybernetics, 52*, 141-152.

Hornik, K., Stinchcombe, M. & White, H. (To Appear). Multi-layer Feedforward Networks are Universal Approximators. In *Neural Networks*. .

Huberman, B. A. & Hogg, T. (1987). Phase Transitions in Artificial Intelligence Systems. *Artificial Intelligence, 33*, 155-172.

Jordan, M. I. (1986). Serial Order: A Parallel Distributed Processing Approach. ICS report 8608, La Jolla: Institute for Cognitive Science, UCSD.

Kurten, K. E. (1987). Phase transitions in quasirandom neural networks. In *Institute of Electrical and Electronics Engineers First International Conference on Neural Networks*. San Diego, II-197-20.

Lapedes, A. S. & Farber, R. M. (1988). How Neural Nets Work. LAUR-88-418: Los Alamos.

Lapedes, A. S. & Farber, R. M. (1988). Nonlinear Signal Processing using Neural Networks: Prediction and system modeling. *Biological Cybernetics, To appear*.

Lehnert, W. G. (In Press). *Advances in Connectionist and Neural Computation Theory*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Lippman, R. P. (1987). An introduction to computing with neural networks. *Institute of Electrical and Electronics Engineers ASSP Magazine, April*, 4-22.

Lorenz, E. N. (1963). Deterministic Nonperiodic Flow. *Journal of Atmospheric Sciences, 20*, 130-141.

MacWhinney, B. (1987). In *Mechanisms of Language Acquisition*. Hillsdale: Lawrence Erlbaum Associates.

Mandelbrot, B. (1982). *The Fractal Geometry of Nature*. San Francisco: Freeman.

Maxwell, T., Giles, C. L., Lee, Y. C. & Chen, H. H. (1986). Nonlinear Dynamics of Artificial Neural Systems. In *Proceedings of a workshop on Neural Networks for Computing*. Snowbird, UT, 299-304.

McClelland, J. L. (1985). Putting Knowledge in its Place. *Cognitive Science, 9*, 113-146.

McClelland, J. & Kawamoto, A. (1986). Mechanisms of Sentence Processing: Assigning Roles to Constituents. In J. L. McClelland, D. E. Rumelhart & the PDP research Group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, Vol. 2. Cambridge: MIT Press.

Minsky, M. & Papert, S. (1988). *Perceptrons*. Cambridge, MA: MIT Press.

Mozer, M. (1988). A focused Back-propagation Algorithm for Temporal Pattern Recognition. CRG-Technical Report-88-3: University of Toronto.

Pinker, S. & Prince, A. (1988). On Language and Connectionism: Analysis of a parallel distributed processing model of language inquisition.. *Cognition, 28*, 73-193.

Plunkett, K. & Marchman, V. (1989). Pattern Association in a Back-propagation Network: Implications for Child Language Acquisition. Technical Report 8902, San Diego: UCSD Center for Research in Languge.

Pollack, J. B. & Waltz, D. L. (1982). Natural Language Processing Using Spreading Activation and Lateral Inhibition. In *Proceedings of the Fourth Annual Cognitive Science Conference*. Ann Arbor, MI, 50-53.

Pollack, J. B. (1987). Cascaded Back Propagation on Dynamic Connectionist Networks. In *Proceedings of the Ninth Conference of the Cognitive Science Society*. Seattle, 391-404.

Pollack, J. B. (1988). Recursive Auto-Associative Memory: Devising Compositional Distributed Representations. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*. Montreal, 33-39.

Pollack, J. B. (1989). Implications of Recursive Distributed Representations. In D. Touretzky, (Ed.), *Advances in Neural Information Processing Systems*. Los Gatos, CA: Morgan Kaufman.

Pollack, J. B. (To Appear). Recursive Distributed Representation. In *Artificial Intelligence*. .

Rivest, R. L. & Schapire, R. E. (1987). A new approach to unsupervised learning in deterministic environments. In *Proceedings of the Fourth International Workshop on Machine Learning*. Irvine, 364-475.

Rumelhart, D. E. & McClelland, J. L. (1986). PDP Models and General Issues in Cognitive Science. In D. E. Rumelhart, J. L. McClelland & the PDP research Group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, Vol. 1. Cambridge: MIT Press.

Rumelhart, D. E., Hinton, G. & Williams, R. (1986). Learning Internal Representations through Error Propagation. In D. E. Rumelhart, J. L. McClelland & the PDP research Group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, Vol. 1. Cambridge: MIT Press.

Sejnowski, T. J. & Rosenberg, C. R. (1987). Parallel Networks that Learn to Pronounce English Text. *Complex Systems, 1*, 145-168.

Selman, B. (1985). Rule-Based Processing in a Connectionist System for Natural Language Understanding. CSRI-168, Toronto, Canada: University of Toronto, Computer Systems Research Institute.

Skarda, C. A. & Freeman, W. J. (1987). How brains make chaos. *Brain & Behavioral Science, 10*.

Smolensky, P (1987). A method for connectionist varible binding. Technical Report UU-CS-356-87. Boulder, Colorado: Computer Science Dept, Univ. of Colorado.

Tomita, M. (1982). Dynamic construction of finite-state automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Cognitive Science Conference*. Ann Arbor, MI, 105-108.

Touretzky, D. S. (1986). BoltzCONS: Reconciling connectionism with the recursive nature of stacks and trees. In *Proceedings of the 8th Annual Conference of the Cognitive Science Society*. Amherst, MA. 522-530.

VonNeumann, J. (1958). *The Computer and the Brain*. New Haven: Yale University Press.

Waltz, D. L. & Pollack, J. B. (1985). Massively Parallel Parsing: A strongly interactive model of Natural Language Interpretation. *Cognitive Science, 9*, 51-74.

Wexler, K. & Culicover, P. W. (1980). *Formal Principles of Language Acquisition*. Cambridge: MIT Press.

Williams, R. (1986). The Logic of Activation Functions. In D. E. Rumelhart, J. L. McClelland & the PDP research Group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, Vol. 1. Cambridge: MIT Press.

Wolfram, S. (1984). Universality and Complexity in Cellular Automata. *Physica, 10D*, 1-35.

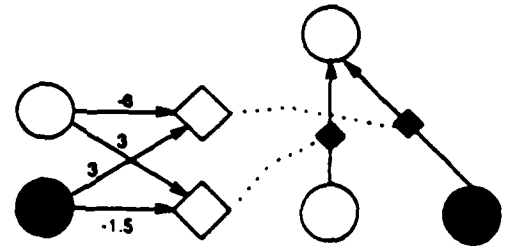


FIG 1

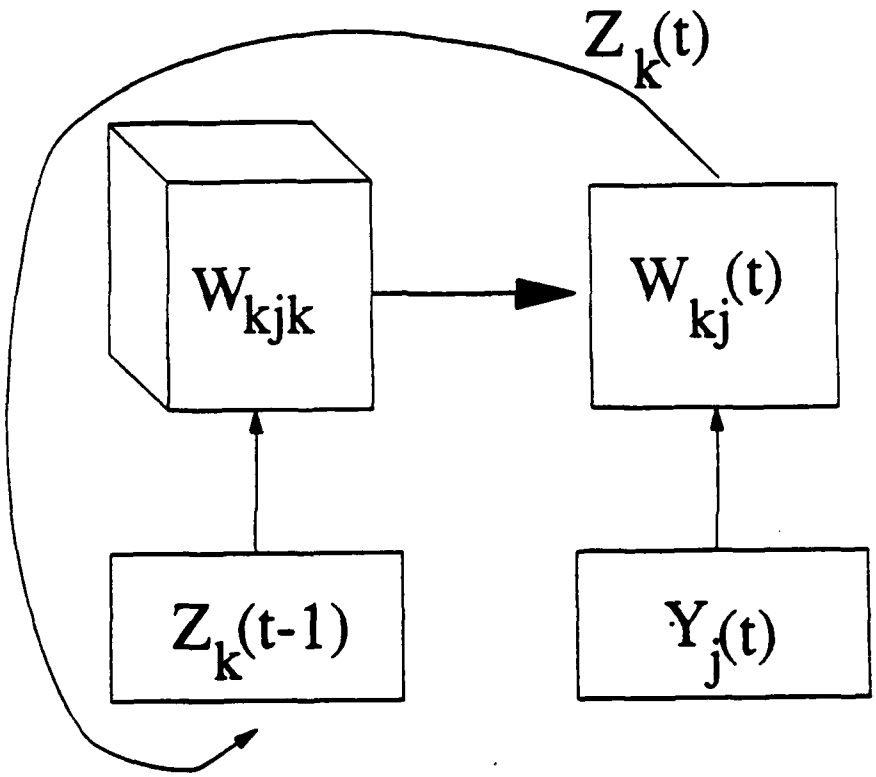
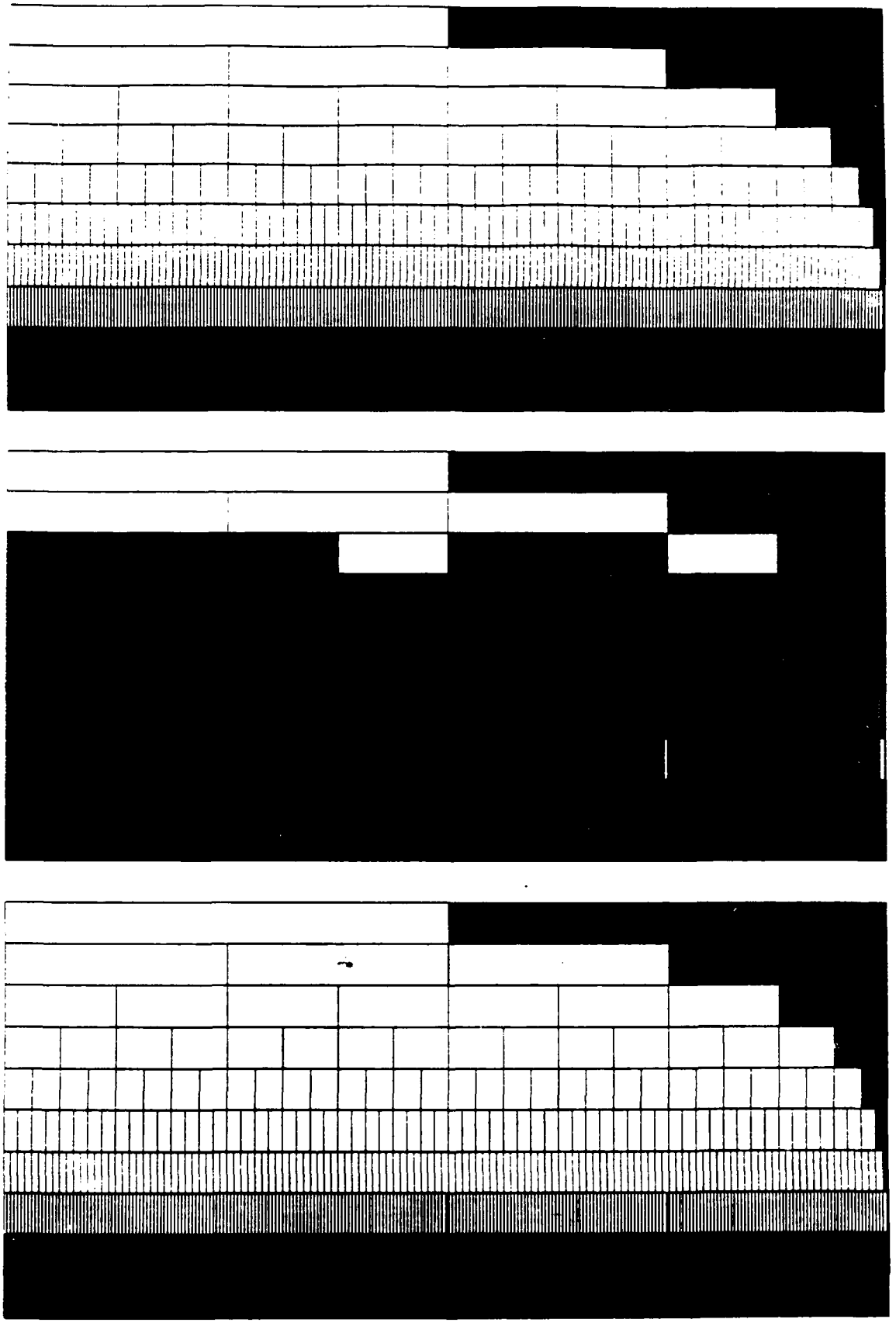
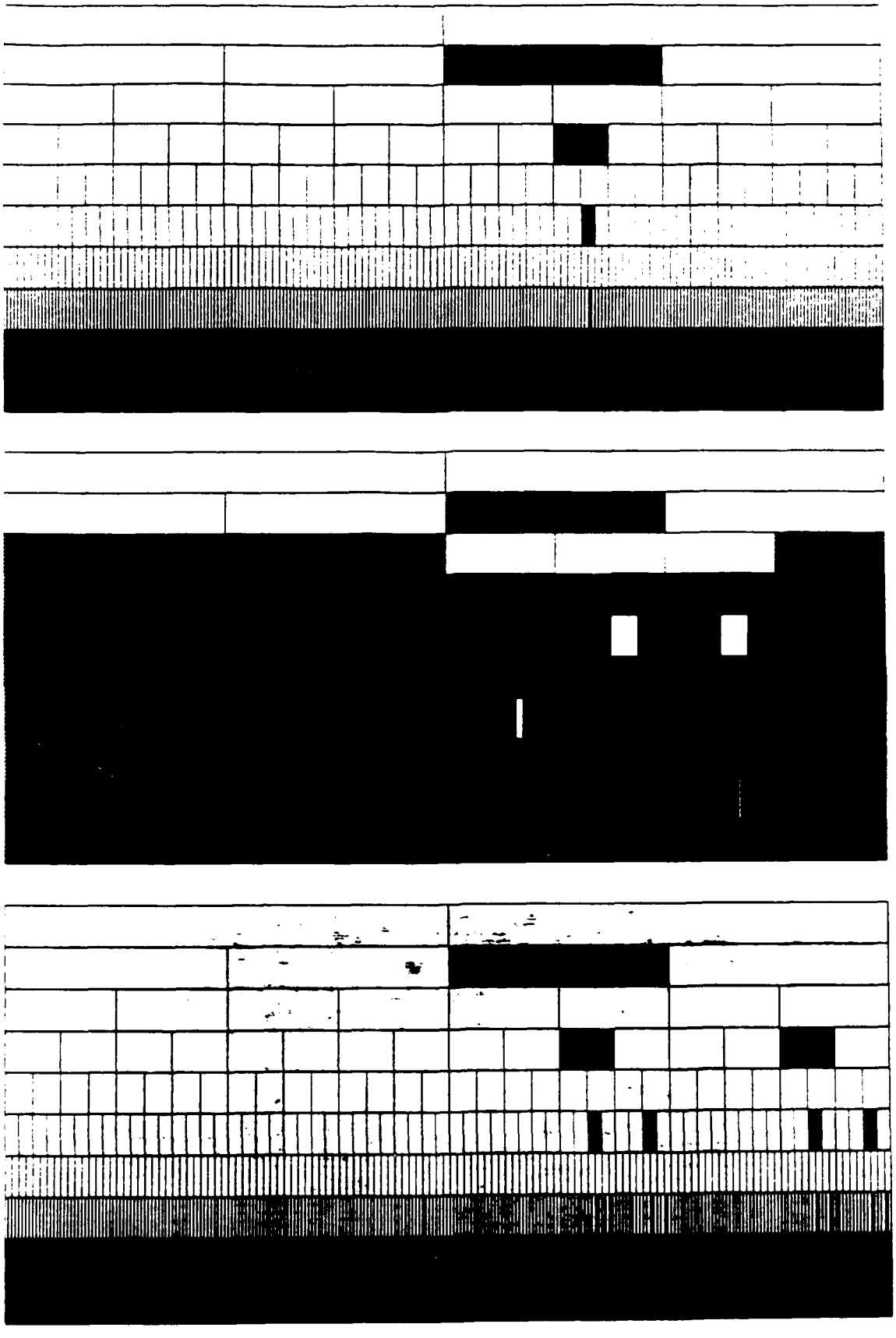


Fig 2

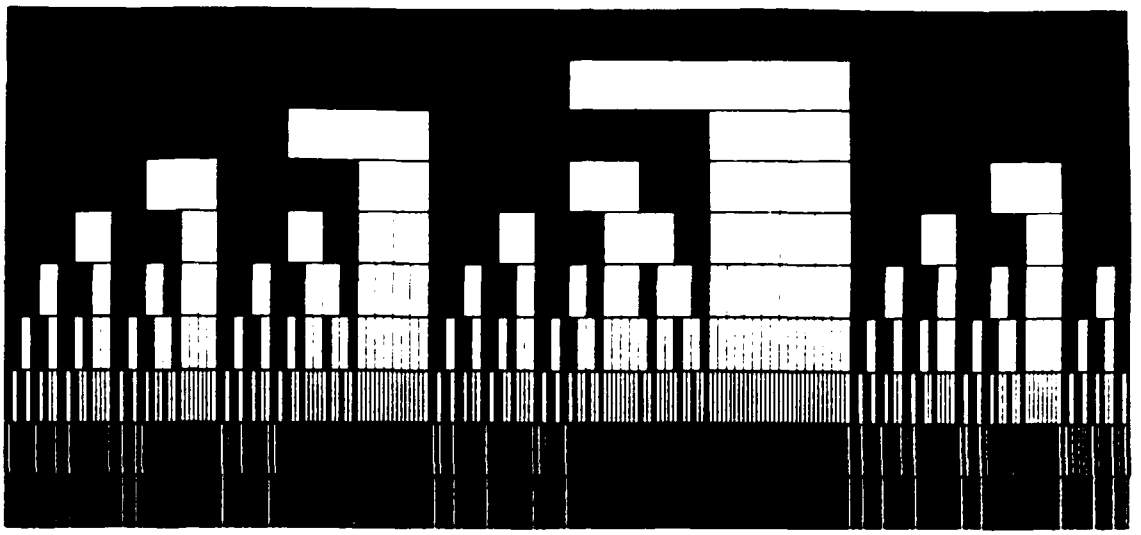


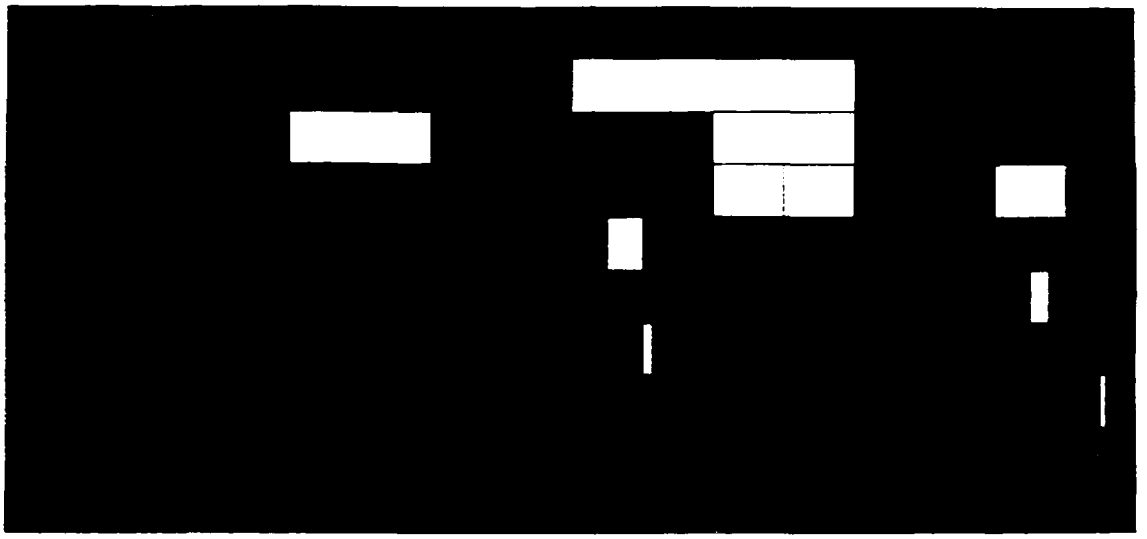
1*

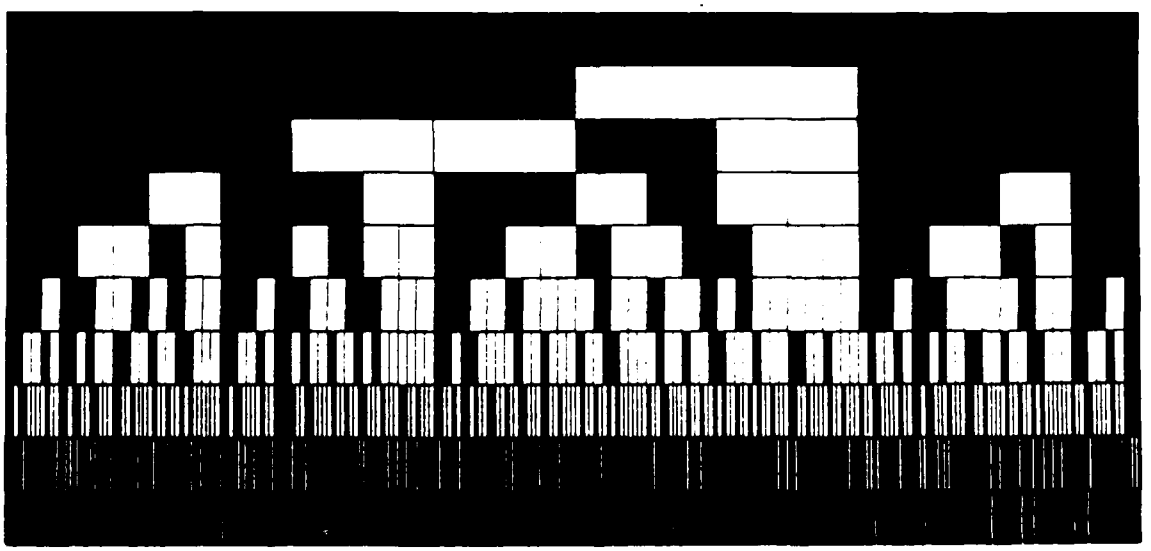
FIG 3



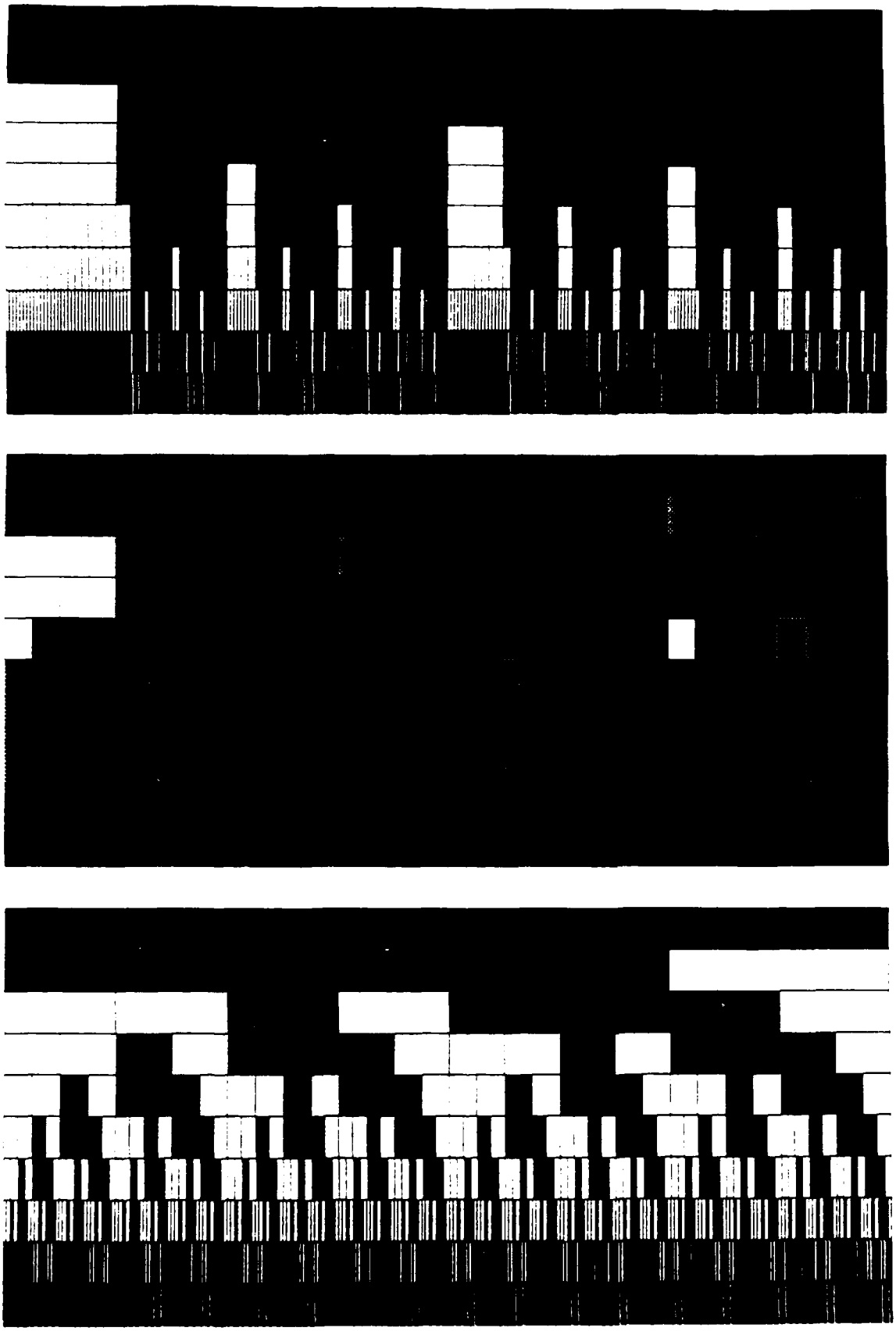
$(10)^*$



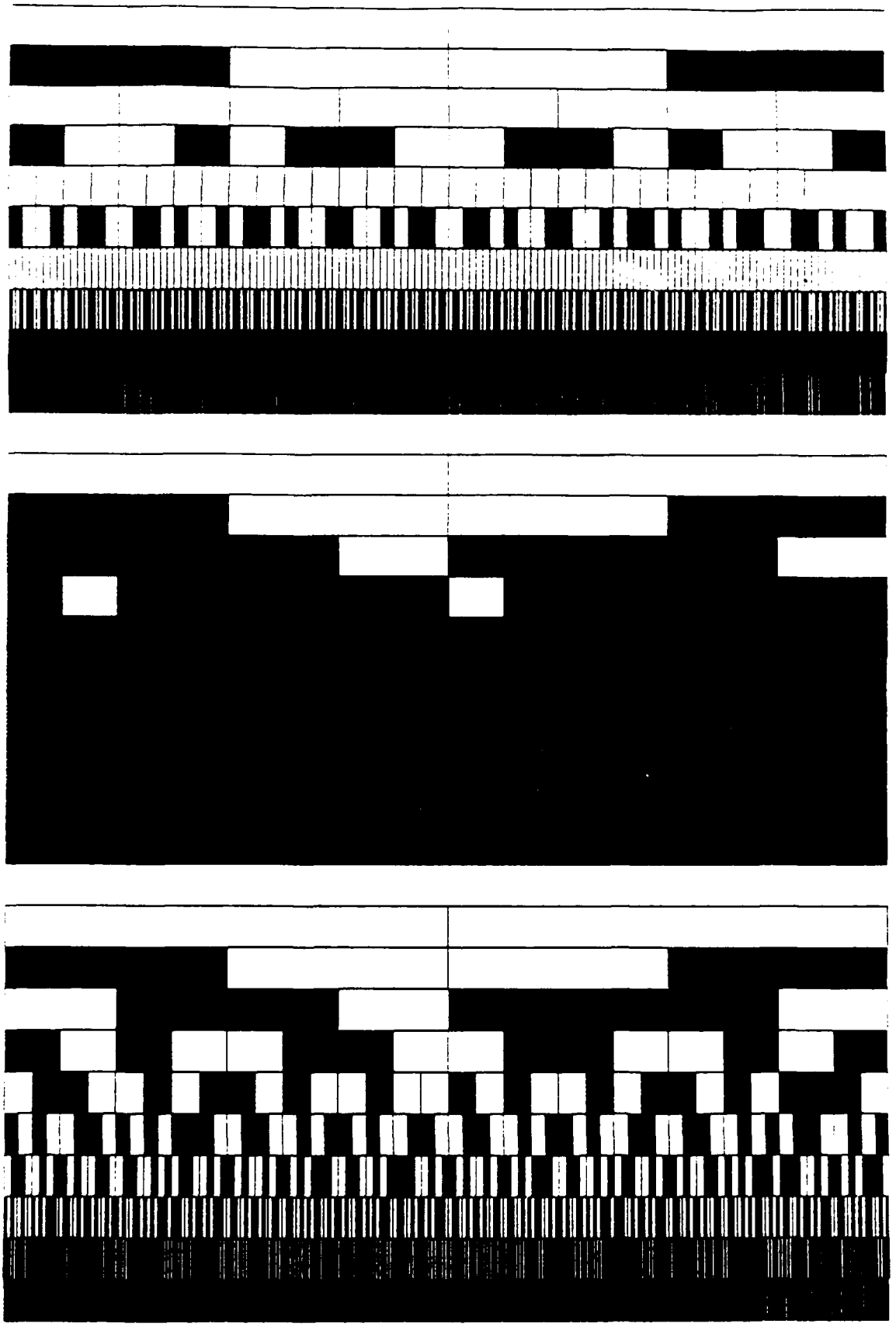




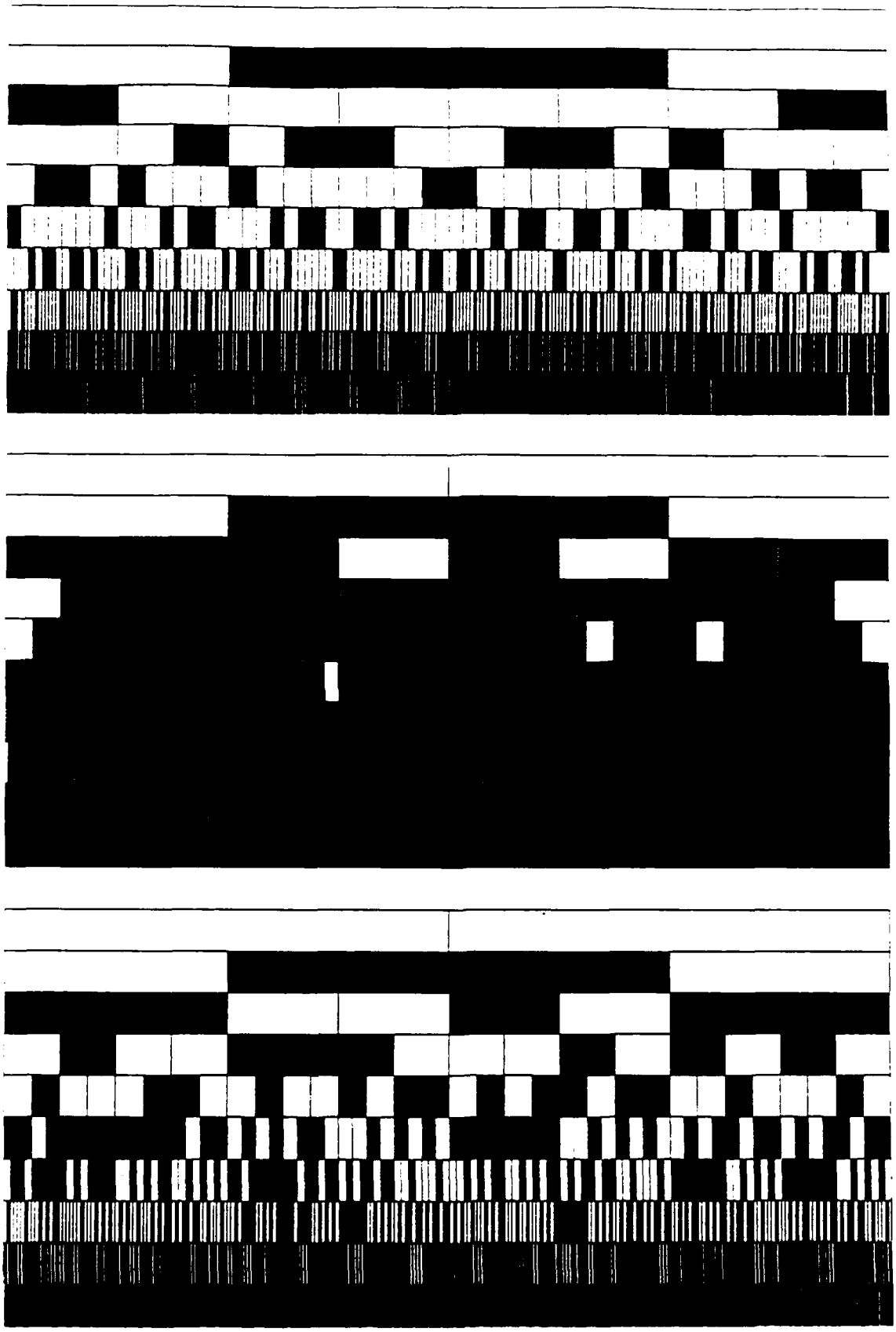
NO GOD ON ~~AFTER~~ GOD 1M



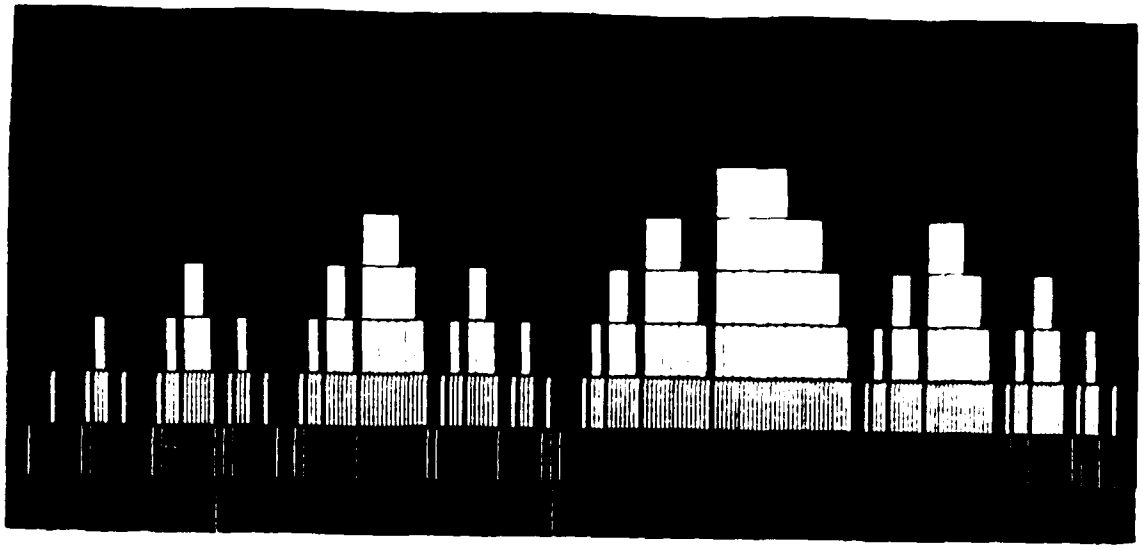
NO MORE THAN 00

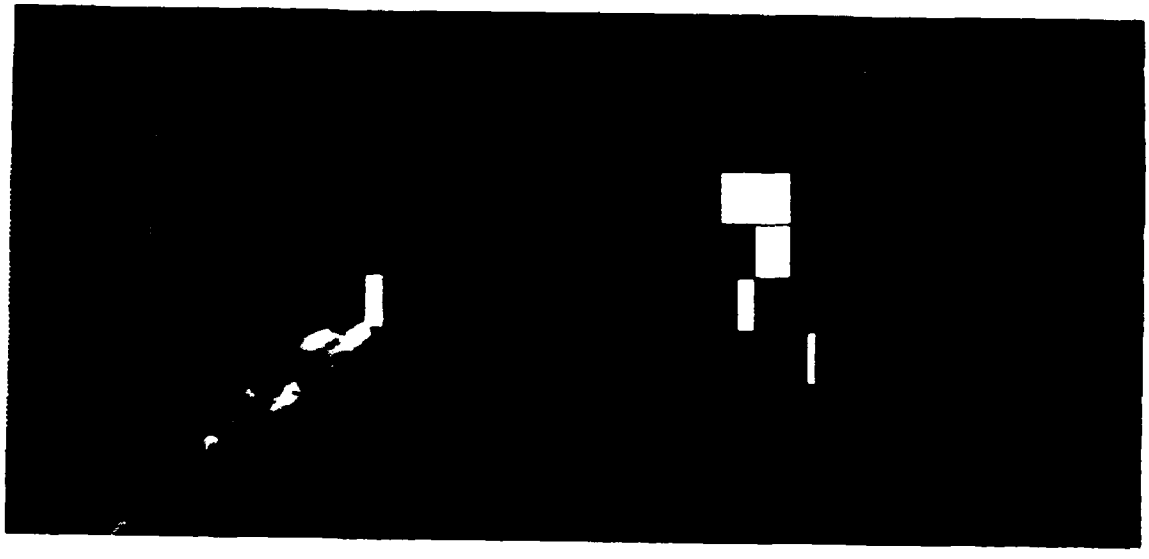


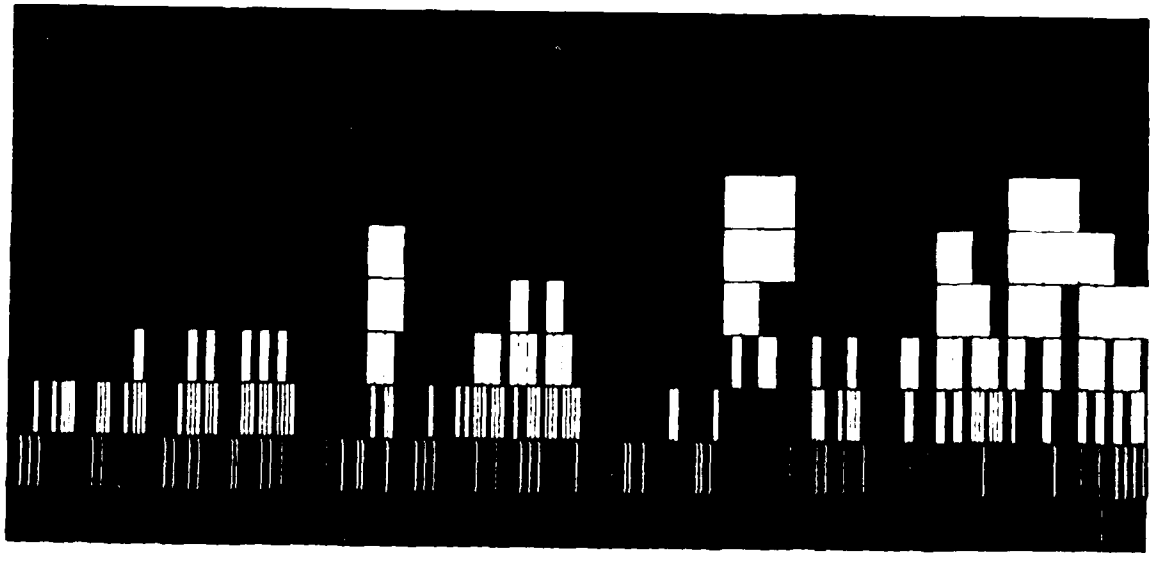
NO ODD SETS OF (01) OR (10)

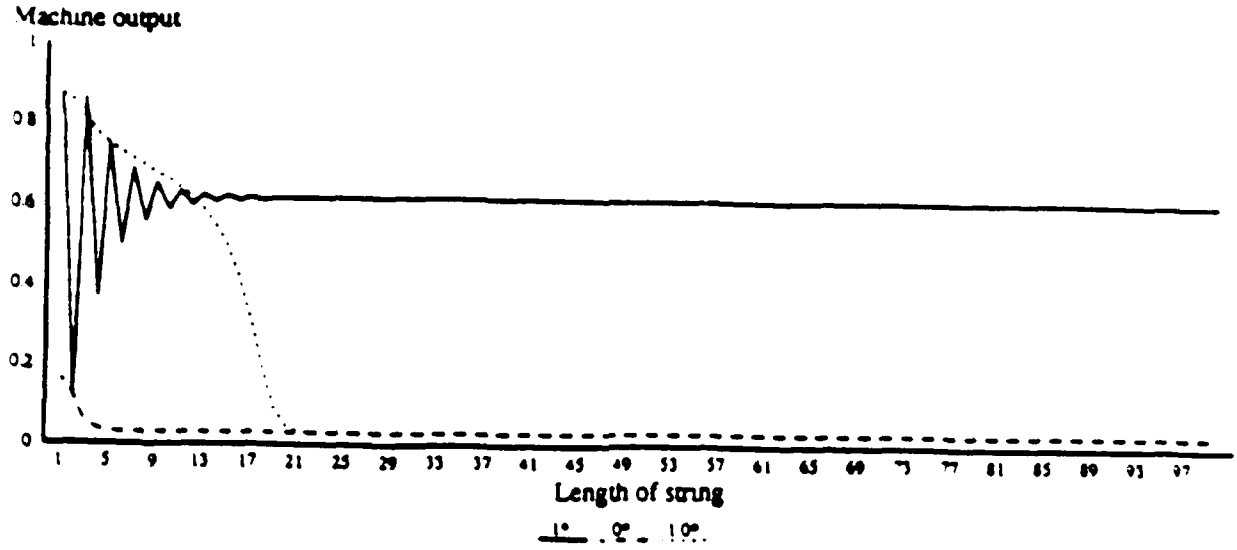


1's - # 0's = 3n

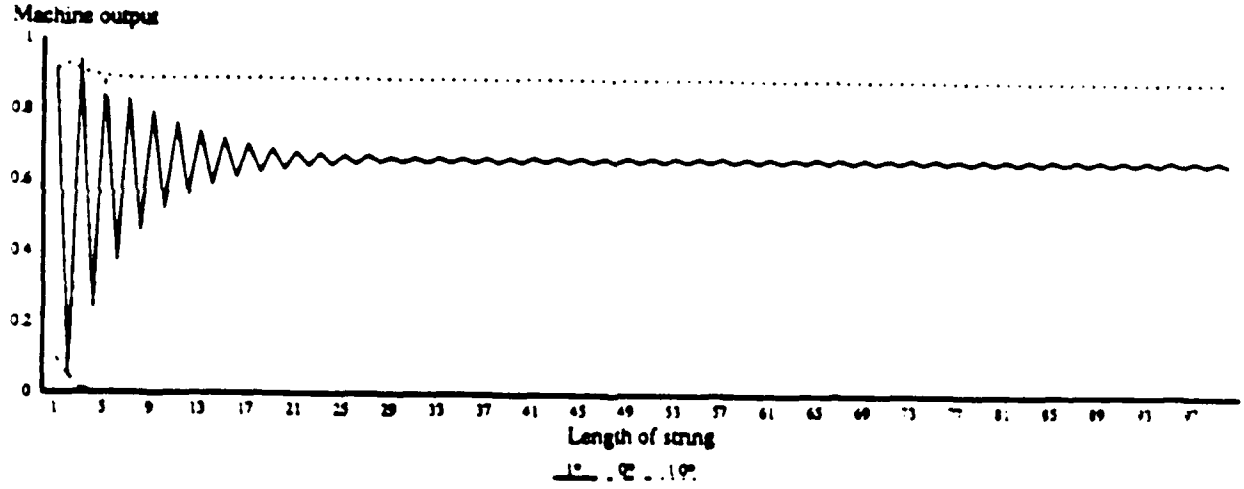




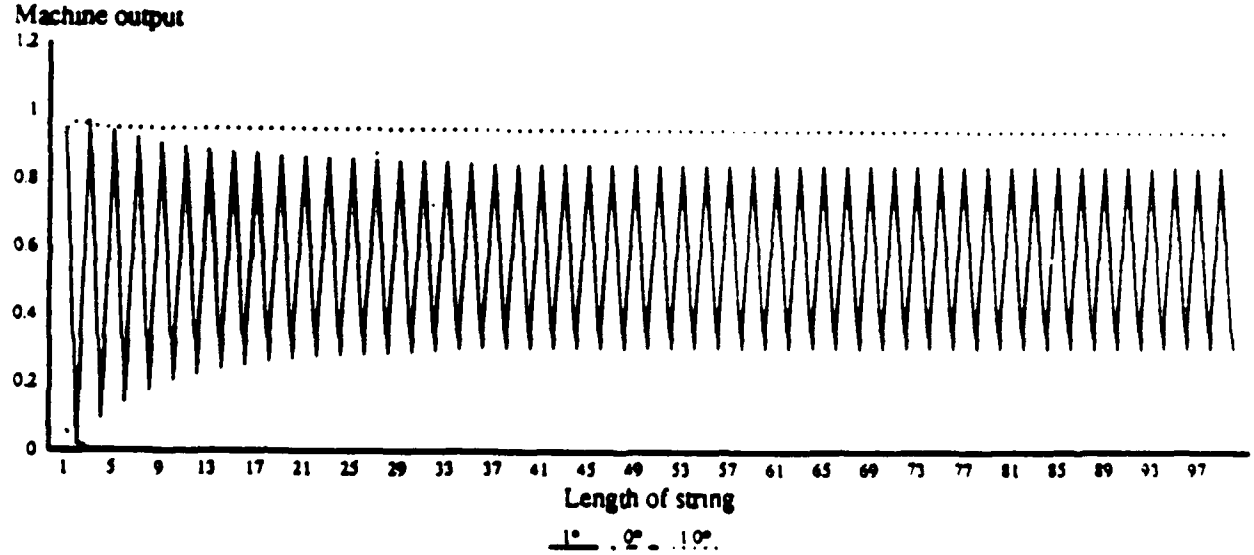




1° . 2° . 10°.



1° . 2° . 10°.



1° . 2° . 10°.

FIG 1D

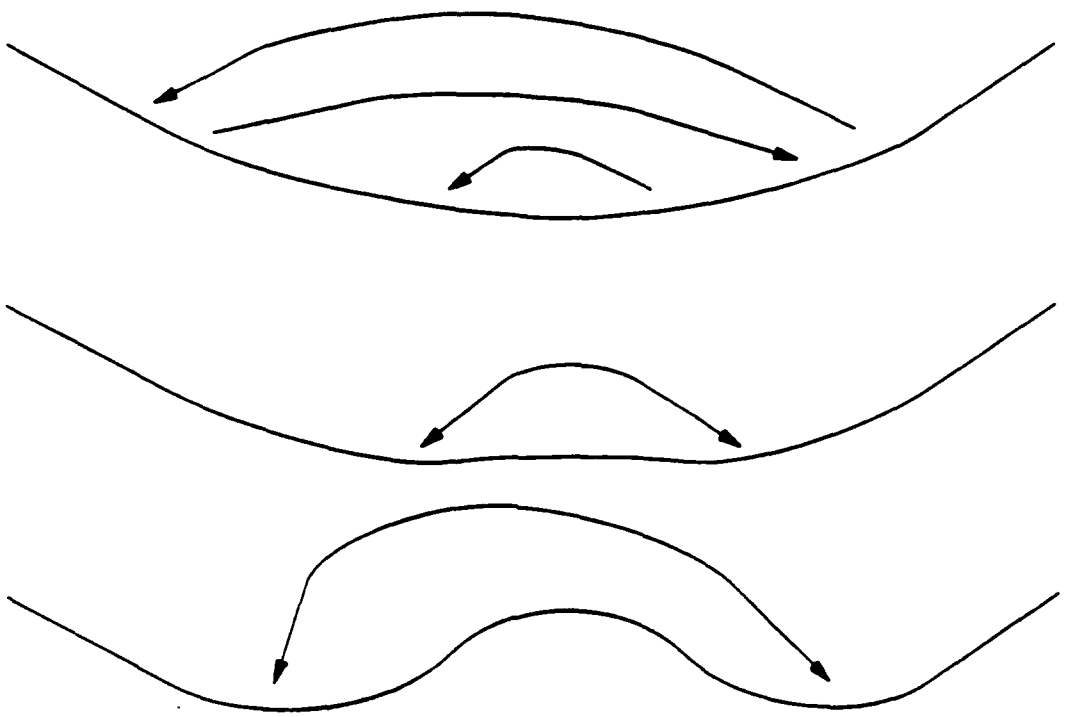
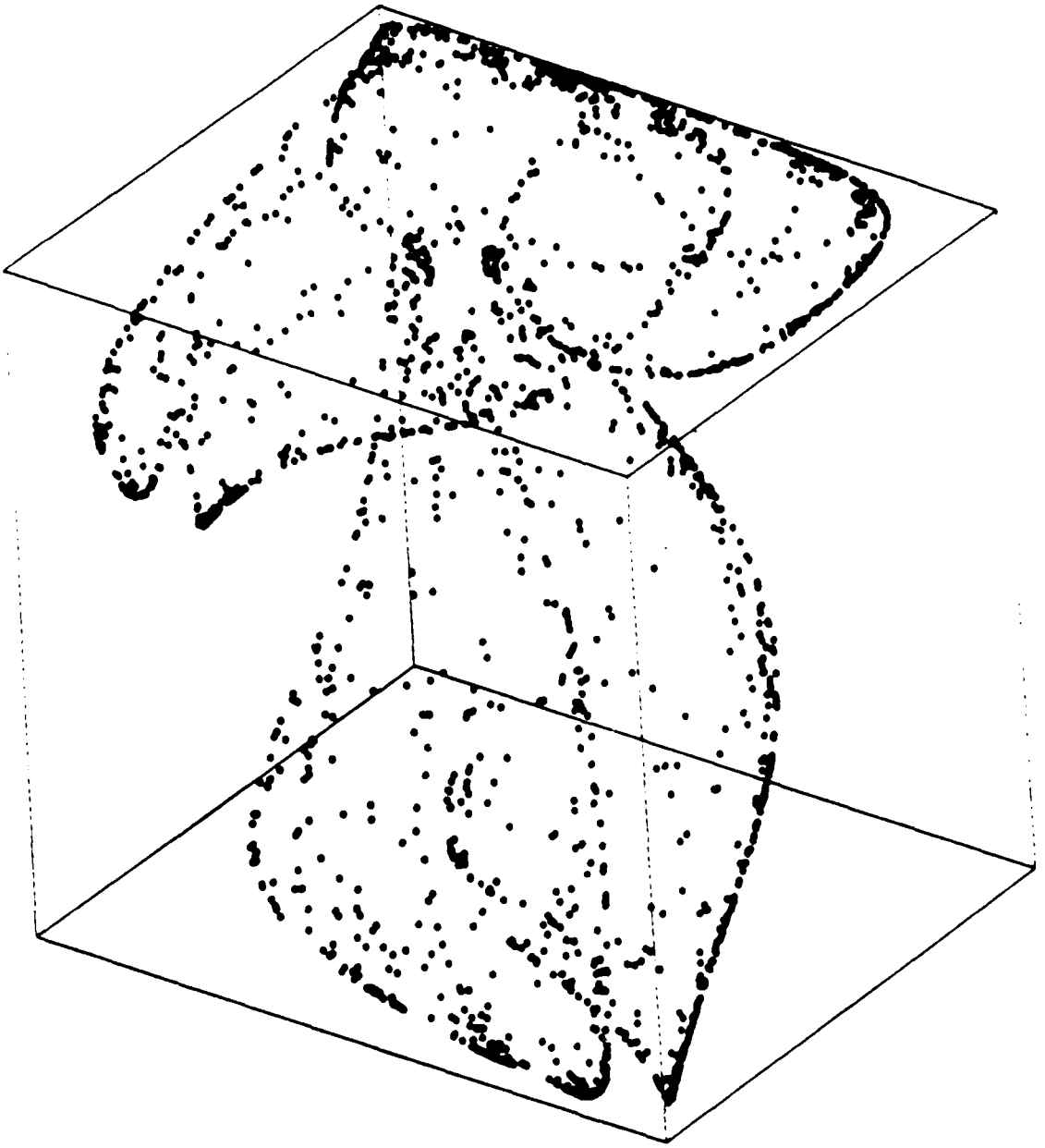
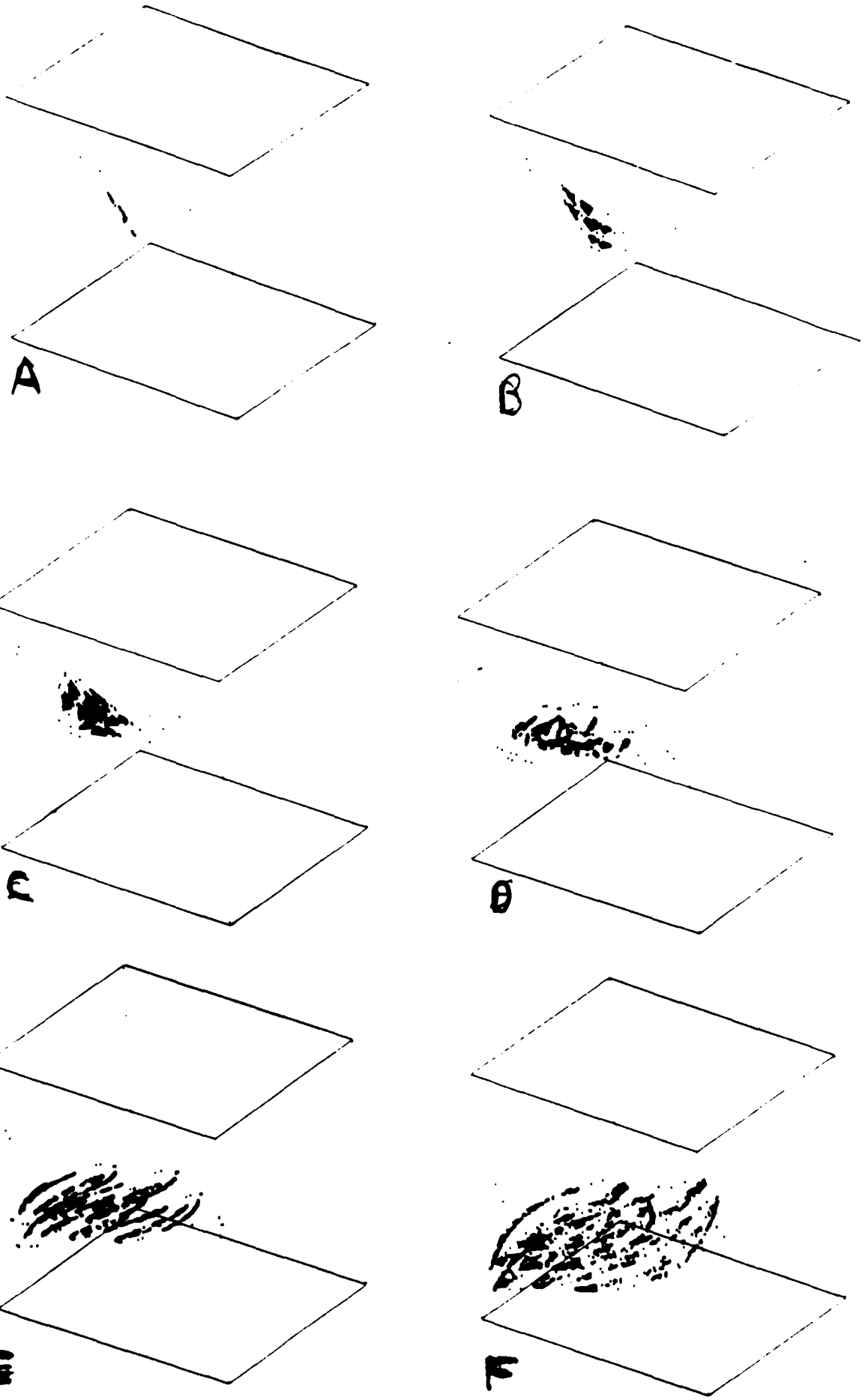


FIG 11

FIG 12




A
B
C
D
E
F

[90-JK-BPSIC]

The Ohio State University
Department of Computer and Information Science
Laboratory for Artificial Intelligence Research

Technical Report
April 1990

Back Propagation is Sensitive to Initial Conditions

John F. Kolen
Jordan B. Pollack
Laboratory for Artificial Intelligence Research
228 Bolz Hall, 2036 Neil Avenue
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

Note: OSU CIS LAIR Technical Report. A similar version entitled "Scenes from Exclusive Or: Back Propagation is Sensitive to Initial Conditions" has been submitted to the Twelfth Annual Conference of the Cognitive Science Society, Boston, MA, July 1990.

Back Propagation is Sensitive to Initial Conditions

John F. Kolen
Jordan B. Pollack

Laboratory for Artificial Intelligence Research
Computer and Information Science Department
The Ohio State University
Columbus, Ohio 43210, USA
kolen-j@cis.ohio-state.edu, pollack@cis.ohio-state.edu

ABSTRACT

This paper explores the effect of initial weight selection on feed-forward networks learning simple functions with the back-propagation technique. We first demonstrate, through the use of Monte Carlo techniques, that the magnitude of the initial condition vector (in weight space) is a very significant parameter in convergence time variability. In order to further understand this result, additional deterministic experiments were performed. The results of these experiments demonstrate the extreme sensitivity of back propagation to initial weight configuration.

April 18, 1990

Back Propagation is Sensitive to Initial Conditions

John F. Kolen
Jordan B. Pollack

Laboratory for Artificial Intelligence Research
Computer and Information Science Department
The Ohio State University
Columbus, Ohio 43210, USA
kolen-j@cis.ohio-state.edu, pollack@cis.ohio-state.edu

Introduction

Back Propagation (Rumelhart, Hinton, & Williams, 1986) is the network training method of choice for many neural network projects, and for good reason. Like other weak methods, it is simple to implement, faster than many other "general" approaches, well-tested by the field, and easy to mold (with domain knowledge encoded in the learning environment) into very specific and efficient algorithms.

Rumelhart et al. made a confident statement: for many tasks, "the network rarely gets stuck in poor local minima that are significantly worse than the global minima."(p. 536) According to them, initial weights of exactly 0 cannot be used, since symmetries in the environment are not sufficient to break symmetries in initial weights. Since their paper was published, the convention in the field has been to choose initial weights with a uniform distribution between plus and minus ρ, usually set to 0.5 or less.

The convergence claim was based solely upon their empirical experience with the back propagation technique. Since then, Minsky & Papert (1988) have argued that there exists no proof of convergence for the technique, and several researchers (Judd 1988; Blum and Rivest 1988; Kolen 1988) have found that the convergence time must be related to the difficulty of the problem, otherwise an unsolved computer science question ($P \stackrel{?}{=} NP$) would finally be answered. We do not wish to make claims about convergence of the technique in the limit (with vanishing step-size), or the relationship between task and performance, but wish to talk about a pervasive behavior of the technique which has gone unnoticed for several years: the sensitivity of back propagation to initial conditions.

The Monte-Carlo Experiment

Initially, we performed empirical studies to determine the effect of learning rate, momentum rate, and the range of initial weights on t-convergence (Kolen and Goel, to appear). We use the term *t–convergence* to refer to whether or not a network, starting at a precise initial configuration, could learn to separate the input patterns according to a boolean function (correct outputs above or below .5) within t epochs. The experiment consisted of training a 2-2-1 network on exclusive-or while varying three independent variables in 114 combinations: learning rate, η, equal to 1.0 or 2.0; momentum rate, α, equal to 0.0, 0.5, or 0.9; and initial weight range, ρ, equal to 0.1 to 0.9 in 0.1 increments, and 1.0 to 10.0 in 1.0 increments.

Each combination of parameters was used to initialize and train a number of networks.[1] Figure 1 plots the percentage of t-convergent (where t = 50,000 epochs of 4 presentations) initial conditions for the 2-2-1 network trained on the exclusive-or problem. From the figure we thus conclude the choice of $\rho \leq 0.5$ is more than a convenient symmetry-breaking default, but is quite necessary to obtain low levels of nonconvergent behavior.

Scenes From Exclusive-Or

Why do networks exhibit the behavior illustrated in Figure 1? While some might argue that very high initial weights (i.e. $\rho > 10$) lead to very long convergence times since the derivative of the semi-linear sigmoid function is effectively zero for large weights, this does not explain the fact that when ρ is between 2 and 4, the non-t-convergence rate varies from 5 to 50 percent.

Thus, we decided to utilize a more deterministic approach for eliciting the structure of initial conditions giving rise to t-convergence. Unfortunately, most networks have many weights, and thus many dimensions in initial-condition space. We can, however, examine 2-dimensional slices through the space in great detail. A slice is specified by an origin and two orthogonal directions (the X and Y axes). In the figures below, we vary the initial weights regularly throughout the plane formed by the axes (with the origin in the lower left-hand corner) and collect the results of running back-propagation to a particular time limit for each initial condition. The map is displayed with grey-level linearly related to time of convergence: black meaning not t-convergent and white representing the fastest convergence time in the picture. Figure 2 is a schematic representation of the networks used in this and the following experiment. The numbers on the links and in the nodes will be used for identification purposes. Figures 3 through 11 show several interesting "slices" of the the initial condition space for 2-2-1 networks trained on **exclusive-or**. Each slice is compactly identified by its 9-dimensional weight vector and associated learning/momentum rates. For instance, the vector (-3+2+7-4X+5-2-6Y) describes a network with an initial weight of -0.3 between the left hidden unit and the left input unit. Likewise, "+5" in the sixth position represents an initial bias of 0.5 to the right hidden unit. The letters "X" and "Y" indicate that the corresponding weight is varied along the X- or Y-axis from -10.0 to +10.0 in steps of 0.1. All the figures in this paper contain the results of 40,000 runs of back-propagation (*i.e.* 200 pixels by 200 pixels) for up to 200 epochs (where an epoch consists of 4 training examples).

Figures 12 and 13 present a closer look at the sensitivity of back-propagation to initial conditions. These figures zoom into a complex region of Figure 11; the captions list the location of the origin and step size used to generate each picture.

Sensitivity behavior can also be demonstrated with even simpler functions. Take the case of a 2-2-1 network learning the or function. Figure 14 shows the effect of learning "or" on networks (+5+5-1X+5-1Y+3-1) and varying weights 4 (X-axis) and 7 (Y-axis) from -20.0 to 20.0 in steps of 0.2. Figure 15 shows the same region, except that it partitions the display according to equivalent solution networks after t-convergence (200 epoch limit), rather than the time to convergence. Two networks are considered equivalent[2] if their weights have the

[1]Numbers ranged from 8 to 8355, depending on availability of computational resources. Those data points calculated with small samples were usually surrounded by data points with larger samples.

[2]For rendering purposes only. It is extremely difficult to know precisely the equivalence

same sign. Since there are 9 weights, there are 512 (2^9) possible network equivalence classes. Figures 16 through 25 show successive zooms into the central swirl identified by the XY coordinate of the lower-left corner and pixel step size. After 200 iterations, the resulting networks could be partitioned into 37 (both convergent and nonconvergent) classes. Obviously, the smooth behavior of the t-convergence plots can be deceiving, since two initial conditions, arbitrarily alike, can obtain quite different final network configuration.

Note the triangles appearing in Figures 19, 21, 23 and the mosaic in Figure 25 corresponding to the area which did not converge in 200 iterations in Figure 24. The triangular boundaries are similar to fractal structures generated under iterated function systems (Barnsley 1988): in this case, the iterated function is the back propagation learning method. We propose that these fractal-like boundaries arise in back-propagation due to the existence of multiple solutions (attractors), the non-zero learning parameters, and the non-linear deterministic nature of the gradient descent approach. When more than one hidden unit is utilized, or when an environment has internal symmetry or is very underconstrained, then there will be multiple attractors corresponding to the large number of hidden-unit permutations which form equivalence classes of functionality. As the number of solutions available to the gradient descent method increases, the more complicated the non-local interactions between them. This explains the puzzling result that several researchers have noted, that as more hidden units are added, instead of speeding up, back-propagation slows down (e.g. Lippman and Gold, 1987). Rather than a hill-climbing metaphor with local peaks to get stuck on, we should instead think of a many-body metaphor: The existence of many bodies does not imply that a particle will take a simple path to land on one. From this view, we see that Rumelhart et al.'s claim of back-propagation usually converging is due to a very tight focus inside the "eye of the storm".

Could learning and momentum rates also be involved in the storm? Such a question prompted another study, this time focused on the interaction of learning and momentum rates. Rather than alter the initial weights of a set of networks, we varied the learning rate along the X axis and momentum rate along the Y axis. Figures 26, 27, and 28 were produced by training a 3-3-1 network on 3-bit parity until t-convergence (250 epoch limit). Table 1 lists the initial weights of the networks trained in Figures 26 through 31. Examination of the fuzzy area in Figure 26 shows how small changes in learning and/or momentum rate can drasticly affect t-convergence (Figures 30 and 31).

Discussion

Chaotic behavior has been carefully circumvented by many neural network researchers (e.g. through the choice of symmetric weights by Hopfield (1982)), but has been reported in increasing frequency over the past few years (Choi and Huberman, 1983; Kurten and Clark, 1986; Babcock and Westervelt, 1987; Derrida and Meir, 1988; Riedal et al., 1988; Sompolinsky et al., 1988). Connectionists, who use neural models for cognitive modeling, disregard these reports of extreme non-linear behavior in spite of common knowledge that non-linearity is what enables network models to perform non-trivial computations in the first place. All work to date has noticed various forms of chaos in network dynamics, but not in learning dynamics. Even if back-propagation is shown to be non-chaotic in the limit, this still does not

classes of solutions, so we approximated.

preclude the existance of fractal boundaries between attractor basins since other non-chaotic non-linear systems produce such boundaries (i.e. forced pendulums with two attractors (D'Humieres et al., 1982))

What does this mean to the back-propagation community? From an engineering applications standpoint, where only the solution matters, nothing at all. When an optimal set of weights for a particular problem is discovered, it can be reproduced through digital means. From a scientific standpoint, however, this sensitivity to initial conditions demands that neural network *learning* results must be specially treated to guarantee replicability. When theoretical claims are made (from experience) regarding the power of an adaptive network to model some phenomena, or when claims are made regarding the similarity between psychological data and network performance, **the initial conditions for the network need to be precisely specified or filed in a public scientific database.**

What about the future of back-propagation? We remain neutral on the issue of its ultimate convergence, but our result points to a few directions for improved methods. Since the slowdown occurs as a result of global influences of multiple solutions, an algorithm for first factoring the symmetry out of both network and training environment (e.g. domain knowledge) may be helpful. Furthermore, it may also turn out that search methods which harness "strange attractors" ergodically guaranteed to come arbitrarily close to some subset of solutions might work better than methods based on strict gradient descent. Finally, we view this result as strong impetus to discover how to exploit the information-creative aspects of non-linear dynamical systems for future models of cognition (Pollack 1989).

Acknowledgments

This work was supported by Office of Naval Research grant number N00014-85-J1200 Substantial free use of over 200 Sun workstations was generously provided by our department.

References

K. L. Babcock and R. M. Westervelt. 1987. Dynamics of simple electronic neural networks, *Physica*, **28D**:305-316.

M. Barnsley. 1988. *Fractals Everywhere*, Academic Press, San Diego, CA. 1988.

A. Blum and R. Rivest. 1988. Training a 3-node Neural Network is NP-Complete. *Proceedings of IEEE Conference on Neural Information Processing Systems*, Denver, Colorado, 1988.

M. Y. Choi and B. A. Huberman. 1983. Dynamic behavior of nonlinear networks, *Physical Review A*, **28**:1204-1206.

J. J. Hopfield. 1982. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings US National Academy of Science* **79**:2554-2558.

B. Derrida and R. Meir. 1988. Chaotic behavior of a layered neural network. *Physical Review A*, **38**:3116-3119.

D. D'Humieres, M. R. Beasley, B. A. Huberman, and A. Libchaber. 1982. Chaotic States and Routes to Chaos in the Forced Pendulum. *Physical Review A*, **26**:3483-96.

B. A. Huberman and T. Hogg. 1987. Phase Transitions and Artificial Intelligence. *Artificial Intelligence*, **33**:155-172.

S. Judd. 1988. Learning in Networks is Hard. *Journal of Complexity* **4**:177-192.

J. Kolen. 1988. Faster Learning Through a Probabilistic Approximation Algorithm. *Proceedings of the Second IEEE International Conference on Neural Networks*, San Diego, California, pp. I:449-454.

J. Kolen and A. Goel. To appear. Learning in Parallel Distributed Processing Networks: Computational Complexity and Information Content. *IEEE Transactions on Systems, Man, and Cybernetics*.

K. E. Kurten and J. W. Clark. 1986. Chaos in Neural Networks. *Physics Letters*, **114A**, 413-418.

R. P. Lippman and B. Gold. 1987. Neural Classifiers Useful for Speech Recognition. In *1st International Conference on Neural Networks* ,IEEE, IV:417-426.

M. L. Minsky and S. A. Papert. 1988. *Perceptrons*. Cambridge, MA: MIT Press.

J. B. Pollack. 1989. Implications of Recursive Auto Associative Memories. In *Advances in Neural Information Processing Systems*. (ed. D. Touretzky) pp 527-536, Morgan Kaufman, San Mateo.

U. Riedal, R. Kuhn, and J. L. van Hemmen.1988. Temporal sequences and chaos in neural nets, *Physical Review A*, **38**:1105-1108.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Learning Representation by Back-Propagating Errors. *Nature* **323**:533-536.

H. Sompolinsky, A. Crisanti, H. J. Sommers. 1988. Chaos in random neural networks. *Physical Review Letters*, **61**:259-262.

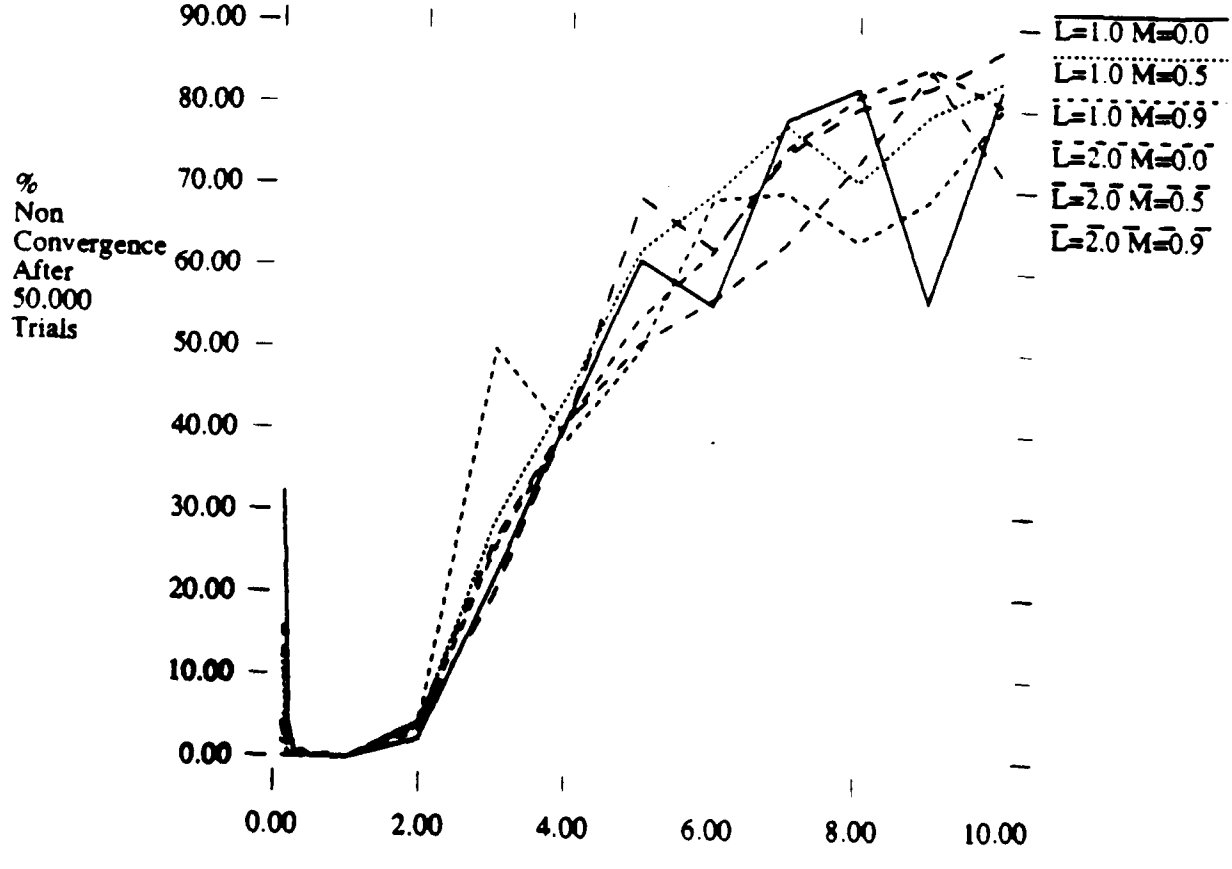


Figure 1: Percentage T-Convergence vs. Initial Weight Range

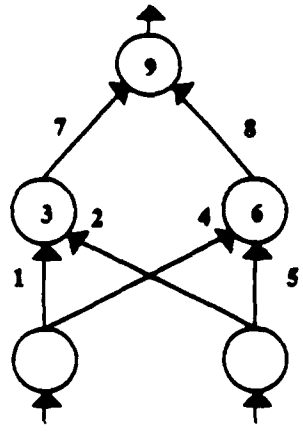


Figure 2 : Schematic Network

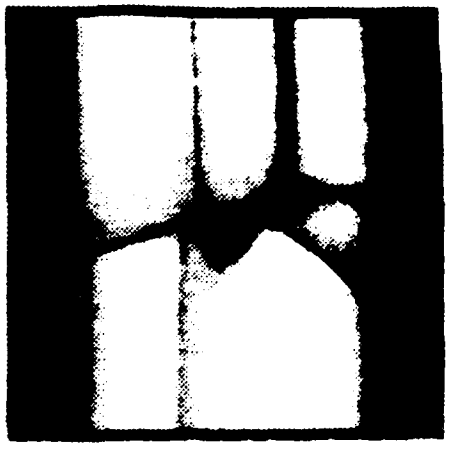


Figure 3 : (-5-3+3+6Y-1-6+7X) η=3.25 α=0.40

Figure 4 : (+4-7+6+0-3Y+1X+1) η=2.75 α=0.00

Figure 5 : (-5+5+1-6+3XY+8+3) η=2.75 α=0.80

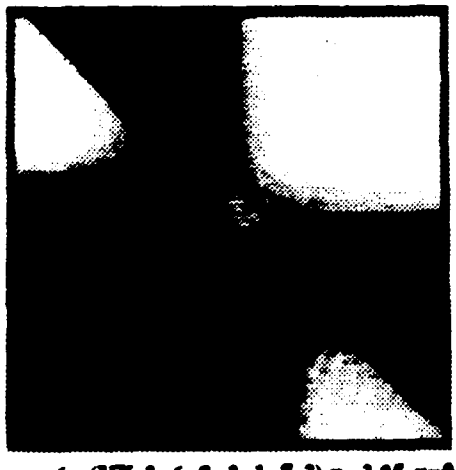


Figure 6 : (YX-3+6+8+3+1+7-3) η=3.25 α=0.00

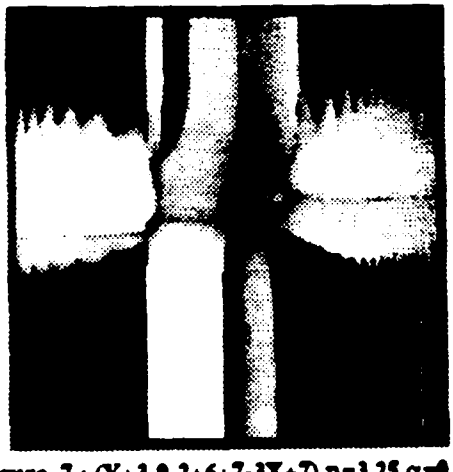


Figure 7 : (Y+3-9-2+6+7-3X+7) η=3.25 α=0.60

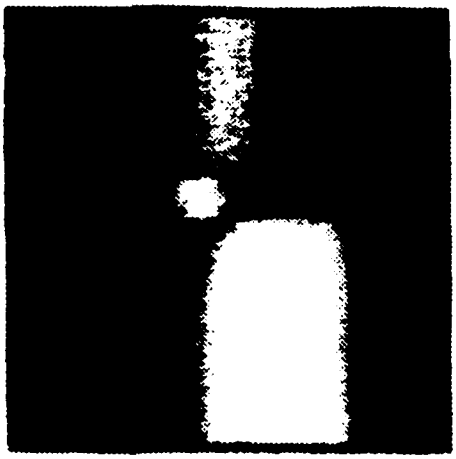


Figure 8 : (-6-4XY-6-6+9-4-9) η=3.00 α=0.50

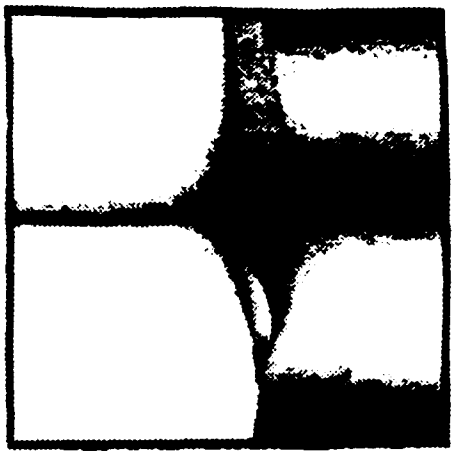


Figure 9 : (-2+1+9-1X-3+8Y-4) η=2.75 α=0.20

Figure 10 : (+1+8-3-6X-1+1+8Y) η=3.50 α=0.90

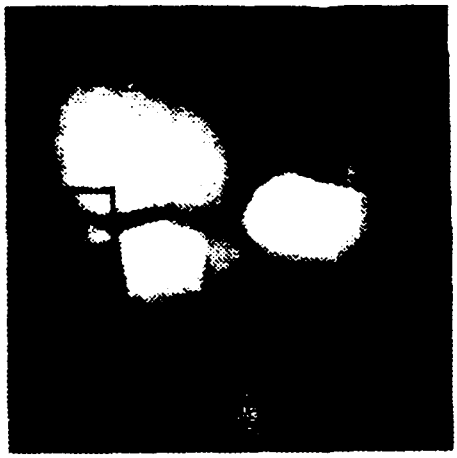


Figure 11 : (+7+4-9-9-5Y-3+9X) η=3.00 α=0.70

Figure 12 : (-9.0,-1.8) step 0.018

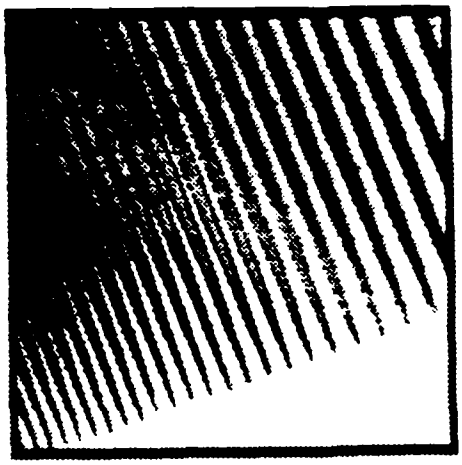


Figure 13 : (-6.966,-0.500) step 0.004

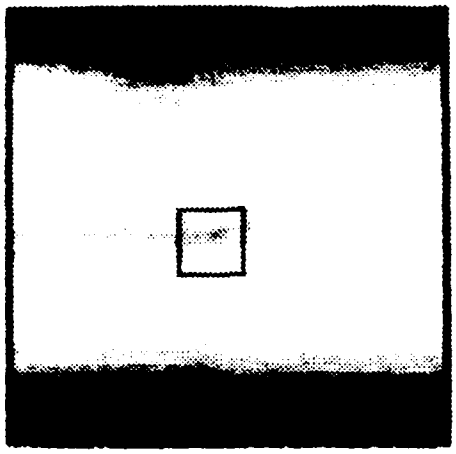


Figure 14 : (-20.000000, -20.000000) step 0.200000

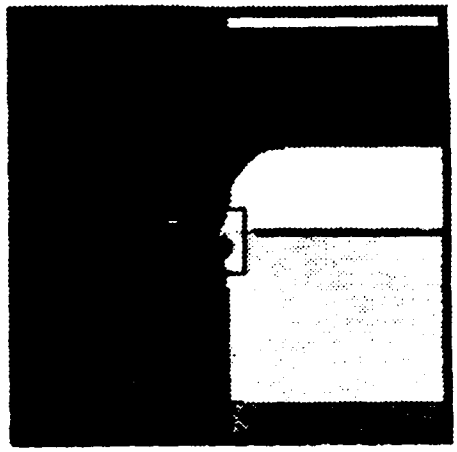


Figure 15 : Solution Networks

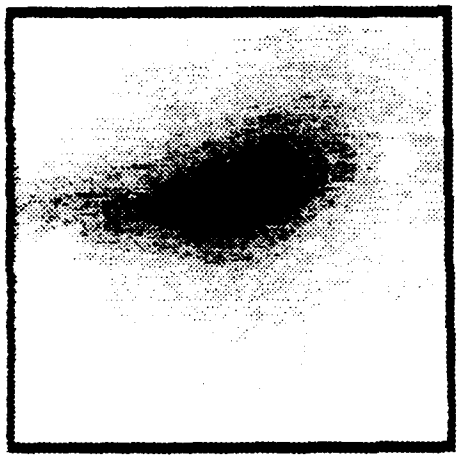


Figure 16 : (-4.500000, -4.500000) step 0.030000

Figure 17 : Solution Networks

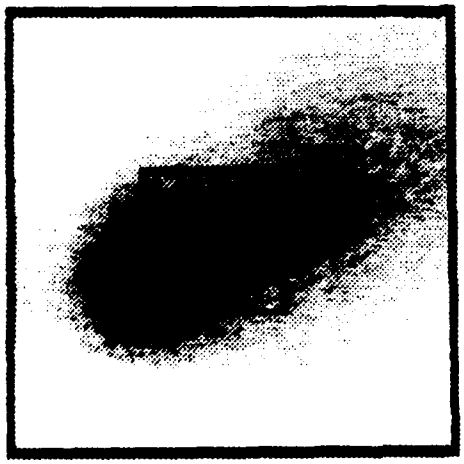


Figure 18 : (-1.680000, -1.350000) step 0.002400

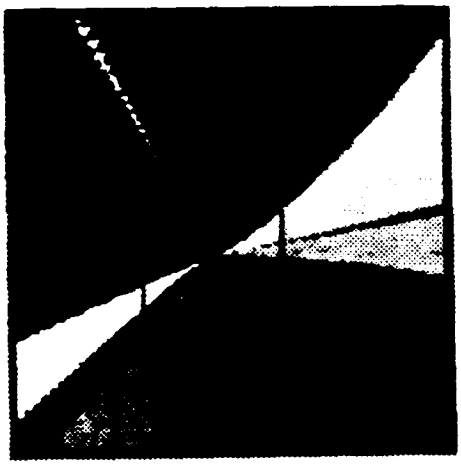


Figure 19 : Solution Networks

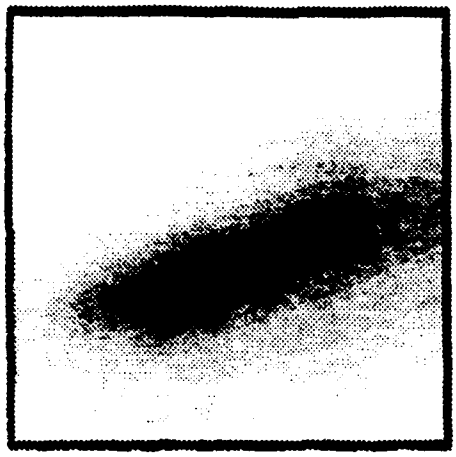


Figure 20 : (-1.536000, -1.197000) step 0.000780

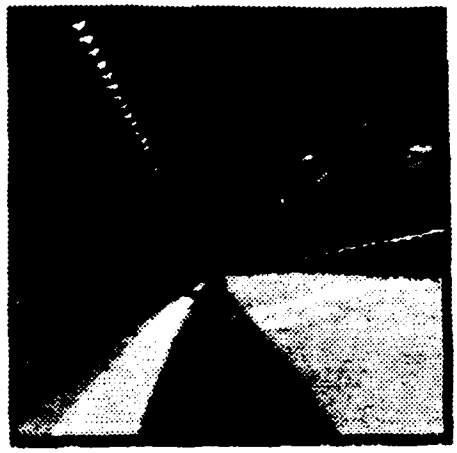


Figure 21 : Solution Networks

Figure 22 : (-1.472820, -1.145520) step 0.000070

Figure 23 : Solution Networks

Figure 24 : (-1.467150, -1.140760) step 0.000016

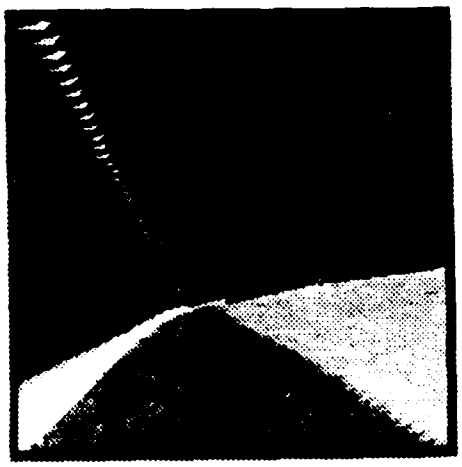


Figure 25 : Solution Networks

Figure 26 : η=(0.0,4.0) α=(0.0,1.25)

Figure 27 : η=(0.0,4.0) α=(0.0,1.25)

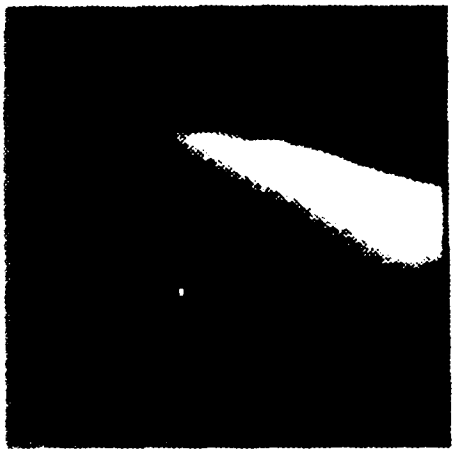


Figure 28 : η=(0.0,4.0) α=(0.0,1.25)

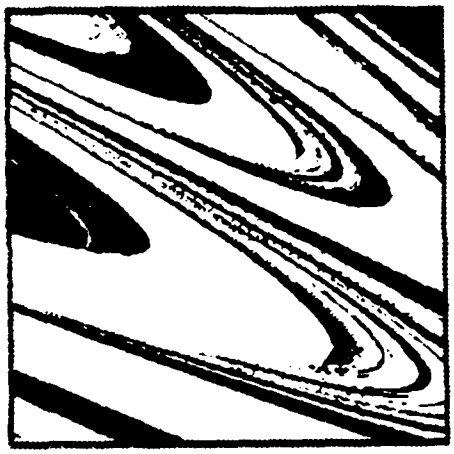


Figure 29 : η=(3.456,3.504) α=(0.835,0.840)

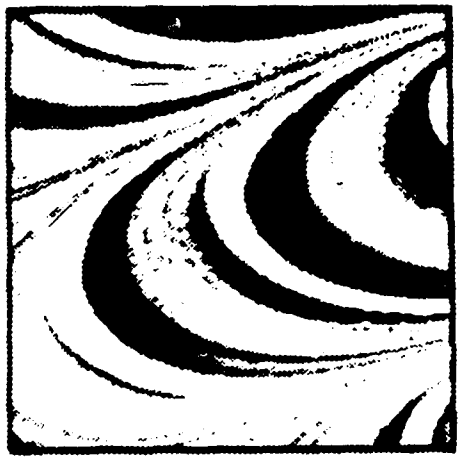


Figure 30 : η=(3.84,3.936) α=(0.59,0.62)

	Figure 26 Figure 29 Figure 30	Figure 27	Figure 28
Weight 1	-0.34959000	-0.34959000	-0.34959000
Weight 2	0.00560000	0.00560000	0.00560000
Weight 3	-0.26338813	0.39881098	0.65060705
Weight 4	0.75501968	-0.16718577	0.75501968
Weight 5	0.47040862	-0.28598450	0.91281711
Weight 6	-0.18438011	-0.18438011	-0.19279729
Weight 7	0.46700363	-0.06778983	0.56181073
Weight 8	-0.48619500	0.66061292	0.20220653
Weight 9	0.62821201	-0.39539510	0.11201949
Weight 10	-0.90039973	0.55021922	0.67401200
Weight 11	0.48940201	0.35141364	-0.54978875
Weight 12	-0.70239312	-0.17438740	-0.69839197
Weight 13	-0.95838741	-0.07619988	-0.19659844
Weight 14	0.46940394	0.88460041	0.89221204
Weight 15	-0.73719884	0.67141031	-0.56879740
Weight 16	0.96140103	-0.10578894	0.20201484

Table 1: Network Weights for Figures 26 through 30

Subdimensional Indexing for Connectionist Case-Based Memory Retrieval

Gregory M. Saunders and Jordan B. Pollack

Laboratory for Artificial Intelligence Research
Computer and Information Science Department
Ohio State University, Columbus, Ohio 43201
saunders@cis.ohio-state.edu, pollack@cis.ohio-state.edu

Abstract

This paper introduces a new approach to case-based memory retrieval (CBMR). Schema-like objects are first encoded using a recursive connectionist network which maps variable-sized symbolic structures into vectors in a high-dimensional space such that similar structures map to nearby vectors. Retrieval is then performed using Euclidean nearest neighbor search on the vectors. We first show that if distances are calculated using the entire vector, retrieval is imprecise because the principal components of the representation wash out the dimensions that code the deeper similaries of the symbolic structures. We then demonstrate that this problem can be overcome by using only a subset of the dimensions, i.e., by using subdimensional indexing. These results show promise for a new method of CBMR, based on a content-addressable memory for patterns.

1 Introduction

Efficient retrieval is a central aspect of any model of memory (Schank, 1982), and is especially critical for schema or case-based reasoning systems, as the DARPA report (1989, p.1) points out:

> The biggest bottleneck is in choosing the best cases to reason with; this potentially massive search problem is handled currently through indexing. While the choice of indices still needs to be better understood, it is possible that the use of parallel implementations make this problem doable in real time.

Our approach to this problem is based on *Recursive Auto-Associative Memory* (RAAM) (Pollack, 1988; in press), a connectionist architecture which can develop fixed-width distributed representations for variable-sized symbolic trees and lists.

A RAAM consists of two functions: a *compressor* and a *reconstructor*, both implemented with standard feed-forward networks, and simultaneously trained. The former is used to recursively compress, from the bottom up, an arbitrary structure into a fixed-width pattern vector; the latter is used to unpack the pattern, from the top down, to recover the original structure.

The resulting vectors are such that similar cases have similar representations. This allows us to perform case retrieval using a nearest-neighbor algorithm as follows. A query is encoded into a vector by the compressor and the cases are retrieved in order of ascending Euclidean distance from the query using only some *mask*, i.e., by using only some subset of the dimensions. We later show how such masks are calculated.

Other researchers have also adopted a connectionist approach towards case-based retrieval (Thagard and Holyoak, 1989; Becker and Jazayeri, 1989; Owens, 1989). Owens discusses the problems of feature extraction and search, emphasizing that similarity in retrieval should be judged by *abstract* or *thematic* features. He argues that connectionist networks are fairly good at performing search once the features have been learned, but that they are poor at learning the appropriate abstract features. As he states (p. 164):

> Connectionist approaches... are very good at deciding how to weight features to measure case similarity, but they do not deal with the problem of how raw data gets turned into sets of features in the first place, nor do they deal with how new features can be learned.... How are features extracted from raw input?

He argues that the processes of feature extraction and memory search should be integrated, but that this is difficult because feature derivation is difficult. We propose that our approach is a possible solution to this problem, since the RAAM automatically learns the features necessary to encode the deeper similarities inherent in the cases.

2 Subdimensional indexing using RAAMs

We begin by formalising the relevant aspects of a case-based reasoning (CBR) system. A *mask* M is simply some subset of the dimensions of a vector space. A CBR system, then, is defined to be a 5-tuple (S, f, L, Q, d_M),[1] where

[1] It could actually be a 4-tuple, since S and f determine L, but a 5-tuple is conceptutally clearer.

S = $\{C_1, C_2, ..., C_n\}$, a set of cases, usually cast as compositional symbolic structures.

f = indexing function, which maps cases into their representations in memory. Hence, memory is defined as...

L = long-term case memory = $\{f(C_i) : 1 \leq i \leq n\}$.

Q = query = $f(C^*)$, where C^* is the symbolic representation for the case to be retrieved.

d_M = a distance metric over L (where the subscript M denotes the dependence on a *mask*), used to judge the similarity between cases in L and the query Q. Hence $d : Q \times L \Rightarrow \Re$

Thus our view of CBMR begins with a set of cases S. Each case is encoded via some indexing function f, generating a memory list L. To perform retrieval on some case C^*, we first generate a query $Q = f(C^*)$. Then the distance function d_M can be applied, so that the cases C_i that are retrieved will those that minimize $d(f(C_i), Q)$.

How is this implemented by the RAAM architecture? When a RAAM is used to transform a set of cases into a set of vectors of dimension k, this automatically specifies a set of 2^k (implicit) indexing functions f, one for each possible mask M. If distance between a case C_i and a query Q is defined in terms of masked Euclidean distance, then the different masks define 2^k different distance functions:

$$d_M(C_i, Q) = \sqrt{\sum_{j \in M} (f(C_i)_j - Q_j)^2}$$

Many of these, of course, will be redundant, but many will also be useful for specialized forms of retrieval which currently involves custom programming.

Why should this method work? Convergence of the network implies that the RAAM is somehow storing the schema-like cases as patterns. Unlike symbolic representations, however, these patterns are distributed, so that similar cases have similar representations. The problem is that the clustering of the patterns captures only structural similarity, and thus only the principal dimensions (of maximal variance) really affect the straightforward Euclidean distance calculation. Using masks allows Euclidean distance to be sensitive to deeper similarities of the symbolic structures.

Nearest neighbor search of such structures is algorithmically efficient — using multidimensional divide-and-conquer it is $O(n \log^{k-1} n)$ if we have n k-dimensional points (Bentley, 1980) — and potentially can be implemented by analog circuitry (Hopfield, 1982). The difficulty lies in determining the appropriate mask for a given query. We have devised a greedy algorithm to perform this task, described below.

Suppose that for a given query Q, we wish to find a mask that will retrieve a set of goal cases $G = \{C_{i_1}, C_{i_2}, ..., C_{i_g}\}$. Each different mask M yields a distance function d_M, which in turn yields a permutation σ of the case-base (where the order $\sigma(i)$ of case C_i is determined by $d_M(C_i, Q)$). The score for a mask is then defined to be

$$S_M = \frac{1}{g} \sum_{j \in G} \left\{ \begin{array}{ll} 1, & \sigma(j) \leq g \\ \frac{1}{\sigma(j)-g}, & \sigma(j) > g \end{array} \right\}$$

This function is really quite straightforward. For each goal case $j \in G$, look at how j ranked in the permutation σ for the mask. If $\sigma(j) \leq g$, then C_j would be one of the top g cases retrieved by M, so add 1 to the score. Otherwise the contribution to the score from j is an exponentially decaying function based upon how close j was to being in the top g cases retrieved. Finally, the summation is divided by g to scale the result, so that perfect masks will have a score of 1, and very bad masks will have a score approaching 0.

Our algorithm, then, was simply to build up the mask M by adding dimensions one at a time, being greedy on the score S_M. The program halted when it could find no dimension that when added to the current mask would increase its score. (No backtracking was used.)

3 Results

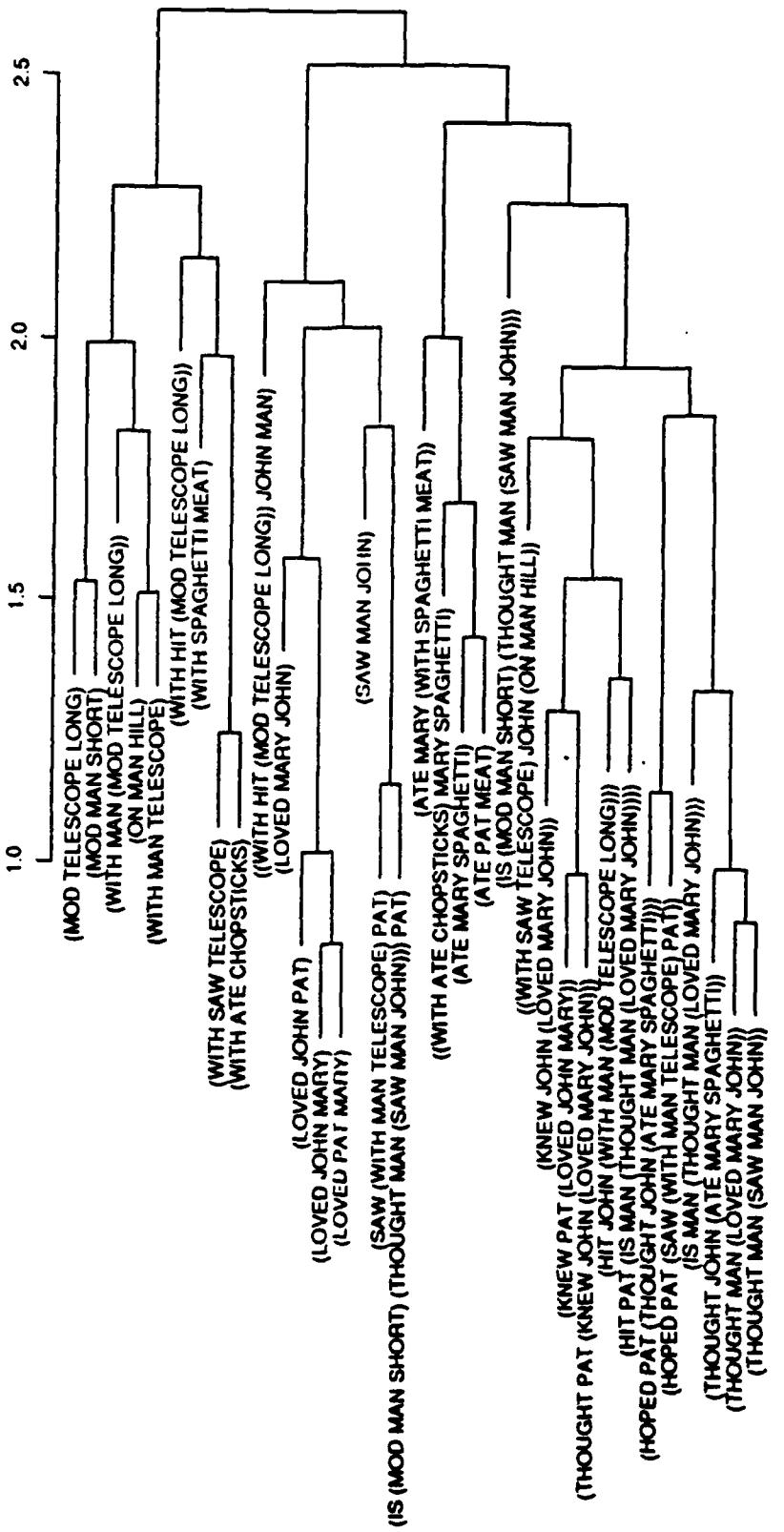
We have implemented subdimensional indexing using the propositions in Pollack (1988) as cases:[2]

1. (loved pat mary)
2. (loved john pat)
3. ((with saw telescope) john (on man hill))
4. ((with ate chopsticks) mary spaghetti)
5. (ate mary (with spaghetti meat))
6. (ate pat meat)
7. (knew pat (loved john mary))
8. (thought pat (knew john (loved mary john)))
9. (hoped pat (thought john (ate mary spaghetti)))
10a. ((with hit (mod telescope long)) john man)
10b. (hit john (with man (mod telescope long)))
11. (hoped pat (saw (with man telescope) pat))
12. (hit pat (is man (thought man (loved mary john))))
13. (saw (is (mod man short) (thought man (saw man john))) pat)

In terms of the notation developed throughout this paper, $n = 14$, and $k = 16$, i.e., there were 14 cases and the RAAM devised a 16-dimensional representation to store them. The

[2]Case 10 was ambigous and had two possible interpretations. See Pollack (1988) for more details.

clustering diagram below is based upon the entire RAAM representation and shows the similarity between patterns in the full 16-dimensional space. (Note that there are more than 14 nodes in this diagram because a RAAM also develops representations for all of the "subcases".)



We tested this case-base with many different queries, two of which were *(nil pat nil)* and *(loved nil nil)*. The former was intended to retrieve all those cases in which *pat* was an agent. The latter was applied with two different intentions: to retrieve stories **about** *love*, i.e., ones that contained *love* as the action, and to retrieve stories **involving** love, i.e., ones which contained the proposition *love* at any level in the story.

The table below compares the effect of using *masks* for retrieval (as discovered by our greedy algorithm) vs. using the entire k-dimensional representations developed by the RAAM. (σ_M = order in masked permutation, σ_u = order in unmasked permutation, *i.e.*, by using the entire representation, S_M = masked score, S_u = unmasked score.)

Query	Goal cases (ordered by σ_M)	σ_M	σ_u	S_M	S_u
(nil pat nil)	(ate pat meat)	1	3	1	.43
	(knew pat (loved john mary))	2	13		
	(hoped pat (saw (with man telescope) pat))	3	11		
	(thought pat (knew john (loved mary john)))	4	8		
	(loved pat mary)	5	1		
	(hoped pat (thought john (ate mary spaghetti)))	6	20		
	(hit pat (is man (thought man (loved mary john))))	7	15		
(loved nil nil) **about** love	(loved mary john)	1	1	1	.51
	(loved john pat)	2	6		
	(loved pat mary)	3	5		
	(loved john mary)	4	8		
(loved nil nil) **involving** love	(knew pat (loved john mary))	2	28	.68	.44
	(loved pat mary)	3	5		
	(thought pat (knew john (loved mary john)))	4	27		
	(loved john pat)	5	6		
	(loved mary john)	9	1		
	(knew john (loved mary john))	10	32		
	(thought man (loved mary john)	12	21		
	(is man (thought man (loved mary john)))	14	24		
	(hit pat (is man (thought man (loved mary john))))	16	22		
	(loved john mary)	22	8		

4 Conclusions

Our approach to CBMR involves using a RAAM to develop k-dimensional representations for a set of symbolic cases, and then applying subdimensional indexing to retrieve the cases most similar to a given query. Feature extraction occurs automatically because it is learned by the RAAM, and will be dynamic because any new case that is added to the case-base will involve a slight adjustment of the weights between levels, which will in turn slightly adjust the representations of each case.

The above results clearly show that similarity judments using *masked* Euclidean distance are more effective than those based on then entire representation. Furthermore, the difference in results between querying for cases **about** love vs. querying for cases **involving** love shows that subdimensional indexing is also effective at capturing the deeper similarities in the symbolic cases (Owen's so-called thematic features).

The biggest open issue is the scalability of RAAMs. We are currently trying to train a RAAM on a larger, more realistic case-base, excerpted from Hammond (1989). However, in CBR applications, since the feature representations built by RAAM never need to be reconstructed, absolute convergence of the RAAM is not crucial to construct the indices. Furthermore, while the current technique is rigidly syntactic in nature, relying on the semantic order implicit in the case-base designer's choice of a uniform symbol structure, it is certainly very straightforward to add thematic similarity feedback to a RAAM's training regimen. Thus a nearest-neighbor subdimensional indexing scheme should be a powerful and elegant weapon added to the case-based memory retrieval arsenal.

Acknowledgements

Pollack is supported by Office of Naval Research Grant N00014-89-J-1200. Saunders is supported by a National Science Foundation Graduate Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the Office of Naval Research or the National Science Foundation.

References

[Becker and Jasayeri, 1989] Becker, L. and Jasayeri, K. (1989). A connectionist approach to case-based reasoning. *Proceedings: Case-Based Reasoning Workshop*, pages 213–217.

[Bentley, 1980] Bentley, J. L. (1980). Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229.

[DARPA, 1989] DARPA (1989). Case-based reasoning. *Proceedings: Case-Based Reasoning Workshop*, pages 1–13.

[Hammond, 1989] Hammond, K. J. (1989). *Case-Based Planning: Viewing planning as a memory task.* Academic Press, Inc., San Diego.

[Hopfield, 1982] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA*, 79:2554–2558.

[Kolodner, 1989] Kolodner, J. L. (1989). Selecting the best case for a case-based reasoner. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pages 155–162. Ann Arbor, MI.

[Owens, 1989] Owens, C. (1989). Integrating feature extraction and memory search. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pages 163–170. Ann Arbor, MI.

[Pollack,] Pollack, J. B. Recursive distributed representations. To appear in *Artificial Intelligence.*

[Pollack, 1988] Pollack, J. B. (1988). Recursive auto-associative memory: Devising compositional distributed representations. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, pages 33–39. Montreal.

[Schank, 1982] Schank, R. C. (1982). *Dynamic Memory.* Cambridge University Press, Cambridge.

[Thagard and Holyoak, 1989] Thagard, P. and Holyoak, K. J. (1989). Why indexing is the wrong way to think about analog retrieval. *Proceedings: Case-Based Reasoning Workshop*, pages 36–40.

Avoiding Fusion in Floating Symbol Systems

Peter J. Angeline*and Jordan B. Pollack
Department of Computer and Information Sciences
Laboratory for Artificial Intelligence Research
The Ohio State University
Columbus, Ohio 43210

May 16, 1990

Abstract

Miikkulainen and Dyer (1988) have demonstrated a method, called extended back-propagation (XBP) whereby a back propagation network can construct appropriate input representations by propagating error back to a "virtual" localist input layer. Our investigation showed that when applied in a auto-associative fashion these "floating symbols" tend to fuse, which is a significant problem for modern connectionist representation research. In this paper, we present a method, Pushprop, which allows auto-associative networks to develop floating symbols which do not fuse, and are not artificially constrained with embedded constant features, large initial separation, or unique output patterns. Our initial results use this technique on a prototype application in the domain of optical character recognition.

*All correspondence to this author. POSTER - Cognitive Science and AI: Representational Issues

Avoiding Fusion in Floating Symbol Systems

Peter J. Angeline* and Jordan B. Pollack

Laboratory for Artificial Intelligence Research

Department of Computer and Information Sciences

The Ohio State University

Columbus, Ohio 43210

May 17, 1990

1 Summary

One of the historical problems for connectionist systems has been the arbitrary nature of neural representations, usually designed by a researcher taking advantage of domain knowledge. Sometimes it is possible to construct representations which non-trivially encode a significant amount of knowledge, greatly simplifying the associative task, e.g. [2] and [12]. One of the potential benefits of modern connectionist work is that certain architectures can automatically construct complex representations during the acquisition of a task, allowing the relationships between learning and representation to be studied in detail [3].

We use the term *floating symbol systems* to denote a broad class of these connectionist models which allow complex distributed representations (symbols) to co-evolve (float) with the weights in a neural network system (e.g. [8], [1] and [10]). As a starting point, Miikkulainen and Dyer in [8] have demonstrated a method, called extended back-propagation (XBP) whereby a back propagation [11] network can construct

*All correspondence to this author. POSTER - Cognitive Science and AI: Representational Issues

appropriate input representations by propagating error back to a "virtual" localist input layer.[1] However, this process does not reflect all the constraints necessary for the construction of *distinct* input pattern representations. Our initial motivation for this project was to remove some of the arbitrary character of the learning environment for Recursive Auto-Associative Memory (RAAM) [9], which devises fixed-width representations for recursive data structures. In attempting to float the input representations for RAAM, we found that the system could converge in a few epochs with all input representations indistinguishable! In further work on simple auto-association, it became quite clear that floating symbols tend to "fuse".

Miikkulainen and Dyer did not face this *symbol fusion problem*, since they started with very distinct input patterns and assumed unique output patterns, both of which help maintain separation of the floating input symbols. In subsequent work, Miikkulainen [7] has used partially floating symbols in which some features are preset and not modified by learning. Elman, as well, did not observe fusion in his word clustering experiments since he did not allow his networks to converge but stopped after only a few epochs, well before fusion occurred.

In this paper, we present a method, "Pushprop," which allows auto-associative networks to develop floating symbols which do not fuse, and are not artificially constrained with embedded constant features, large initial separation, unique output patterns or incomplete training. Our initial results use this technique on a prototype RAAM application in the domain of optical character recognition.

The Pushprop algorithm enforces a pre-specified separation between every pair of floating symbols. This is accomplished by modifying the error function used to compute partial derivatives for the output of a feedforward network to globally push apart patterns which threaten to fuse. The error function is based on the Euclidean distance, $d(p_i, p_j)$, between two input pattern representations and is defined as:

$$E_{ij} = \frac{1}{2}(\sigma - d(p_i, p_j))^2$$

where σ is the minimum allowed Euclidean distance between pattern p_i and pattern p_j. We can thus derive the update algorithm for a component in an floating symbol by inserting the above error function equation

[1] Geoff Hinton [4] was the first to note that the weights from a fully local input layer to a more compact hidden layer are a representation.

Figure 1: Strokes used in the representation of characters.

into the derivation for the Back Propagation algorithm given by Rumelhart et al.[2] The resulting update to a specific position in an input pattern is then:

$$\frac{\partial E}{\partial p_{in}} = \frac{p_{in} - p_{jn}}{d(p_i, p_j)}(-\sigma + d(p_i, p_j))$$

where p_{in} is the nth position in the ith input pattern representation. We can now use a gradient descent on E_{ij} to effectively separate each of the input patterns from the others. By running this separation method after updating the input pattern representations with XBP, the representations are guaranteed to be distinct by at least σ regardless of the initial conditions.

To test the algorithm, both XBP and Pushprop algorithm were run on a 10-5-10 RAAM which constructed representations for letters from sets of vertical and horizontal line segments. The letters T, L, J, F, C, U, N, H, I, O, A and E were represented as binary trees using floating symbols for the 6 distinct visual line segments or "strokes" shown in Figure 1, as HT, HM, HB, VL, VM, and VR. Table 1 shows the tree encodings for each of the 12 letters. While the representations for each of the letters is being learned by the RAAM, the XBP method is adapting the representations for the 6 floating symbols, either alone or

[2]Michael Jordan [5] has anticipated most uses of soft constraints to guide neural network learning, and while he never used this type of "repulsive" constraint, our work is clearly in this tradition.

Figure 2: Graph showing the converged sumbol separation (y-axis) versus the initial symbol separation (x-axis). Dark lines connect the data points for Run 1 and and the light lines connect the data points for Run 2. Dashed lines mark performance of Pushprop while solid lines mark the performance of XBP. For each run, the initial weights for the RAAM were identical in each trial and for each algorithm.

with the Pushprop separation being applied after each epoch.

We chose 18 different initial floating symbol sets (with varying degrees of separation), and a single set of initial weights for the network per run.[3] Using just the XBP technique, only 8 out of the 18 sets of initial symbol sets did not fuse. This is shown graphically in Figure 2. It is evident from the experiments that if the initial separation between the input pattern representations is very small, auto-association usually forces some subset of them to fuse. As the initial separation between the floating symbols increases, the chances that XBP alone will construct distinct representations for all the input patterns increases. In contrast, when using the Pushprop with $\sigma = 0.1$, all 18 test cases converged with distinct floating symbols.

[3] Because Back-propagation has been shown to be very sensitive to initial conditions [6].

letter	*tree of strokes*	*letter*	*tree of strokes*	*letter*	*tree of strokes*
T	(ht vm)	L	(hb vl)	J	(vr hb)
F	((hm ht) vl)	I	((ht hb) vm)	U	(hb (vl vr))
N	(ht (vl vr))	H	(hm (vl vr))	C	((ht hb) vl)
O	((ht hb) (vl vr))	A	((hm ht) (vl vr))	E	((hm ht) (hb vl))

Table 1: Tree Representations for Characters

While our initial results have applied the Pushprop method for avoiding fusion of floating symbols to our RAAM work, the method is clearly applicable to most self-supervised auto-associative architectures. Further, the basic theory behind the Pushprop method is an interesting addition to the connectionist toolbox in that it offers a new "least commitment" style of constraining the evolution of complex connectionist representations: each symbol develops its own niche, sufficiently separated from neighbors.

References

[1] J. L. Elman. Finding structure in time. Technical Report CRL 8801, University of California, San Diego, April 1988.

[2] M. A. Gluck, G. H. Bower, and M. R. Hee. A configural-cue network model of animal and human associative learning. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, 1989. Ann Arbor, Michigan.

[3] S. J. Hanson and D. J. Burr. What connectionist models learn: Learning and representation in connectionist networks. *Behavioral and Brain Science*, 1989. To Appear.

[4] G. E. Hinton. Learning distributed representations of concepts. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12, 1986. Amherst, MA.

[5] M. I. Jordan. Supervised learning and systems with excess degrees of freedom. Technical Report COINS Technical Report 88-27, Massachusetts Institute of Technology, Boston, November 1987.

[6] J. Kolen and J. B. Pollack. Back prop is sensitive to initial conditions. Technical Report 90-JK-BPSIC. The Ohio State University, Columbus, February 1990.

[7] R. Miikkulainen. A neural network model of scritp processing and memory. Technical Report UCLA-AI-90-03, University of California, Los Angeles, March 1990.

[8] R. Miikkulainen and M. Dyer. Forming global representations with extended back propagation. *Proceedings of the IEEE Second Annual Conference on Neural Networks*, 1988. San Diego.

[9] J. B. Pollack. Recursive distributed representations. To appear in *Artificial Intelligence*.

[10] J. B. Pollack. Recursive auto-associative memory: Devising compositional distributed representations. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, pages 33–39, May 1988. Montreal.

[11] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations through error propagation. In D. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 25–40. MIT Press, 1986.

[12] D. Rumelhart and J. McClelland. On learning the past tense of english verbs. In D. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2, pages 216–271. MIT Press, 1986.

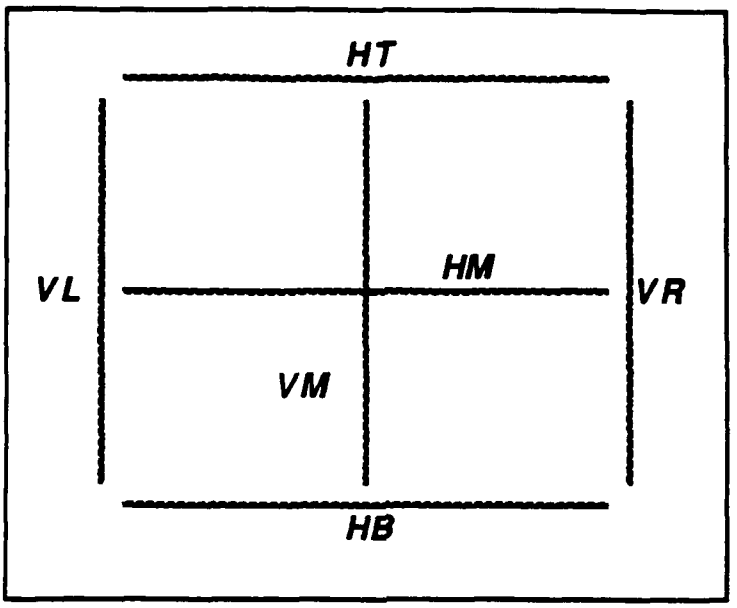


Figure 1 - Strokes used in the formation of characters.

Figure 2 - Separation of Symbols