Technical Report No. 236

A stochastic learning algorithm for neural networks

John W. L. Merrill and Robert F. Port[1]
Departments of Linguistics and Computer Science
Indiana University
Bloomington, Indiana 47405

March 17, 1988

[1]Research partially supported by NSF grants DCR–85–18725 and DCR–85–05635

1 Introduction

This paper describes a stochastic learning algorithm for abstract neural networks. An abstract neural network is an assemblage of computationally trivial elements, nodes, which communicate with one another by means of simple messages along directed paths. It has become evident that these abstractions of neuronal function can simulate a number of cognitive processes [14], particularly in view of the ability of these networks to function as learning models. Unfortunately, many of the methods heretofore presented for learning techniques apply in limited contexts only. Most popular algorithms are modifications of gradient descent, and are, therefore, limited to learning systems with continuously variable parameters. Some popular algorithms require that the desired outputs from the network also be available at each time. Williams [18] calls this kind of feedback "instructive". By contrast, stochastic algorithms can employ "evaluative" feedback consisting of a global evaluation of the performance of the network. Instructive feedback is inappropriate in training networks to perform tasks for which there are many different, but equally acceptable, means to a solution. Among such tasks are the categorization or analysis of temporally distributed signals (such as speech) or the memorization of arbitrary strings of tokens.

Not all deterministic algorithms suffer from all these limitations. For example, Barto and Sutton [2] presented a deterministic algorithm which trains in an evaluative context. Their algorithm applies to on-line learning, in which an individual changes its own structure in a continuous fashion in order to improve its own performance. The work described in this paper concerns off-line learning, in which the performance of a population of individuals is externally modified. In this respect, their algorithm is more appropriate for simulating learning, while ours is more appropriate for simulating evolution.

The search method described in this paper is a fusion of ideas from simulated annealing [9] and genetic algorithms [6,3]. Simulated annealing is a form of hill-climbing in which inferior choices are occasionally made, guaranteeing that the search algorithm never gets "stuck" in a local minimum. The procedure is started by selecting a random element of the domain space to serve as the initial "parent". At each time thereafter, the then-present parent is altered slightly, yielding a new "child". The function to be optimized is computed at both. If the child yields a better value than the parent, then the parent is replaced by the child. If not, then a random choice is made, depending on the comparative performance of the two, and the one chosen is taken as the new parent. Then, the procedure is repeated. In a genetic algorithm, a search space is examined by maintaining a large collection of elements of that space. Small modifications are made to members of that population, which is periodically culled in order to improve the performance of members of the sample. The search technique explored here is derived from both of these systems. It can be viewed either as a form of simulated annealing which uses a large population and multiple operators

to search its domain space more efficiently, or as a Genetic Algorithm which uses a collection of local improvement operators to yield a more efficient system.

This document is divided up into two major sections. In the first section, the structure of the algorithm discussed herein is presented, and many of the technical decisions that were made during its construction are explained. In the second section of the document, results obtained by applying this algorithm to two problems involving the training of recurrent networks are presented and discussed. Finally, the paper concludes with a general discussion of three topics concerning the results discussed in the paper: the field of applicability of the algorithm, its relation to AR-P learning, and several directions of future research.

2 Algorithm Structure

There are a number of technical issues which must be addressed in designing a random search algorithm to solve any task. Among these are the representation of elements of the search space, the selection of the initial population, the construction of effective search operators, and the development of a procedure for selecting the individuals to be retained in the population. A rough schematic of our algorithm is shown in Figure 1.

Notation. Several different time units will be used during the course of the paper. A single clock step of a synchronous network is referred to as a *click*. A single modify-and-evaluate cycle in a genetic algorithm will be referred to as an *iteration*.

2.1 Representing elements of the search space

Genetic search has usually been applied to domains such as function optimization over $\mathbf{R}^n$ and optimization of classifier systems [6,7,3], as well as more unusual situations like the Traveling Salesman Problem [5,15]. In each of these cases, the domain spaces are easily represented as bit-strings. Neural networks are not naturally amenable to representation as strings, but rather to representation of their weights as either a square array or as a list of lists. The weights on the edges in the network are represented as bit strings. The choice between the two representations of the weight array is dictated by efficiency considerations, since for sparse networks, each click would take $O(n^2)$ multiplications if the array representation were employed, but fewer if the other were used. For pedagogical reasons, we will talk about weight arrays throughout the balance of this paper, since that representation makes some fine points of the algorithm clearer.

2.2 Selecting the Initial Population

The initial population for a genetic algorithm is most commonly selected randomly, in order to maximize the area searched initially, and in order to avoid biased selection which might lead to a sub–optimal final solution. We have followed this strategy to the extent possible. Unfortunately, large networks may include several thousand weights. The initial population can be so diverse that the algorithm runs for a prohibitive amount of time before any learning begins. We have found that a reasonable compromise is to bias the initial selection procedure so that only a very small number of the weights in each network are non-zero.

2.3 Selecting Appropriate Search Operators

The choice of search operators is the key to any successful stochastic search paradigm. In standard genetic search, two operators are used to drive the search, one taking a single argument (the "mutation operator") and the other taking two arguments (the "cross-over operator"). These operators are applied repeatedly to members of the population, and the results of these applications then replace their parents. We follow this strategy. In contrast to the situation in standard genetic search, however, the results of applications of these operators are not necessarily included in the population. In order to improve the performance of the algorithm, a form of conditional acceptance is employed, in which superior children always replace their parents, and in which inferior children sometimes replace them. If the second clause is invoked frequently enough, then the search can be made very close to true hill-climbing without risking capture by local minima. In order to encourage the algorithm to converge, the likelihood of accepting an inferior child is reduced during the course of the search.

We have experimented with four different cross-over operators.

1. An operator taking a fixed block of bits from the representation of each of the weights corresponding to the inputs and outputs of a node and exchanging the blocks.

2. An operator differing from the preceeding only in exchanging all of the weights corresponding to one node between two networks rather than only a portion of each weight. This is shown in Figure 2.

3. An operator which exchanges the corresponding weights on the inputs to several adjacent nodes.

4. An operator combining features of the second and third of these, exchanging all of the weights corresponding to a set of adjacent nodes.

Metaphorically, these operators cut pieces of corresponding nodes from the two networks and paste each one into the other's network.

For any of these operators to work, all networks must be the same size and have the same topology. Each node must take inputs from the same nodes in both networks, although the weights associated with those inputs might be zero. It is also helpful if the "function" of each node exchanged is the same as that of the node for which it is substituted. To encourage this, the algorithm uses a selection procedure to adaptively force the networks to be similarly structured. Following Holland [6], members of the current sample are ranked according to some function of their performance, and a new population is generated from a biased sample of the previous population. Each element is duplicated with frequency exponential in its rank. We have found that making the selection procedure more stringent (that is, more like the 1-best strategy) as the search proceeds is advantageous, since variety in the population is less important when the search is near to the optimal member of the search space.

3 Network structure

All networks discussed in this paper consisted of three types of nodes: input/output nodes, hidden nodes, and constant nodes. To present an input to the network, the program set the values of the input nodes to the desired input. The values thus installed were destroyed by the computations that the input nodes performed in the next click. Hidden nodes were only connected to input nodes and to other hidden nodes; their presence allowed the network to behave as if it had a memory. Each of the networks also contained a single node with constant output 1. Including a node like this among the inputs to a node allows it to behave as if it included a bias without actually distinguishing that bias coefficient from the coefficients on the inputs on the variable nodes. (See [13] for a more complete discussion.)

All non-constant nodes in the networks were linear threshold nodes. At each time step, each one computes a weighted sum of its inputs, then maps that sum through the squashing function:

$$squash(x) = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases}$$

This value is the next output of the node.

We now proceed to detailed descriptions of two of the tasks to which this algorithm has been applied.

4 Short-term memory task: Experiment 1

Although algorithms like Hebbian learning or back–propagation are suitable for simulating some kinds of conditioned or learned behavior, they are inadequate for investigating or modeling cognitive

processes. One such process is short-term, associative memory. In this task, we trained networks to become such memories.

A hetero-associative memory is a device which accepts halves of pairs as input and reconstructs the complete pair as output. If the device were presented with a list of pairs which included ⟨horse, five⟩ and then probed with either ⟨horse, −⟩ or ⟨−, five⟩, it should recover the original pair, ⟨horse, five⟩. Such systems have been investigated in a computational framework as the basis of context-addressable memories [8]. Our device more closely approximates a human model of memory, in that it is allowed only a short rehearsal period.

4.1 Methods

Networks were constructed as described above, with 16 input–output nodes, 8 hidden nodes, and one bias node. A population of 5 networks was maintained at all times. Two members of the population were subjected to mutation on every iteration. In addition, one pair of members of the population underwent crossover on each iteration. The population was re-selected every 10 iterations. The 16 8-bit strings that balance $+1$ and -1 served as stimulus tokens. Ten lists were made of pairs of these strings which served as targets and probes. Each list contained either 3 or 4 token pairs; list length was randomly chosen.

An experimental run consisted of two parts, a presentation phase and a testing phase, as shown in Figure 3. During presentation, the activations of the input/output nodes were set to a stimulus pair for one click, and then the outputs of all of the nodes in the network were allowed to vary according to the rules of the network. Since the input–output nodes took inputs from elsewhere in the net, their values were destroyed immediately. The net was allowed to run for two clicks, and then another pair was presented. On completion of each list of pairs, the network was probed with a sequence of bit-strings, each of which was composed of a valid token concatenated to a string of 8 zeroes. On half of the test trials, the probe token was in the memory list. In this case, the network was required to reconstruct the original token pair two clicks later. The other half of the time, the network was presented with a valid token not drawn from the memory list. In this case, the network was required to complete the string with eight -1's. Each trial was scored by counting the number of bits correct out of 16 two clicks after presentation. The score received by the network was the sum of scores across all trials.

Each training session consisted of many experimental runs, so that any given bit-string would be associated with many different partners during the course of each test. To test generalization, the final network was presented with a list of novel lists (constructed from the same token list) after the termination of the entire training process.

4.2 Results

Performance. The best of the networks trained as described above achieved an average score of 72.8% correct bits when tested on the training data. While this is not ecellent, it must be kept in mind the the task is difficult. In human recall, there is redundant semantic information available to the subject; the tokens in these lists had no such redundancy. The performance of the best network in the population increased uniformly from an initial score of roughly 40%. When tested on novel lists, the best network correctly recalled 61.7% of all bits in tokens of the memory set, and correctly inferred 74.5% of all bits in the tokens which it was to reject as new.

Network bias. Since half of all test tokens had a string of -1's as the correct response, the network would be expected to have a bias to always respond with -1. We computed the number of errors which occurred when a $+1$ was substituted for a -1, and vice versa. The network incorrectly classified 21.5% of all -1's on the memory set, and incorrectly classified 22.5% of the -1's from distractors. The network incorrectly classified 53.9% of all $+1$'s on trained tokens and 35.9% of all $+1$'s on distractors. As a consequence, there is a significant difference in the mean number of 1's in the prsented tokens and in the unpresented tokens.

4.3 Discussion

Performance of the network. The results indicate that the network constructed in this task can be used as a recognition model, simply by drawing a threshold on the number of features in the output string which are set to $+1$, since roughly 65% of all bits will be set to -1 on recognized tokens, and roughly 80% of all bits will be set to -1 on distractors. We conjecture that the network's performance might be improved by making the behavior of its constituent nodes more flexible. (For instance, by making the node response functions continuous instead of discrete.)

Usefulness as a recognition system. The recognition portion of this model uses an unusual method for deciding whether a token is "new" or not. Where some memory models "examine" their contents, and compute some formal familiarity function which is based upon an ensemble average similarity, this model considers its contents and tries to recall the completion of the token. If the token fails to complete, then the network concludes, by default, that it is not in the memory set. It would be difficult to extend this to a full recall model, since the tokens which are recalled tend to be distorted. Possibly, employing a sufficiently redundant feature set could allow a lexicon to restore an original token from its distorted recalled form.

5 Sequence prediction task: Experiments 2–4

The networks for the associative memory task were recurrent, but they were only required to recall a single "fact". Another function for a recurrent network could be the analysis of patterns that develop over time. In order to investigate the ability of the algorithm to train networks to perform tasks like this, we trained networks to recognize and predict circular shift registers, a generalization of a task from Rumelhart, Hinton, and Williams [13]. They installed a bit string in a register, and the network was run for a fixed period. The output was expected to be the original input shifted two positions to the left. This system was restricted to learning the behavior of one shift register at a time.

In this task, networks were trained to recognize and predict the behavior of a restricted class of circular shift registers. Each network consisted of six nodes that served as both input and output nodes and a number of hidden nodes that were neither inputs nor outputs. The output from each node was connected to the inputs of every other node. In addition, each network had a constant bias source with output 1. Each shift register contained six bits, one block of them on, and one block off, with 'on' represented by $+1$, and 'off' by -1. The block of -1's could be empty. The register was shifted a constant number of cells each click. Thus, there were thirty-six possible registers—six possible initial arrangements and six possible rotations, including the null rotation. (See Figure 4.) This task is analogous to presenting periodic events with unknown period and asking the system to analyze the periodic behavior.

5.1 Methods

During training, each network was evaluated for its performance on each register in turn. To evaluate the performance of a network on any particular register, the initial state of that register was installed on the input–output nodes of the network (again, the values were installed, but not latched), information was allowed to propagate through the network for three clicks, the network was presented with the contents of the same register after one shift, and information was allowed to propagate through the network. After each such sequence of operations, the program computed the dot product of the network outputs and the desired outputs, yielding a number between -6 and $+6$. This number was returned as the score of the net on the trial. The network's overall score was the sum of the scores on the trials.

Experimental procedures. Each of the genetic searches described below was run with samples of 15 elements of the domain of the search, which will be referred to as a population of individuals. Every iteration, the mutation operator was applied to 6 individuals; every second iteration, the cross-over operator was applied to 3 pairs of individuals. The population was reselected ev-

ery 10 generations. Within this framework, a number of experiments were performed and some experimental parameters were varied. We report here on the results of several of these experiments.

5.2 Results

Since the networks employed in this series of experiments were small and composed of nodes with limited function, maximum attainable performance appears to be about 88% correct bits. The expected score for a random network is 50%, although a majority rule strategy (that is, reporting whichever of -1 or $+1$ is more frequent) yields a score of 75%.

Experiment 2: Importance of conditional acceptance. In order to clarify the importance of conditional acceptance to the performance of the algorithm, the algorithm was run four different times in each of four different conditions shown in Figure 5. The conditional acceptance operator (1) acted on the results of both operators (solid line), (2) acted only on the results of the mutation operator (dashed line), (3) acted only on the cross-over operator, and (4) acted on neither operator. It can be seen that the networks generated with conditional acceptance perform better than those generated without.

Experiment 3: Importance of including cross-over. To evaluate the importance of cross-over when combined with simulated annealing, the algorithm was run four times with cross-over and four times without cross-over. The results are shown in Figure 6. After about 2000 generations, the performance of the algorithm with cross-over (dashed line) has reached asymptotic performance, but the performance of the algorithm without cross-over (solid line) has not yet peaked. It will eventually perform as well as the other version, but only after about 6000 generations. We also noted that, after training, the optimal network generated without crossover had 41% of the edges in the net active. The optimal network generated with crossover had only 35% of its edges active.

Experiment 4: Differences in the cross-over operator. A number of different cross-over operators can be included in the algorithm. We have only investigated operators which tend to preserve the structure of the networks upon which they operate. In particular, the operators studied here preserved functional subunits of the networks to which they were applied, whether those units were pieces of the weights on all of the connections on a node, or the inputs to a node, or the weights corresponding to the links on several nodes.

We studied the behavior of the four operators discussed in 2.3, above. These results are shown in Figure 7. The performance curve obtained with the least destructive cross-over, (1), is shown in solid line in panel A of Figure 7, and is repeated, for reference, as a dotted line in the other panels.

The performance curve with cross-over (2) is shown in panel B, with cross-over (3) in panel C, and with cross-over (4) in panel D. The least destructive cross-over, (1), was consistently better.

5.3 Discussion

Comparison with past work. Rumelhart, Hinton, and Williams [13] discuss a task which is similar to the one discussed here. They observe that back-propagation can be used to train recurrent networks if those networks are "unwound" in advance. A network is unwound by replacing all recurrent connections with layers of feedforward connections. Each layer corresponds to one time step of the input. This technique requires that the network have depth at least as great as the length of any sequence of input tokens or output behaviors. Thus, the shift register in [13], when unwound, has three levels, each of which is an exact copy of the original network.

Echoic Memory. Our original motivation for this task was to explore simple perceptual-like tasks that seem to require an echo-like memory. Echoic memory is a portion of human auditory memory in which a raw, nearly unanalyzed, version of the acoustic signal persists for about a very short time. The sequence prediction task requires a similar device to code the snapshots already seen. These results suggest that random search algorithms can be used to discover such networks.

6 General discussion

Both the associative recall and sequence prediction tasks have close parallels to biologically relevant tasks. Apparently, this algorithm can be used to train networks to perform such tasks. A number of issues arise in considering such a research enterprise.

Technical aspects of the algorithm. As one can see from examination of figure 5, inclusion of conditional acceptance dramatically improves performance of the search algorithm. Performance can be improved still further by including a cross-over operator in the algorithm in addition to employing conditional acceptance to the mutation operator (figure 6), provided that the cross over operator is not too destructive (figure 7).

Population size. We have found that including a local improvement operator (mutation with conditional acceptance) in a genetic search algorithm reduces the size of the population required to guarantee good performance (cf. [15]). Instead of employing a population of size about 50, as is standard, we employed populations of size 5 and 15. In fact, in other experiments, we have obtained satisfactory results using far smaller populations, provided local improvement was used.

We find, however, that if the population is made too small, then performance degrades to roughly that of pure simulated annealing.

Appropriateness of stochastic search. Stochastic algorithms are unique among learning algorithms in applicability to training networks in environments that satisfy three criteria. First, stochastic algorithms apply when evaluative, instead of instructive, feedback is available. Instructive feedback consists of telling the network how it should behave. For each stimulus, the algorithm must be provided with the exact desired response (as in back-propagation). In contrast, evaluative feedback consists of telling the network how well it is doing on average, typically after many trials. Second, stochastic algorithms apply even if environmental feedback can be delayed by a variable amount. Third, stochastic algorithms apply when the domain of the search is not topologically connected. Gradient descent is restricted to search on manifolds, and is, therefore, inappropriate for combinatorial searchs, such as network architecture search.

Many interesting tasks require evaluative, rather than instructive, feedback. Consider training a network to recognize speech or to categorize any other temporally distributed signal (for instance, a melody). The only times at which the behavior of the network can be confidently prescribed are at the beginning of the process, when the network should be in a neutral state, and at the end of the process, when the network should have classified the signal as a word or song. At all other times, all that can be safely demanded of the network is that it must not mis-classify signals. Crafting a teacher function to enforce these restrictions requires the arbitrary selection of target outputs at the intermediate times [17,1]. In some sense, this is parallel to prescribing the outputs of hidden nodes in a network. Such a prescription can frequently be constructed, but there is no reason to believe that the solution so obtained will be anything like the "natural" solution.

In contrast, consider the exclusive-or problem [13]: A feed-forward network is presented with two Boolean inputs and is trained to produce a Boolean output which corresponds to the exclusive-or of the two inputs. Given a pair of inputs and an output, it is easy to compute not only how well the network performs, but also the amount to change the output. This is a task which is naturally implemented in an instructive context. Again, consider the circular-shift-register task from the same paper. Here, a recurrent network is trained to behave like a fixed circular shift register using a modified form of back-propagation in which the network is "unwound". That form of back-propagation is limited to cases where the feedback is instructive and presented after a fixed period of time.

Comparison to associative reward-punishment learning. Stochastic algorithms are not unique in applying in evaluative environments. One deterministic algorithm which functions in an evaluative environment is Barto and Sutton's Associative Reward-Punishment algorithm [2],

which performs an estimated gradient search to maximize reward and minimize punishment, thus simulating instrumental conditioning. Each node in an AR-P network maintains running averages of the values of its outputs, its weighted inputs and their correlations. When reward is given, these averages are used to guide weight modification. AR-P requires only evaluative feedback, and hence is more general than those algorithms which require instructive feedback. In addition, the AR-P algorithm is not limited to feed-forward networks, since there is no notion of propagating error back to parent nodes. Instead of requiring that the error due to each node be computed at each learning step, the learning algorithm contents itself with following a reward gradient in the mean.

Most importantly, AR-P is tailored to situations where feedback is erratic or crude, since the nodes in the network maintain running averages of their history instead of attempting to update their state instantaneously. As a consequence, AR-P networks could (in principle) be applied to training networks to recognize speech without choosing values of the outputs from time-slices at inappropriate times; instead, the network could be presented with a simple right-wrong judgement.

This algorithm has one drawback: It is confined to searching connected spaces, like all derivatives of gradient descent. Some authors (for instance, McClelland and Rumelhart, in the introduction to their book [14]) have argued that this is not an important limitation. Others [4,11], have produced evidence that the type of learning a network can achieve is affected by its initial edge structure as well as the behavior of its constituent nodes. While AR-P is appropriate for simulating on-line learning in an individual, it is not appropriate for simulating the development of innate structures that make "learning" certain things natural. Our system in experiments 3–6 could not help learning the shift registers which were presented to it.

Future directions The algorithm described in this paper is limited to long-term adaptation only. Although it generalizes easily to short-term learning predicated upon long-term adaptation (such as would characterize learning to speak a language), the edge search method we have employed in this report is too inefficient to be practical in such searches. Further, while this algorithm has yielded encouraging results when applied to the training of small networks, its performance when applied to larger networks is disappointing. If the algorithm employed an edge representation which encoded structure in a scale-invariant way, as in Mjolsness et. al. [12], then that limitation could be obviated. We are currently investigating the development of representations of edge structure which encode a variety of different configurations in a scale-independent, biologically plausible fashion.

7 Conclusions

We have discussed a stochastic learning algorithm for deterministic abstract neural networks, and discussed its applicability to tasks which are not easily solved by more traditional means, demon-

strating this by using to solve two such problems. In solving these problems, we have shown that networks capable of short-term processing can be constructed using only a coarse evaluation of long-term performance of the system. The significance of this result for cognitive science in general is that by approaching the problem of temporally distributed information in this way, we need only evaluate the networks for their ability to perform certain tasks—in this case, predicting a temporal pattern or associating arbitrary items—rather than actually show them what they should be doing at each point in time, as required by teacher-based feedback.

Stochastic algorithms permit learning in disconnected spaces, such as the space of possible network architectures. This kind of learning is important in evolutionary searches, such as would be necessary to enstantiate innate knowledge in network topology. The ability of nervous systems to self-organize so that they detect patterns distributed in time has proven difficult to simulate Stochastic learning techniques applied to network architectures may be useful in constructing such systems.

References

[1] Anderson, S. A., Merrill, J. W. L., and R. F. Port. *Applying sequential networks to speech recognition*, (in preparation).

[2] Barto, A. and Sutton, R.. *Simulation of anticipatory responses in classical conditioning by a neuron-like adaptive element*, **Behavioral Brain Research** 4 (1982), pp. 221-235.

[3] De Jong, K. *Genetic Algorithms: a 10-year Perspective*, in J. Greffenstette (ed.) **Proc. of an Int'l Conf. on Genetic Algorithms and Their Applications**, Carnegie–Mellon University, Pittsburg, 1985, pp. 169-177.

[4] Dolan, C., and M. G. Dyer. **Symbolic Schemata in Connectionist Memories: Role Binding and the Evolution of Structure**, UCLA AI Lab Tech. Rept. UCLA-AI-87-11.

[5] Greffenstette, J., R. Gopal, B. Rosmaita and D. Van Gucht. *Genetic algorithms for the Traveling salesman problem*, in J. Grefenstette (ed.), **Proc. of an Int'l Conf. on Genetic Algorithms and Their Applications**, Carnegie–Mellon University, Pittsburg, 1985, pp. 112-120.

[6] Holland, J. **Adaptation in Natural and Artificial Systems**, University of Michigan Press, Ann Arbor, 1975.

[7] Holland, J., K. Holyoak, R. Nisbett and P. Thagard. **Induction: Processes of Inference, Learning and Discovery**, MIT Press, Cambridge, 1986.

[8] Hopfield, J. *Neural networks and physical systems with emergent collective computational abilities*, **Proc. Nat. Acad. Sci., U.S.A. 79** (1982).

[9] Kirkpatrick, S., C. D. Gelatt and M. P. Vecchi. *Optimization by simulated annealing*, **Science,220**(1985), pp. 671-680.

[10] Lehar, S. and Weaver, J. *A developmental approach to neural network design*, in Caudill and Butler, eds., **Proceedings of the First International Conference on Neural Networks** (1987), SOS printing:San Diego, pp. II 97-104.

[11] Merrill, John W. L. and R. F. Port. *Fractal Neural Networks*, (in preparation).

[12] Mjolsness, E., Sharp, D. H., and B. K. Alpert. *Scaling, Machine Learning, and Genetic Neural Networks*, Los Alamos National Laboratory Tech. Rept. LA-UR-88-142.

[13] Rumelhart, D., G. Hinton, and R. Williams. *Learning internal representations by error propagation*. In Rumelhart and McClelland, eds., **Parallel Distributed Processing: Studies in the Microstructure of Cognition**, MIT Press, Cambridge, 1986, pp. 318-361.

[14] Rumelhart, D. and J. McClelland. **Parallel Distributed Processing: Studies in the Microstructure of Cognition**, MIT Press, Cambridge, 1986.

[15] Suh, Jung Y., Dirk Van Gucht. *Incorporating Heuristic Information into Genetic Search*, Indiana University Department of Computer Science Technical Report, Indiana University, February 1987.

[16] Szu, H. *Fast simulated annealing*, in Denker, J. S. (ed.) **Neural Networks for Computing**, AIP Conference Procedings, Vol. 151, American Institute for Physics, New York, 1986.

[17] Watrous, R. L. and L. Shastri. *Learning Phonetic Features Using Connectionist Networks: An Experiment in Speech Recognition*, in Caudill and Butler, eds., **Proceedings of the First International Conference on Neural Networks** (1987), SOS printing:San Diego, pp. IV 381-388.

[18] Williams, R. J.. *A Class of Gradient-Estimating Algorithms for Reinforcement Learning in Neural Networks* in Caudill and Butler, eds., **Proceedings of the First International Conference on Neural Networks** (1987), SOS printing:San Diego, pp. II 601-608.

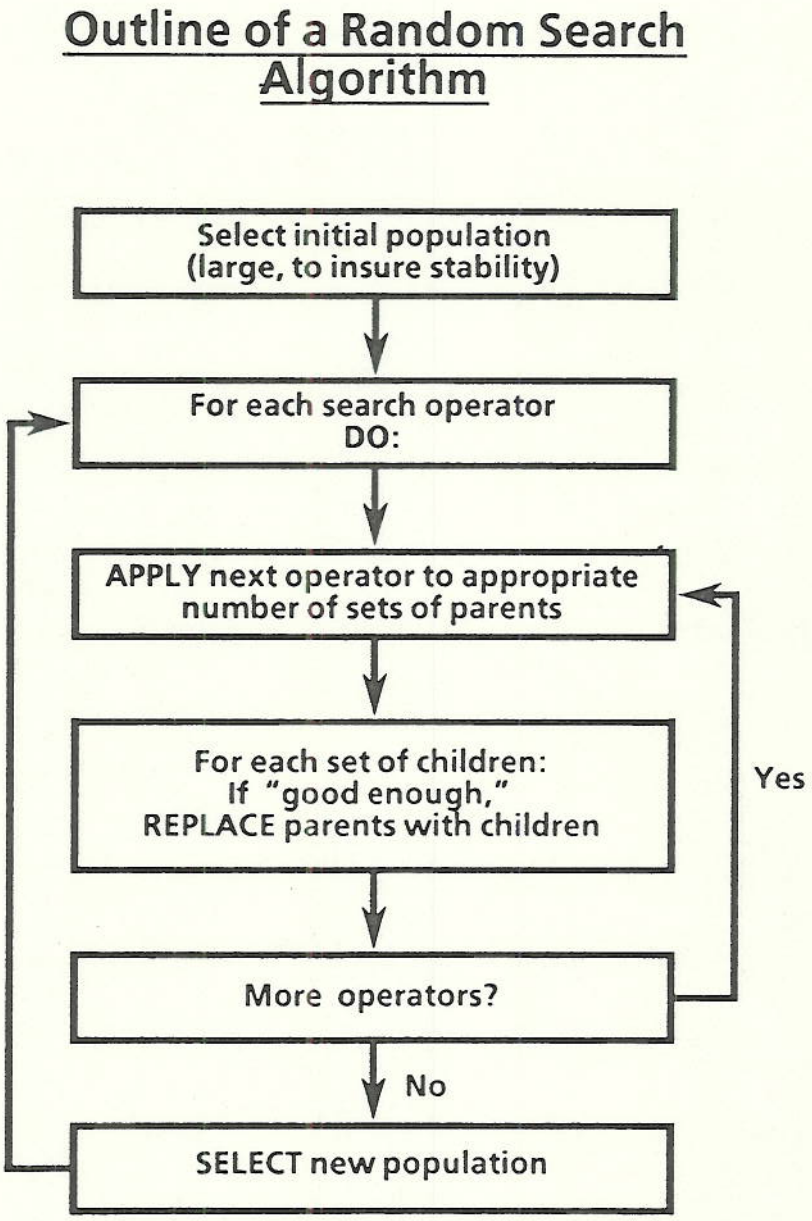

Figure 1. The generalized outline of a random search algorithm such as simulated annealing and genetic search. After selection of an initial population of individuals, the first search operator is applied to some individuals and, if the children pass the conditional acceptance test, they replace their parents. Then another operator is selected and a similar modify-and-test cycle is completed. Finally, the entire population is selected to favor the better performers and the cycle of search operators is looped again.

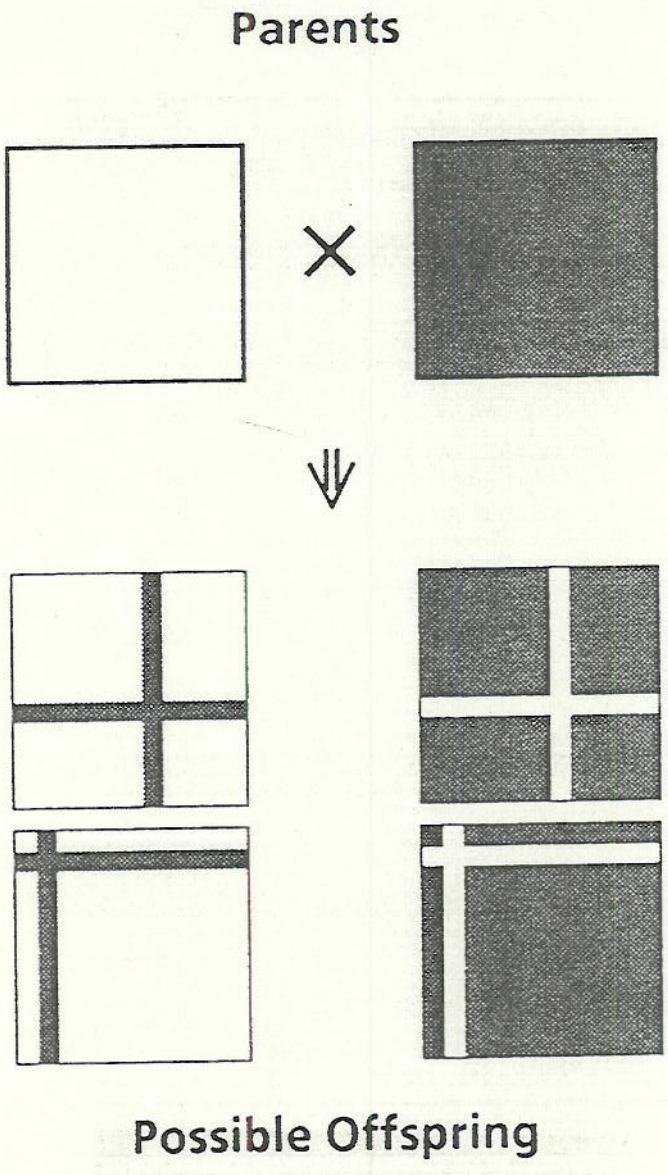

Figure 2. One of the crossover operators employed in our experiments. In this representation, each parent appears as a square matrix of input weights (in columns) and output weights (in rows). Thus, a cross-hair figure represents all the nodes that feed directly to the center of the crosshair and all the nodes that it sends information to directly. The offspring, then, mostly resemble one parent or the other but with these two weight segments switched between the parents.

Associative Memory Task
Experimental Run

Presentation Phase: 8-bit patterns of –1 and +1, displayed as 0 and 1 for visibility)

		Stimulus Pair A	Stimulus Pair B	Network Response A	Network Response B
t1	pair1	11001100	00110011		
				(2 clicks)	
t4	pair2	11000011	00001111		
				(2 clicks)	
t7	pair3	00111100	0101010		
				(2 clicks)	

Test Phase: Probe with item from memory set or not from memory set.

	Stimulus A	Stimulus B	Network Response A	Network Response B
Memory Set Probe				
t10	????????	010101		
			(2 clicks)	
t13			00111100	010101
Non-Memory Set Probe				
t10	00001111	????????		
			(2 clicks)	
t13			00001111	00000000

Figure 3. A schematic of a single experimental run of the network described in Task 1. First, a collection of token pairs are presented to the network, a single token is presented. Two clicks later, the correct pair should be produced if a probe in the memory set was presented. If a distractor probe was presented, the network should signify that by completing the probe with a string of -1's. In this display, {0, 1, ?} replace {−1, +1, 0} for visibility.

Sample Test Run on Individual Net

Time	Stimulus Pattern	Network Input	Network Output	Score
t_0	●●○○○○	●●○○○○		
		(3 network cycles)		
t_1	○○●●○○	○○●●○○		
		(3 network cycles)		
t_2	●○○○○●		●○○○○○	4 (83%)

Figure 4. A sample test run on a individual net. The second column from the left shows the stimulus pattern with time going from top to bottom. Although the actual patterns were $+1$'s and -1's, we employ • and ○ here for clarity. A single time frame of the shift register is presented and the net has three cycles to propagate the information. Then the second sample is presented and the net has three more cycles to operate. On the next cycle, we expect to see the system 'predict' the pattern on its input/output nodes. In this case, one bit is incorrect, so the dot-product for vectors of -1 and $+1$ is 4 (83% correct.) Evaluation of this network is done after accumulating the dot products of many such trials.

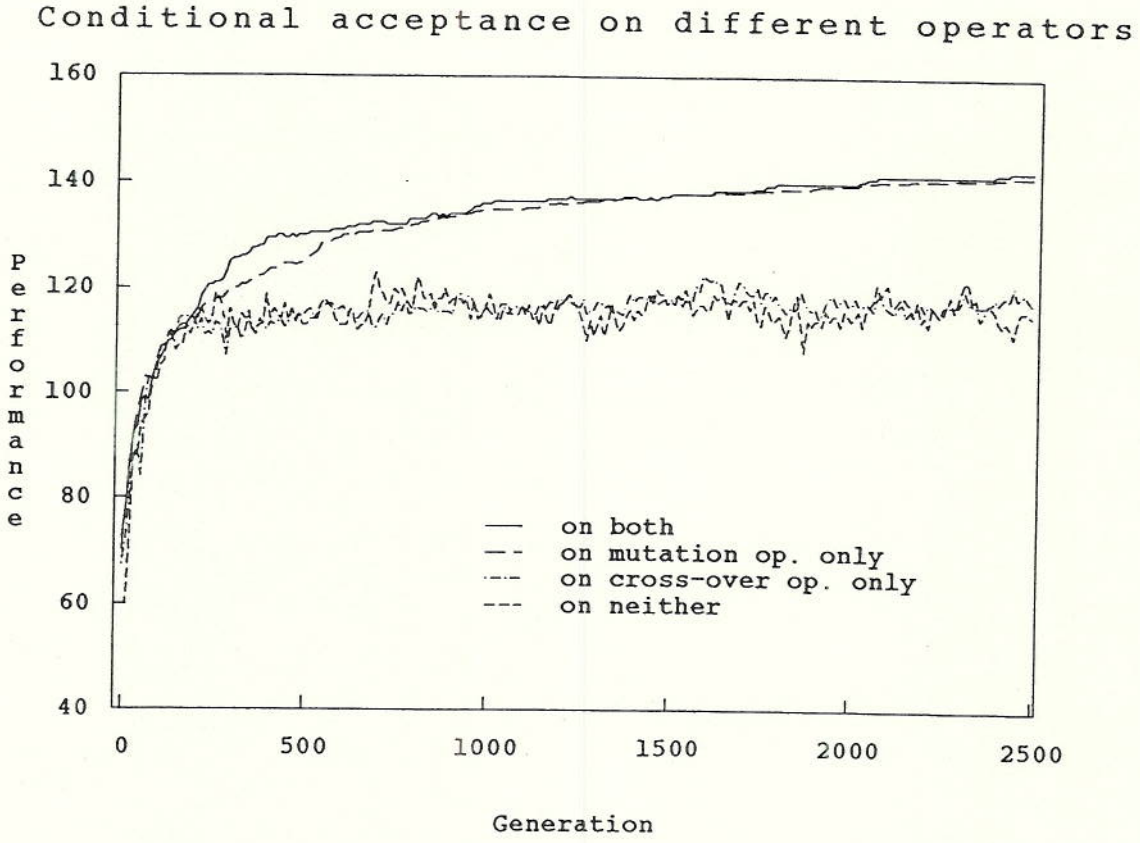

Figure 5. Results of Experiment 2. The effects of applying a conditional acceptance procedure after different combinations of operators in the search. Four different conditions are displayed: conditional acceptance applied to both operators, applied only to the mutation operator, applied after only the cross-over operator, and never applied. Performance is measured in bits correct, as in Figures 6 and 7.

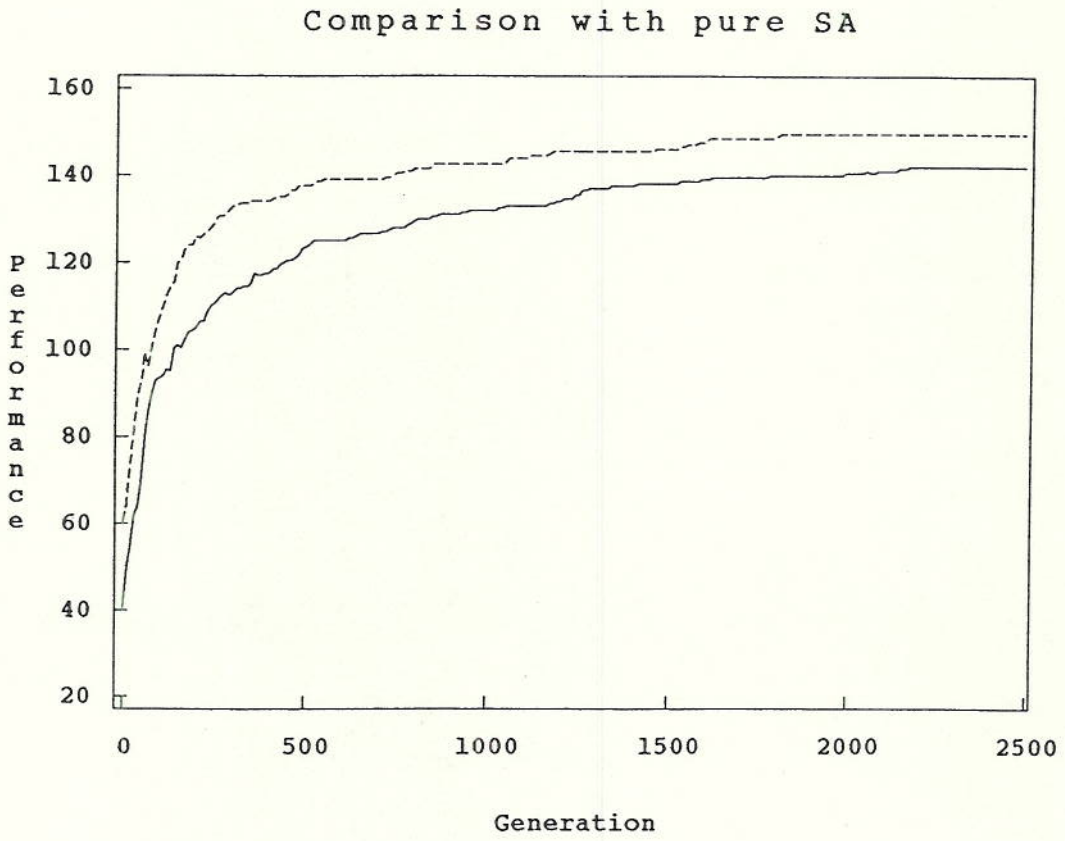

Figure 6. Results of Experiment 3. This shows performance of a network using genetic search to solve our task with and without cross-over, demonstrating the superiority of the algorithm when cross-over is included. The dashed line shows average performance including cross-over, and the solid line shows performance without cross-over.

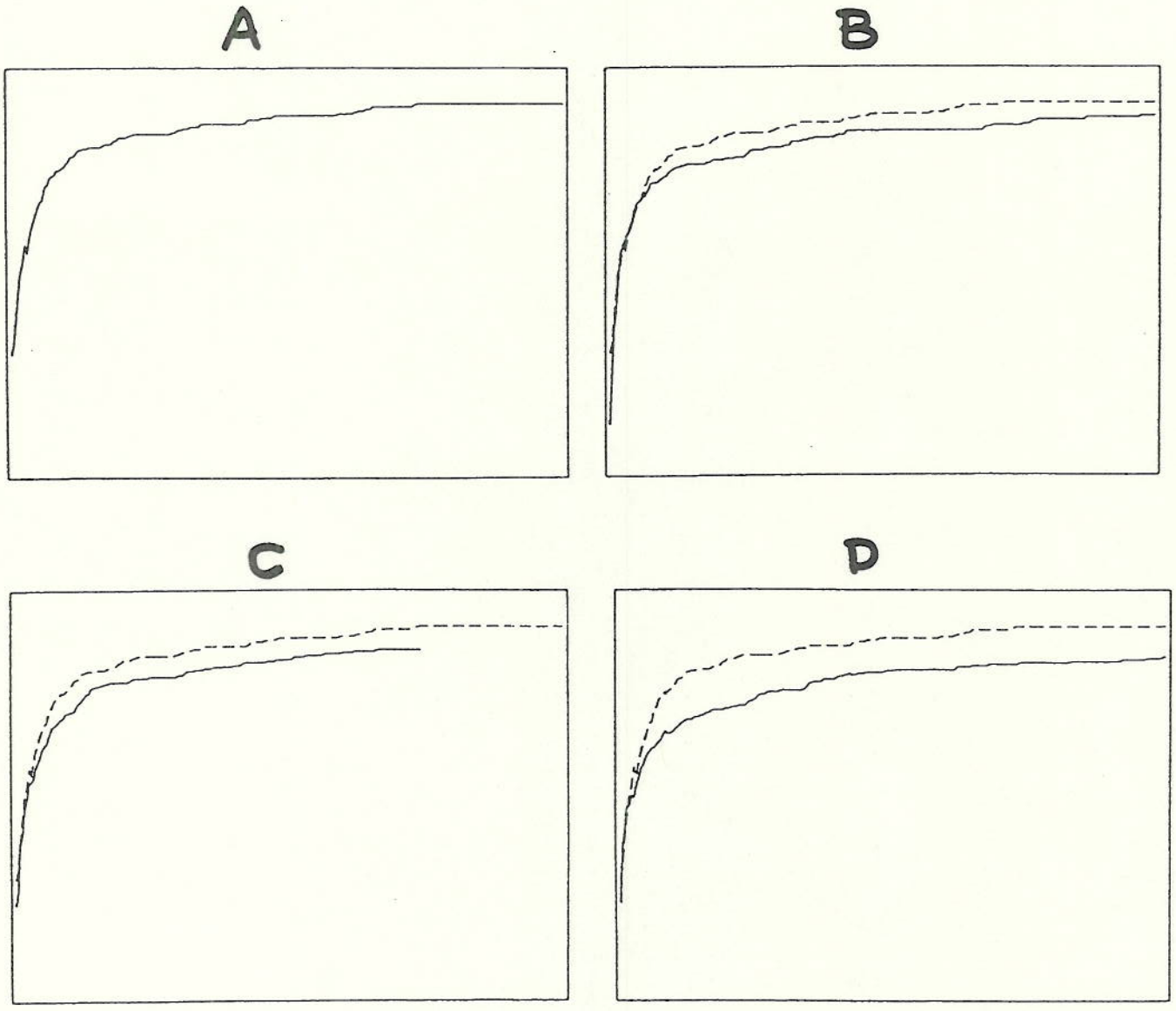

Figure 7. Results of Experiment 4. These data show the performance of four difference versions of the cross-over operator. The best performer, shown in panel A is repeated for reference in the other panels. The best performing crossover was the one that only changed a single bit of information in the input and output weight segments for a randomly chosen node. Each of the scales in the figures is the same as the scale in Figure 6.