Temporary Memory for Examples Can Speed Learning in a Simple Adaptive System

Lawrence Davis
Tica Associates
36 Hampshire Street
Cambridge, MA 02139 USA
(617) 864 2292
FAX (617) 494 4850
Email: 70461.1552@
compuserve.com

Stewart Wilson
The Rowland Institute For Science
100 Cambridge Parkway
Cambridge, MA 02142 USA
(617) 497 4650
FAX (617) 497 4627
Email: wilson@smith. rowland.org

David Orvosh
Tica Associates
36 Hampshire Street
Cambridge, MA 02139 USA
(617) 864 2292
FAX (617) 494 4850
Email: 70461.1552@
compuserve.com

Abstract

A temporary memory for recently experienced examples was coupled with BOOLE, a previously developed one-step classifier system that learns incrementally. Regimes were found in which the performance of the combination was superior to that of BOOLE alone, whether the task was to maximize average performance with respect to number of external trials, or the sum of external plus internal trials. The results suggest that storing and periodically reviewing a limited number of recent examples may enhance learning rates in otherwise strictly incremental adaptive systems.

1. Introduction

Many artificial autonomous learning systems, whether based on networks or classifier systems, are *incremental*--that is, they may learn something from each trial, example, or experience when it occurs, but they do not store raw experience for possible later study and processing. On the other hand, a number of important techniques from the AI domain of concept learning [9-10] collect and store large numbers of examples before processing them as a batch to derive the concepts. These latter techniques are distinctly *non*-incremental and differ from those of the incremental systems, but they can often learn faster in terms of the total number of examples that must be processed.

Despite this, incremental techniques are usually regarded as more realistic for autonomous adaptive systems, which may have limited leisure for off-line processing and whose environments are often noisy and non-stationary. We wondered, however, whether there might be ways in which small amounts of temporary memory for recent examples could be beneficial to incremental systems. Higher animals appear to have temporary memory, and people sometimes review and rehearse recent situations which had, e.g., unexpected outcomes. In this paper we present some very simple experiments in which temporary memory added to an existing incremental learning system gave performance superior to that of the incremental system alone. In our view, these results may be related to the psychological phenomenon of rehearsal [4].

In the next section we describe the *multiplexer* problem--a Boolean concept learning domain in which the experiments were carried out. In Section 3 we describe the incremental learning system BOOLE to which we attached the temporary example memory. In Section 4 we describe our first procedure for using memory with BOOLE, and show that this procedure greatly improves BOOLE's performance when one measures only the number of presentations of examples from an external source. In Section 5 we describe a second procedure and show that this improves BOOLE's performance versus the sum of the number of external and internal presentations. Finally in Section 6 we present our conclusions.

2. The Multiplexer Problem

Our problem domain was one in which randomly generated bit strings of length L were presented to the system and it had to learn to reply with the correct value, 1 or 0, of a Boolean function F of the bits of the string. The function F used was the so-called Boolean multiplexer in which a subset of the bits address one of the remaining bits, the value of which gives the value of F for that string. For example, in the 6-bit multiplexer, F(011011) = 0. The first two bits are the addressing bits. They form the number "1" in binary, thus addressing the second of the remaining four "data" bits, i.e., data bit 1, whose value is 0. Similarly, F(110001) = 1, where the last bit, data bit 3, is the one addressed by the first two bits. A larger problem, the 11-multiplexer, can be constructed the same way, except that in it the first three bits address the remaining eight bits of an 11-bit string.

The multiplexers form a family of relatively intricate Boolean functions that have served as task domains and have permitted comparisons in studies of several learning systems, including connectionist networks, classifier systems, and decision trees [1-3,10-11]. We chose the multiplexers here especially because they formed the task domain for earlier research with the incremental classifier system BOOLE to which, in this work, we coupled temporary memory. The multiplexer problem's high level of difficulty makes it a good stand-in for more realistic problems that may also require the use of memory.

3. BOOLE

The program BOOLE is a specialization of the general classifier system model [8] to one-step, or "stimulus-response", problems in which the system simply sees an environmental input, produces an output, and collects a reinforcement or payoff from the environment. The basic system is presented in [11], to which the reader is referred for details. The system was extended in [3], which showed that its performance could be significantly improved by large increases in the intensity of reinforcement; because of this new reinforcement scheme, the modified system was called NEWBOOLE. For the present paper, however, we used BOOLE in its original guise, taking, for the 6-multiplexer experiments, exactly the parameter settings of Figure 1 of [11], and for the 11-multiplexer experiments, the parameter settings of the dotted curve of Figure 4 of that paper.

Very briefly, BOOLE contains an initially random population (a set) of condition-action rules called classifiers, in which the condition part of the classifier (a string of length L formed from 1, 0, and the "don't care" symbol #) may match an input string, and the action says in what way (1 or 0) that classifier would have the system respond. On a particular trial, the input string will be matched by a subset [M] of the population: in general, some matchers advocate the response 1, the rest advocate 0. The system's decision occurs as follows.

Associated with each classifier is a scalar value called *strength* which is adjusted by the system's reinforcement algorithm and estimates that classifier's worth to the system. Given, as above, a particular input string and the resulting set [M] of matching classifiers, the system decides its output by selecting a single classifier from [M] using a probability distribution over the strengths of the classifiers in that set. (That is, the probability of being selected equals a classifier's strength divided by the sum of the strengths of all classifiers in [M]--a sort of "roulette wheel" but with wheel sectors sized according to strength.) The system output is then simply the value of the action (1 or 0) of the selected classifier.

For the experiments in this paper, BOOLE was reinforced according to the following regime. Let the *action set* [A] consist of the classifiers in [M] that advocated the selected output. Then, if the output was *correct*, the strengths of classifiers in [A] were incremented by the quantity 1000 divided by the number of members of [A]. If the output was *wrong*, the strengths of [A] were reduced by 80 percent of their current values. In addition, in both cases, and prior to the above adjustments, the strengths of [A] were reduced by 10 percent (see [11] for further details). The effect of reinforcement is to increase the likelihood that previously correct classifiers control future decisions.

Finally, BOOLE employs a periodic *genetic algorithm* (GA) step (see [6] and [7] for introductions to genetic algorithms) in which *new* classifiers are produced by selecting a pair of high-strength classifiers, copying them to form offspring, and then recombining or mutating the offspring strings before inserting them into the population (two low-strength classifiers are deleted).

1. Run BOOLE as usual on an external example.

2. Add this ordered triple to the head of the *memory-examples* list: <100 stimulus response>, where 100 is the example's initial weight, stimulus is the example's bit string, and response is the correct response to that stimulus.

3. Sort *memory-examples* by decreasing order of the weight of its members.

4. If the length of *memory-examples* is greater than *memory-size*, remove the last member of *memory-examples*.

5. If the length of *memory-examples* is less than *memory-size*, go to step 1. Otherwise, execute step 6 and then go to step 1.

6. Memory Cycle: Perform a-d *number-of-examples-to-run* times:
   a. Select an example randomly from *memory-examples*, where each example's chance of being selected is proportional to its weight.

   b. Present the example to BOOLE and reward BOOLE using the example's answer as the correct response.

   c. Decrement the weight of the example by 1.

   d. Run BOOLE's genetic algorithm step.

Figure 1: Temporary memory regime for reducing the number of external presentations.

The rate at which the GA step occurs is keyed stochastically to the rate of input trials (e.g., one new offspring for every two input trials, on average.) Through the GA, the system searches the space of potential classifiers. In general, BOOLE evolves a population consisting primarily of classifiers whose conditions map straightforwardly to the terms (conjuncts) of the disjunctive normal form of the function being learned [11].

4. Storing Examples in Memory

In the results reported here we implemented the temporary memory as a self-contained adjunct to BOOLE, making no attempt to integrate the two (a point discussed further in [5]). We represented the memory simply as a fixed-length list of examples.

Our intuition in adding memory was that by maintaining a list of examples whose classification is known and by presenting those examples to BOOLE between presentations of external examples, the performance of BOOLE with respect to the number of external presentations could be substantially increased. The components of our memory module were as follows:

memory-examples, a list of triples of the form <weight stimulus response>. This list contains the temporary memory. The three elements associated with each example are: a *weight* measuring the example's prominence in memory; a copy of the *stimulus* from an externally-presented example; and a copy of the correct *response* for that stimulus.

memory-size, a variable determining the maximum length of *memory-examples*. The value of this variable was held constant at 40 in the experiments reported in the next section, and was varied from experiment to experiment in the experiments reported in this section.

number-of-examples-to-run, a variable set to the desired number of examples from memory to present to BOOLE after each presentation of an external example. This value varied from experiment to experiment.

MEMORY SIZE		TRIALS PER MEMORY CYCLE					
		1	4	10	30	50	100
1	external	598	325	272	624	*	
	total	1195	1620	2978	19302		
4	external	661	384	290	582	**	
	total	1318	1902	3150	17910		
10	external	645	388	338	604	1446	
	total	1280	1902	3618	18429	72922	
30	external	672	392	325	314	315	510
	total	1314	1840	3279	8839	14586	48049
50	external	592	417	318	267	235	257
	total	1134	1885	3001	6767	9506	20935
100	external	647	416	342	283	235	222
	total	1194	1679	2763	5761	6998	12389

Table 1: Trials to 90% performance for BOOLE with memory on the 6 multiplexer problem. Results are averaged over 24 runs. Without memory BOOLE requires ~1088 external presentations to achieve a performance level of 90%. * indicates runs that were halted at 3930 external, 197,000 internal presentations with performance level < 80%. ** indicates runs that were halted with performance < 90% at 200,000 total presentations.

Our first algorithm for employing memory in the context of a run of BOOLE on a multiplexer problem uses these variables, and is detailed in figure 1.

The algorithm described in Figure 1 is a simple procedure for storing examples after they have been presented externally, and for running them between external presentations. We used a roulette wheel technique for selecting examples rather than cycling through the examples in order because cycling produced diminished performance. We hypothesize that this occurred because examples that are similar and that are grouped together may cause BOOLE to focus too much on them while deleting classifiers that are good for other classes of examples. A single occurrence of such a cluster of examples may not cause BOOLE to suffer, but repeated presentation of such clusters does. Use of probabilistic example choice reduces this effect. Note that step 6d involves the *probabilistic* running of BOOLE's GA step. A classifier is produced in this step with the same probability that it would be produced after the presentation of an external example.

The size of the memory and the number of examples to run after each external presentation are variables that had interesting effects on the performance of the system. Table 1 shows the number of external trials and number of total trials (external plus internal trials) for runs of BOOLE on the 6-multiplexer problem, with the size of memory held constant in each row and the number of internal presentations held constant in each column. The value in each cell of the table is based on the average of 24 runs, where the run was terminated when the average performance of BOOLE over the last 50 external examples was greater than or equal to 90 percent correct, or when the number of total presentations exceeded our limit. Note that the average number of external trials required by BOOLE to achieve this level of performance without memory was 1088.

There are several interesting features of the data in Table 1. The first is that the row showing results with memory size held constant at 1 displays significant performance enhancements. But this column represents the effect of re-presenting the most recent external example *number-of-examples-to-run* times. In the cell for memory size 1 and 1 trial, we see that BOOLE achieves 90% performance in nearly half the number of trials required without memory. But this is very little memory at all. In particular, it only requires "remembering" an example until the next external example is presented. Repeating the example just given doubles BOOLE's speed. Repeating it four or ten times makes BOOLE three times as fast. Repeating it thirty times is worse than repeating it 10 times.

The results for memory size 1 with 10 trials are comparable to those gained with NEWBOOLE. This fact leads us to hypothesize that the gains derived from NEWBOOLE's strategy of greatly

MEMORY SIZE		TRIALS PER MEMORY CYCLE					
		1	4	10	30	50	100
1	external	4557	3242	3966	*		
	total	9113	16206	43618			
4	external	5143	2958	3986	*		
	total	10282	14772	43808			
10	external	5040	3684	4228	*		
	total	10070	18379	46407			
30	external	4940	3328	3306	5580	*	
	total	9850	16519	36063	172080		
50	external	4203	2619	2680	**	**	
	total	8356	12896	28984			
100	external	4108	2721	2220	1191	1052	**
	total	8116	13204	23416	33930	48661	

TABLE 2: Trials to 90% performance for BOOLE with memory on the 11 multiplexer problem. Results are averaged over 24 runs. Without memory BOOLE requires ~7598 external presentations to achieve a performance level of 90%. * indicates runs that were halted at 30,000 external, 871,000 total presentations with performance level < 77%. ** indicates runs that were halted with performance < 90% at 100,000 total presentations.

increasing negative reinforcement when an example is gotten wrong can also be achieved by repeating the example multiple times, allowing BOOLE to get the example wrong if there is a significant chance of doing so.

Note that, counter to what one might expect, the results do not get better as one goes down each column. We believe that the beneficial effects of selecting mixed examples from a larger memory are offset in the 6-multiplexer data by the fact that runs terminate quite quickly. A run with memory size 50 does not begin to use memory until 50 examples are in memory. This means that BOOLE with a smaller memory begins to use memory sooner, and is therefore likely to terminate more quickly.

The results do not get uniformly better as the number of trials increases in rows with small memory sizes. We hypothesize that the use of a great many trials from a small memory degrades BOOLE's performance because the examples being seen by BOOLE are not representative of the complete distribution of examples, and BOOLE focuses too much on the examples currently in memory when the ratio of memory trials to memory size is large. Once memory is size 50 and up, increasing the number of trials appears to increase performance.

Table 2 shows the algorithm's performance on the 11-multiplexer problem. Runs were executed as described above, except that the window over which we measured average performance was increased from 50 to 100 external presentations. Note that the average number of trials required by BOOLE without memory on this problem was 7598. The observations we have just made apply to these results as well.

These data are not, in general, surprising. We conclude that if one uses the traditional measure of classifier system performance for problems like the multiplexer problem--average performance with respect to external presentations--then one can significantly increase BOOLE's speed in attaining high performance levels by storing examples in memory and by executing cycles of internal example presentation, reinforcement, and genetic algorithm triggering after each presentation of an example from the environment. The use of memory in this way could be quite useful to natural organisms, in situations in which the acquisition of external example data is costly, compared to the use of such data under a simulated regime. Of course, a great deal of internal processing is required in order to produce the greatest improvements in performance. In the next section we discuss techniques for minimizing the processing of examples from internal or external sources.

5. Estimating the Value of Memory Examples

It is not always the case that the processing of examples from memory is inexpensive compared to the processing of examples from the environment. Perhaps there is little time in which to process examples internally. Perhaps it is just as costly to process such an example as it is to acquire an example from the environment. Can memory be of assistance in such cases? In particular, if the cost of executing an example from memory equals the cost of executing an example from the environment, can memory be of assistance to BOOLE?

It is clear from the data presented in Tables 1 and 2 that the algorithm in Figure 1 will degrade BOOLE's performance when memory processing is as expensive as processing examples from the environment, since in every cell of Tables 1 and 2 the total number of examples processed by BOOLE exceeds the number processed by BOOLE without memory. In this section we describe two memory-based algorithms that decrease the total number of trials required by BOOLE. The algorithm described in Figure 1 adjusted the weight of examples based on the number of times they had been presented to BOOLE. The algorithms introduced here adjust the weight of the examples in memory based on the relative strengths of the classifiers matching the example's stimulus. These two algorithms, BALANCE and WRONG, are calculated in the following way. Let r denote the total strength of the correct classifiers matching an example's stimulus, and let w denote the total strength of the incorrect classifiers matching the example's stimulus. Then the weight of an example under the BALANCE regime is:

$1 - \left(\frac{|r-w|}{(r+w)} \right)$, while the weight of an example under the WRONG regime is $\frac{w}{(r+w)}$.

BALANCE, the first algorithm described above, sets the weight of an example in memory to the degree to which the strengths of the correct and incorrect classifiers for that example are equal. The intuition here is that an example is valuable to BOOLE if BOOLE has something to say about it, if the things BOOLE says are about equally strong, and if the things BOOLE says conflict. Running such an example will help settle the conflict for BOOLE, either by causing the correct classifiers to receive additional reinforcement, or by causing the incorrect ones to lose strength.

WRONG, the second algorithm described above, sets the weight of an example in memory to the ratio of the total strength of the classifiers that matched it and were incorrect to the sum of the strengths of all classifiers that matched it. The intuition behind this algorithm is that an example is valuable to BOOLE if BOOLE gets it wrong– the more wrong, the more valuable. Examples matched only by classifiers that classify them incorrectly get the highest weight.

We discovered early on that the use of examples under the BALANCE or WRONG regimes must be sparing at best if one is to improve BOOLE's performance on total presentations, and we modified our algorithm for running memory accordingly. The modified algorithm is given in Figure 3, where the variable *memory-ratio* is used in place of the variable *number-of-examples-to-run*.

The difference between this algorithm and the previous one, aside from the replacement of techniques for computing example weight, lies in the way the algorithm selects memory examples. We initially expected that a roulette wheel approach to example selection would be best. However, while selection based on weight worked well early in a run, late in a run a single example sometimes dominated in memory, and is then presented every time. This seemed to cause BOOLE to concentrate too much on a single example, and degraded performance. The regime described in Figure 3 uses weight to decide which examples to maintain in memory, but not to decide which examples to run.

Tables 3 and 4 show the results of using this algorithm on the 6-multiplexer and the 11-multiplexer, based on variation in the value of the *memory-ratio* parameter. The data show that for all the values of *memory-ratio* less than 1.0 for BALANCE, and for all the values for WRONG, the performance of BOOLE is improved, if one's criterion is based on the number of examples presented to BOOLE either from the environment or from memory. The optimal values of *memory-ratio* lie near 0.25, suggesting that if one wishes to reduce total trials, one should maintain a list of hard examples in memory and process one of them for each four examples processed from the environment.

1. Run BOOLE as usual on an external example.

2. Compute the weight of the external example using BALANCE or WRONG.

3. Place at the head of the *memory-examples* list the ordered triple consisting of the external example's weight, stimulus pattern, and correct response.

4. Remove from the *memory-examples* list any example with weight less than or equal to .00000000000001.

5. Sort *memory-examples* by decreasing order of the weight of its members.

6. If the length of *memory-examples* is greater than *memory-length*, remove the last member of *memory-examples*.

7. If fewer than 10 examples have been presented to BOOLE, go to step 1. Otherwise, run memory examples as described in step 8 and go to step 1.

8. Where the length of *memory-examples* is N, cycle through each example on the *memory-examples* list using a probability level = *memory-ratio* / N.
Carry out the following procedure on each example:

   A. If a random test of the probability level fails, do nothing with the example.

   B. Otherwise,

      a. Present the example to BOOLE and reward BOOLE using the example's answer as the correct response.

      b. Replace the example's weight with the weight computed by BALANCE or WRONG.

      c. Run BOOLE's genetic algorithm procedure.

Figure 3: Temporary memory regime for reducing the total number of external and internal presentations.

METHOD	*MEMORY-RATIO*			
	.1	.25	.5	1.0
BALANCE	980	913	945	1131
WRONG	998	959	1025	1041

TABLE 3: Total trials to 90% performance for BOOLE with memory on the 6 multiplexer, averaged over 22 runs. Without memory BOOLE requires ~1088 external presentations to achieve 90% performance.

METHOD	*MEMORY-RATIO*			
	.1	.25	.5	1.0
BALANCE	7502	6703	6854	7877
WRONG	7337	6985	7659	6680

TABLE 4: Total trials to 90% performance for BOOLE with memory on the 11 multiplexer, averaged over 21 runs. Without memory BOOLE requires ~7598 external presentations to achieve 90% performance.

A point of interest to us in future work is understanding the effects of BALANCE and WRONG on adaptive systems. These two regimes maintain quite different lists of examples--in fact, once the use of memory has gotten under way, systems using BALANCE and WRONG on the 11-multiplexer problem contain non-intersecting memory lists. Investigation of the conditions under which these regimes perform poorly and well is one of our future goals.

6. Conclusions

We have shown that the idea of adding rudimentary memory capabilities to a stimulus-response classifier system can improve its performance under either of two natural performance measurements. The most dramatic improvements are obtained, as one might expect, when the system is not penalized for processing examples in memory. However, clear improvements may be obtained as well when the use of memory incurs a penalty equal to that for processing examples originating in the system's environment.

The sort of memory we have introduced here--a simple fixed-length list of examples that have been experienced--is not the only sort of memory one can envision combining with adaptive systems, and other types should be explored. We believe that investigation of those other types, and continued investigation of the memory regimes detailed here, is likely to produce interesting results.

References

[1] Anderson, C. W. (1986). Learning and Problem Solving with Multilayer Connectionist Systems. Ph.D. Dissertation (Computer and Information Science). The University of Massachusetts.

[2] Barto, A. G. (1985). Learning by statistical cooperation of self-interested neuron-like computing elements. Human Neurobiology 4, 229-256.

[3] Bonelli, P., Parodi, A., Sen, S., and Wilson, S. (1990). NEWBOOLE: a fast GBML system. Machine Learning: Proceedings of the Seventh International Conference (pp. 153-159). San Mateo, CA: Morgan Kaufmann.

[4] Bower, Gordon H. and Hilgard, Ernest R. (1981). *Theories of Learning.* Englewood Cliffs, NJ. Prentice-Hall.

[5] Davis, L. (forthcoming). Covering and memory in classifier systems. To appear in proceedings of the October 1992 Workshop on Classifier Systems, Houston, Texas.

[6] Davis, L. (1991). Handbook of genetic algorithms. New York: Van Nostrand Reinhold.

[7] Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. Reading, MA: Addison-Wesley.

[8] Holland, J. H. (1986). Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (eds.), Machine learning, an artificial intelligence approach. Volume II. Los Altos, California: Morgan Kaufmann.

[9] Michalski, R. S. (1986). Understanding the nature of learning: issues and research directions. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (eds.), Machine learning, an artificial intelligence approach. Volume II. Los Altos, California: Morgan Kaufmann.

[10] Quinlan, J. R. (1988). An empirical comparison of genetic and decision-tree classifiers. Proceedings of the Fifth Internation Conference on Machine Learning (pp. 135-141). San Mateo, CA: Morgan Kaufmann.

[11] Wilson, S. W. (1987). Classifier systems and the animat problem. Machine Learning 2, 199-228.