R. M. Friedberg

A Learning Machine: Part I

Abstract: Machines would be more useful if they could learn to perform tasks for which they were not given precise methods. Difficulties that attend giving a machine this ability are discussed. It is proposed that the program of a stored-program computer be gradually improved by a learning procedure which tries many programs and chooses, from the instructions that may occupy a given location, the one most often associated with a successful result. An experimental test of this principle is described in detail. Preliminary results, which show limited success, are reported and interpreted. Further results and conclusions will appear in the second part of the paper.

Introduction

We are seldom satisfied to have assigned our more laborious tasks to machinery. We turn with impatience to whatever still occupies our time and ask whether ingenuity cannot bring it, too, into the domain of automation. Although modern electronic computers have relieved us of many tedious calculations, we are still faced with difficult tasks in which the slowness of our thoughts and the shortness of our memory limit us severely, but for which present machines are less adequate than we because they lack judgment. If we are ever to make a machine that will speak, understand or translate human languages, solve mathematical problems with imagination, practice a profession or direct an organization, either we must reduce these activities to a science so exact that we can tell a machine precisely how to go about doing them or we must develop a machine that can do things without being told precisely how. This paper explores the second possibility.

If a machine is not told *how* to do something, at least some indication must be given of *what* it is to do; otherwise we could not direct its efforts toward a particular problem. It is difficult to see a way of telling it *what* without telling it *how,* except by allowing it to try out procedures at random or according to some unintelligent system and informing it constantly whether or not it is doing what we wish. The machine might be designed to gravitate toward those procedures which most often elicit from us a favorable response. We could teach this machine to perform a task even though we could not describe a precise method for performing it, provided only that we understood the task well enough to be able to ascertain whether or not it had been done successfully.

Such a machine, it may be objected, would only follow precise orders just as present computers do. Even if it acted sometimes at random, we should have to give it a method for generating random numbers. We should have to give it a method for correlating its behavior with our responses and for adjusting its behavior accordingly. In short, although it might learn to perform a task without being told precisely how to perform it, it would still have to be told precisely how to learn.

This is true, but it does not lessen the desirability of such a learning machine. On the one hand, even the simplest feedback devices do things without being told exactly how. A thermostat is not told at what level to keep the furnace running, although it is told how to readjust the furnace if the room temperature is too high or too low. On the other hand, even the most powerful learning device known may well follow a precise program at a very elementary level. It appears that each neuron of the human brain follows laws of cause and effect, but the organization of the brain is so complex that a determinism is not manifest in its activities as a whole. Between the thermostat and the brain there may be no gulf in principle. Yet in practice there is a gulf so wide that bridging it would be an enormous achievement. When we look at the mechanism of a thermostat, we can see in detail how the thermostat does its job. When we examine the parts of the brain, we are at a loss to understand, from their properties, how the brain does what it does, except in a vague way. What we want, then, is to equip a machine with a learning procedure by which it can develop methods that cannot, at least, be deduced *trivially* from an examination of the learning mechanism.

Certain difficulties confront us immediately. If, as suggested earlier, the machine is to try out methods and select the better ones, we must present it *a priori* with a well-defined universe of methods from which it must choose those to be tried. If this universe is small, then the "inventiveness" of the machine is severely limited and the value of the methods that it develops depends more on our astuteness in choosing a universe containing good methods than on the ability of the learning procedure to pick the best methods from among those in the universe. For example, we might design a method that used several parameters and cause the learning procedure to vary the parameters until it found the most successful set of values. The universe of permissible methods would then consist of all methods combining the form we devised with arbitrary values of the parameters. While this might yield excellent results for some problems, for others we probably could not devise any general form which did not exclude some methods much superior to any it included. In order really to give the learner a "free hand," we should present it with a universe which, although well-defined, is so large and varied that we are not even acquainted with the forms of all the methods it contains. Of course we must expect that in any universe so uncensored the majority of methods are useless.

This raises another difficulty in turn. If the universe is very large, the learning procedure cannot practically try out each permissible method repeatedly in order to evaluate it. Methods that resemble one another must be associated in classes, and a record must be kept on each class. In this way the success or failure of a method will be interpreted as a reflection not only on that method but on all its classmates. Thus a large universe of methods may be sifted in a relatively small number of trials, provided that the criterion by which two methods are classed together is a good one. So again the effectiveness of the learner may be limited by the inadequacy of whatever principle we devise.

Our experience of stored-program computers suggests a scheme which may fit the requirements. Let the universe of methods consist of all programs that can possibly be written for a given computer. This universe is well-defined, yet presumably it excludes no conceivable method except by reason of the computer's size, and even for a small computer it includes a great many of the methods that ingenuity might discover, although senseless programs are naturally in the majority. Let a class consist of all programs having a certain instruction in a certain location. Thus each program is a member of as many different classes as there are locations, and the learning procedure, in comparing the performance of two non-overlapping classes of programs, really evaluates one instruction against another that might occupy the same location.

At first thought it seems that not much can be expected from this plan of classification. Surely, having individual instructions in common is only a superficial resemblance between programs. Programmers know all too well that two programs may have almost identical *form,* differing only in one or two instructions, and yet have entirely different *intent,* one carrying out the programmer's wishes and the other producing "garbage." On the other hand, a very slight change in intent may require a drastic change in form, as when an instruction is inserted and a whole block of instructions must be displaced, so that no location in that block contains the same instruction as before.

Nevertheless, the scheme can be defended. Form and intent, to be sure, are related quite discontinuously in the compact, economical programs that programmers write, but a learning machine would probably develop much more inefficient programs in which many irrelevant instructions were scattered among the instructions that were essential to the intent. Among such programs, slight changes in form might well correspond to slight changes in intent, so that programs falling into the same classes tended to perform similar acts.

The versatility of the scheme is in its favor. In order to make the learner turn its attention from one problem to another, one need only change the criterion by which one informs it of success or failure. Moreover, we wish our machine not merely to learn to solve one isolated problem after another, but to develop an ability to handle whole classes of related problems. Programmers have found that certain sequences of instructions, or subroutines, occur again and again in many of their programs. It is as though any sensible program, no matter what its purpose, must rest on the same basic fabric of program organization. From this point of view, it is quite plausible that the learner, by including certain subroutines in a program, could improve greatly its chances of adjusting the rest of the program so as to perform a task successfully, regardless of just what task was assigned to it. The instructions composing such a subroutine ought to acquire good records, since the class of programs having these instructions in common would contain a particularly high concentration of successful programs. If the learner were to improve its general performance by attaching good records to the instructions composing a number of valuable subroutines, we might justifiably say that it had acquired not a mere *habit* of answering a certain problem correctly, but a general *ability* to do well on a large class of problems.

It is true that a subroutine usually consists of several instructions, and we propose here to evaluate only single instructions, since keeping a record even on all pairs of instructions would require enormously more time and storage. Perhaps the scheme could be improved by giving the learner a flexible way of reassigning its bookkeeping space. A special record might be kept for a pair of instructions if programs containing the pair did considerably better than programs containing either instruction without the other. On the other hand, the record on a single instruction might be dropped if programs containing it did neither better nor worse, on the average, than other programs. However, the scheme as it stands may well suffice for learning subroutines. Suppose that those programs which contain a special pair of instructions tend very often to be successful. Each member of this pair should enjoy from the outset a slight statistical advantage over its competitors, because among the (admittedly rare) programs tested that contain the other member of the pair, those that also contain the first member are more often successful than those that do not. This statistical advantage should cause the learning procedure more and more often to select programs for trial that contain one or both of the pair. The more often each member of the pair is used, the greater advantage does the other enjoy over its competitors. Eventually both members of the pair should have good records and be used often. The same process is conceivable for subroutines of arbitrary length.

Plausible though the foregoing arguments may sound to sympathetic ears, the critical mind notes that they depend more on far-fetched assumptions and less on demonstrable premises, the further they proceed. One may doubt seriously that a machine can really accomplish anything by trying out many programs and keeping a record in which each instruction is associated with the successes and failures of programs containing it. Supposing that this procedure did lead to some progress, one may ask whether even the simplest problem would not require the trial of an astronomical number of programs, especially if progress were to depend on the gradual influence of very small statistical differences. Therefore, an experiment was begun to test a learning procedure of this type. A hypothetical computer was designed for this purpose and called Herman, the letters of which stand for nothing in particular. Herman has a very simple logic such that every number of 14 bits is a meaningful instruction and every sequence of 64 instructions is a performable program. An outside agent called the Teacher causes Herman's program to be performed many times and examines Herman's memory each time to see whether a desired task has been performed successfully in that trial. The Teacher's announcements of success and failure enable a third element, the Learner, to evaluate the different instructions which, on different occasions, appear in Herman's program. Basing its acts on this evaluation, the Learner tries to include "good" instructions in the program rather than "bad" ones. The experiment is run by simulation of these three elements on the IBM 704 Electronic Data Processing Machine.

The remainder of Part I contains a description of the experiment and some early results. The experiment is unfinished at the time of the present writing. Part II will appear later with additional results and conclusions drawn from them.

Experimental methods

• *Computer*

Herman is a sequential stored-program computer with a program of 64 instructions in locations numbered I_0 to I_{63}. During the running of this program, the instructions are not modified, but they may be modified between runs by the learning procedure. The program itself acts upon the data in 64 locations numbered D_0 to D_{63}. Each data location D_n contains one bit. Each instruction location I_n contains a 14-bit instruction. When an instruction is executed, its first two bits are interpreted as an operation code; its next six bits, which form a number a, are interpreted either as a data address D_a or as an instruction address I_a, depending on the operation code; and its last six bits are also interpreted either as a data address D_b or as an instruction address I_b. The way in which the instruction is executed depends not only on the operation code and on the two numbers a and b, but also on the number n of the location I_n in which the instruction is stored. If

$\overbrace{n}^{\text{6 bits}}$ is the number of a location I_n containing

“$\overbrace{op}^{\text{bits 1-2}}\ ,\ \overbrace{a}^{\text{bits 3-8}}\ ,\ \overbrace{b}^{\text{bits 9-14}}$” ,

then the instruction in I_n is executed as follows:

If $op = 0$, take the next instruction
from I_a if D_n contains 0,
from I_b if D_n contains 1.

If $op = 1$, put into D_n, 0 if either D_a or D_b contains 0,
1 if both D_a and D_b contain 1.
Take the next instruction from I_{n+1}.

If $op = 2$, put into D_b the bit that appears in D_a. Take the next instruction from I_{n+1}.

If $op = 3$, put into D_n and into $D_{a(I_b)}$ the complement of the bit that appears in D_a. (The number $a(I_b)$ is found in positions 3 to 8 of the instruction location I_b.) Take the next instruction from I_{n+1}.

The choice of these particular operations was partly arbitrary and partly based on thought. The operations are powerful enough so that any procedure can in principle be programmed. (Actually the finite size of this computer makes sufficiently complex problems unprogrammable, but it was anticipated that the 64 instructions allowed would be more than were needed to program any of the problems submitted to the computer in the course of the present limited project.) The operations are simple enough not to impose any plan of organization on the data handled by the computer in the way that the structure of the IBM 704, for example, naturally groups the bits in storage into 36-bit binary numbers.

The peculiar use of addresses deserves explanation. As suggested above, the validity of keeping a separate record of success and failure for each instruction may be questioned on the ground that an instruction might be particularly well suited to play a part in an organized program and might thus tend to acquire a good record *as long as a certain other instruction was in the program;* when this other instruction was removed from the program, the former instruction might cease to have any virtue, and its good record would be misleading. For example, if Instruction 31 were to place the result of a calculation in a data location *x*, and if Instruction 32, which followed it, were to use the datum in location *x* to perform a calculation, these two instructions would be related in a sensible way and might be expected to contribute to the chances of success of a program containing them. In a conventional machine this relationship could not be viewed as a property of either instruction alone, for it depends on their both using the same data location *x*. If Instruction 31 were changed, Instruction 32 would lose its virtue unless (unlikely occurrence) the new instruction happened also to place data in location *x*.

In Herman, Instruction 32 might be "3, 31, 33." If Instruction 31 has an *op* code of 1 or 3, it places the result of the operation in D_{31}, the very location from which a datum is taken by the instruction "3, 31, 33." This is true no matter what the address bits in Instruction 31 are. Similarly, if Instruction 33 has an *op* code of 1, 2, or 3, it takes a datum from location D_a, where a is the number appearing in bits 3 to 8 of Instruction 33. In this same location is placed the result of executing the instruction "3, 31, 33." This is also true no matter what the number a is, or what the number in bits 9 to 14 of Instruction 33 is. Therefore, if the instruction "3, 31, 33" appears in location I_{32} and Instructions 31 and 33 are varied at random, the probability is ½ that Instructions 31 and 32 are related in the "sensible" way described above and ¾ that Instructions 32 and 33 are so related. If the instruction "3, 31, 33" in I_{32} acquires a good record, it may be expected therefore to continue to justify this record even if Instructions 31 and 33 are altered frequently.

It was for this reason that D_n was used in *ops* 0, 1 and 3 and that $a(I_b)$ was used in *op* 3. These features were not believed to eliminate the dependence of the virtue of an instruction on the presence of another instruction in the program, but they were expected to reduce it. Objections can be made. There may be "sensible" relationships other than that discussed above, relationships which depend on more than one instruction in spite of the special features of Herman. Or, if a learning procedure such as the one envisioned achieves success, it may do so by means of a program which has no characteristics that we would consider "sensible." Nevertheless, certain of the results to be described indicate that the special address features of Herman may have contributed to such success as was achieved.

• *Operation*

Before each trial of Herman, the Teacher places bits chosen at random in certain of the data locations (the input locations). The contents of the remaining data locations are left as they are from the preceding trial. Herman is started at Instruction 0 (that is, the first instruction to be executed is taken from I_0). If Instruction 63 is executed and is not a transfer instruction, then (there being no Instruction 64) Herman's program is considered to have *finished*. If this happens, the Teacher examines the contents of certain of the data locations (the output locations) and decides whether the bits in these locations satisfy a certain relation with the bits placed in the input locations at the beginning of the trial. If the relation is satisfied, the Teacher notifies the Learner of a success; otherwise, of a failure. This notification is the only information that the Learner receives about what the Teacher is doing. Neither the Learner nor Herman's program is "told" which of the locations D_0 to D_{63} are input locations, which are output locations, or what the Teacher's criterion of success is. This is primarily because no way was seen of making any of this information useful to the program or to the Learner without imposing one's own preconceptions on the way in which Herman might attack a problem.

Because of the transfer instruction (*op* 0), it is quite possible for Herman's program either to finish by reaching Instruction 63 after executing only a few instructions or to run for a long time or forever without finishing. Hence an arbitrary upper bound is set on the length of time a program may run on one trial. If, after the length of time required to execute 64 instructions, the program has not finished, Herman is stopped and the Learner is notified of a failure. It was believed that the problems to be presented to Herman could easily be solved by programs that would finish in considerably fewer than 64 instructions.

The choice of a subset of the 64 data locations to serve as input locations, the choice of a subset to serve as output locations, and the choice of a criterion by which the Teacher judges between success and failure together determine a single problem. It was intended that a single problem be presented to Herman for many (e.g., 50,000) successive trials, so that the input locations, the output locations, and the criterion of success would be fixed, while the bits placed in the input locations would be chosen anew at random before each trial. For example, the first problem that was given to Herman, called Problem 1, has D_0 as the only input location, D_{63} as the only output location, and identity between the output bit and the input bit as the criterion of success. Before each trial in which this problem is presented, the Teacher generates a random bit (0 or 1), records it as the input bit, and places it in D_0. If the program finishes in the time permitted, the Teacher examines the bit in D_{63} and notifies the Learner of success or failure according as this bit is the same as the input bit or not. After this has been done for many trials, the Learner should have evolved a program for Herman which will reproduce in D_{63} the bit presented to it in D_0, if not infallibly, at least in a large fraction of the trials.

• *Learning Procedure*

There are 2^{14} different instructions that could possibly occupy a single location I_n. It would be impractical to keep a record on each of these. Instead, two instructions (chosen initially at random) are "on record" at any time for each location I_n, so that there are altogether 128 instructions on record. For each I_n, one of the two instructions on record is "active" and the other is "inactive." In any trial the program executed by Herman consists of the 64 active instructions. The Learner has two ways of altering the program. It frequently interchanges the two instructions on record for a single location, so that first one and then the other becomes the active instruction. This process may be called "routine change." Occasionally the Learner makes a "random change"; that is, it erases one of the 128 instructions from the record and replaces it with a new 14-bit number chosen at random. The routine changes enable the Learner to accumulate data on the relative success of the two instructions on record for each location and gradually to favor the more successful instruction. The random changes are made in order that the Learner not be restricted to the 2^{64} programs that can be made from the instructions on record at any one time.

Both the routine and the random changes are governed largely by a number associated with each instruction on record, called its "success number." The success number is supposed to indicate how well an instruction has served over many thousands of previous trials. Each time a success is reported, the success number of every active instruction is increased by 1. (If the program finished the successful trial before executing more than 32 instructions, the success numbers are increased by 2 instead of 1. This is done in order to encourage the development of programs that do not take a long time to finish, because it was anticipated that the success of the project might depend on the number of trials that could be simulated in the limited computer time available.) When a new instruction is placed on record by a random change, its success number is set initially to a constant S_i. When any success number becomes equal to or greater than a constant S_m, all 128 success numbers are scaled down, i.e., multiplied by a constant r less than 1. The original design used $S_m=2^{15}$, $S_i=\frac{7}{8}S_m$, $r=\frac{63}{64}$. There are two reasons for scaling. One reason is to keep the success numbers at a roughly constant average size, so that they are comparable with S_i. The other reason is to diminish slowly the importance attached to the relative success that various instructions enjoyed a long time ago, compared with the importance of their more recent relative performance. For example, if one instruction achieves 64 more successes than another and thereby acquires a success number that exceeds the other's by 64, scaling will preserve the ratio between the two success numbers but will lessen the difference so that it can be made up by only 63 successes of the second instruction.

Each instruction location I_n has at any time a "state number" which plays a part in determining routine changes. Each time a failure is reported, a certain location I_n is subjected to "criticism." The absolute difference between its state number and the success number of its inactive instruction is taken as the new state number. If the old state number was less than the success number of the inactive instruction, the inactive instruction becomes active and the active instruction becomes inactive. Otherwise the two instructions are left as they are found. Each instruction location in turn, $I_0, I_1, \ldots, I_{63}, I_0, I_1, \ldots$, is subjected to criticism after successive failures. The reason that all the locations are not subjected to criticism after each failure is partly to save computer time and partly to ensure, by making routine changes one at a time, that a large number of different programs will be tried out.

The effect of this method of criticism is that the ratio between the success numbers of the two instructions governs the relative frequency with which each instruction emerges as active. Thus, if the success number of one instruction is roughly twice that of the other, a routine change will ordinarily be made whenever a criticism finds the latter instruction in the active position, but then it will usually require two criticisms to dislodge the former instruction from the active position. But, since exactly 64 failures must occur between successive criticisms of a single instruction location, an instruction may remain active for many more trials than its rival even though its success number is lower, simply by being less often associated with a failure. Thus the frequency with which each instruction on record is active depends partly on how well it is performing currently and partly on its long-term record, represented by its success number. When a certain set of instructions has been found to be successful in one problem and the Teacher now commences to pose another problem, it is intended that the frequent failures of the established program to perform the new problem will induce the Learner to alter the program and to use most frequently the instructions that are most often successful at the new problem. At the same time the instructions that were successful at the old problem ought not to be "forgotten," but should (at least for some time) retain their high success numbers, so that if the old problem is presented again the "memory" of these instructions will aid the Learner to arrive at a successful program. It should be emphasized that a change of problem is not signaled explicitly to the Learner but makes itself felt solely through the report of success or failure after each trial. The ability of the Learner to associate an instruction with a highly favorable long-term record, even while that instruction is currently inactive because it does not serve well in the problem at hand, is felt to be essential to the retention of things once learned.

A random change is made after every 64th failure. The instruction to be replaced is chosen from among one of four groups: the active and the inactive instructions in odd-numbered locations, and the active and the inactive instructions in even-numbered locations. These four groups are considered in turn, one at each random change. Of the group to be considered, an instruction with the lowest success number is replaced by a random instruction, which is given the success number S_i. The reason for dividing the 128 instructions on record into four groups is that considerable computer time is required to find the lowest of 128 numbers. The purpose of timing random changes every so many failures is to make them infrequent when the program is doing well.

The random instruction is obtained from a multiplicative random-number generator. A 35-bit binary random number is multiplied by $23 \times 10^{10}+1$ and divided by 2^{35}. The remainder of this division, a 35-bit number, is taken as the new random number. The quotient yields a random instruction. If we call the lowest-order bit of the quotient bit 1, the *op* code is taken from bits 29 and 28, the first address is taken from bits 25 to 20, and the second address is taken from bits 7 to 2. The way of extracting an instruction was determined by the requirements of simulating Herman on the IBM 704. This random-number generator was chosen because it takes little time and storage and was known not to give zero for many more generations than the project would require. It was not considered necessary that the random instructions used should pass any particular sophisticated test of randomness. The starting random number was 10987654321 (decimal).

The parameters had to be adjusted by guesswork. Random changes should be made often enough so that a variety of programs is available to the Learner, but not so often that "good" instructions are erased from record before they can establish their superiority. The rate at which the importance of ancient successes is diminished by the scaling of success numbers may be estimated as $-\log r/(1-r)S_m$, and is therefore roughly independent of r [since $(1-r) << 1$] and inversely proportional to S_m. This rate should, perhaps, be made comparable to the rate at which the program is renewed by random changes. S_i should be lower than the success numbers of some instructions, so that an instruction may, by acquiring a high success number, preserve itself indefinitely from random changes, but not lower than the success numbers of all instructions, for then a new random instruction would almost surely be removed from record by the next random change before it had a chance to establish its worth. If r is too low, the mass of success numbers will undergo large fluctuations that disturb their relationship to S_i. If r is too high, the rounding error in scaling will distort the ratios of success numbers.

- *Simulation*

Herman, the Learner, and the Teacher are simulated together in the IBM 704. The program runs from 5,000 to 10,000 trials of Herman each minute, including the intervening acts of the Teacher and Learner. The actual execution of Herman's program is the most time-consuming part of each trial. The part of the program that simulates the Teacher is rewritten or altered from day to day so as to present different problems or introduce modifications into the Learner. At the end of each day's run the IBM 704 punches out binary cards representing the state of Herman and the Learner. At the start of a later run, these cards can be read in so that the run will continue as though the 704 had not stopped, with the same active and inactive instructions, success numbers, and state numbers as at the end of the previous run. If desired, the day's run may begin with randomly chosen instructions, success numbers, and state numbers. In the course of each day's run a printed record is produced which indicates whether a previous run is being continued, identifies that run, identifies the problem being presented and any modifications in Herman or the Learner, and lists the number of successes achieved by Herman in each block of 10,000 trials.

Results

At the time of the present writing, only a few preliminary results have been obtained. These do not present a complete or conclusive picture, but they do indicate roughly the capabilities and limitations of Herman, and they suggest avenues of further exploration. It is intended that a more exhaustive set of experiments will be performed and published as Part II of this paper.

Some of the experiments were begun with "random initialization"—that is, random values were assigned to the success numbers, the state numbers, and the instructions in Herman's program. Other experiments were begun with a "history"—that is, these numbers were all given the values they had had at the end of some previous experiment.

- *Experiment 1*

After random initialization, Herman was presented with "Problem 1." In this problem, D_0 is the input location, D_{63} is the output location, and the criterion of success is that the output bit should be identical to the input bit.

The number of successes obtained by Herman in each block of 10,000 trials is shown in Table 1. Herman's progress on the problem may be divided into three stages. In Stage 1 (the first 60,000 trials), the frequency of success climbed steadily from almost 0 to slightly less than ½. Since even a random program may be expected to succeed in Problem 1 in 50% of the trials in which it finishes within the time limit, it is fairly certain that during Stage 1 the time limit was being exceeded in a large fraction of the trials. The fact that the frequency of success stopped rising rather abruptly just before it would have reached ½ indicates strongly that the rise in Stage 1 represents a gradual elimination of time failures, and that at the end of Stage 1 Herman's program was finishing within the time limit in about 90% of the trials and obtaining successes in roughly half of the 90%. This conclusion is supported by the fact that the simulation of each 10,000 trials at the end of Stage 1 required only about half as much running time on the IBM 704 as at the beginning of Stage 1.

In Stage 2 (the next 90,000 trials), the frequency of success fluctuated around roughly 45%. Apparently Herman was making no progress toward achieving the desired relationship between the output bit in D_{63} and the input bit in D_0. In Stage 3 (the last 50,000 trials) Herman suddenly "hit the jackpot." Since the Learner changes Herman's program only after a failure, it is obvious that if Herman hits on a program that is certain of success that program will remain unchanged as long as the same problem is presented. The program used by Herman during Stage 3 is reproduced in Chart 1. A careful examination reveals that it is certain to succeed indefinitely at

Problem 1, no matter what sequence of input bits it is given.

Table 1 *Problem 1*

Block of 10,000 Trials	*Number of Successes*
1st	26
2nd	511
3rd	1,822
4th	3,057
5th	3,853
6th	4,648
7th	4,741
8th	4,601
9th	4,387
10th	4,623
11th	4,123
12th	2,488
13th	4,246
14th	4,554
15th	4,382
16th	10,000
17th	10,000
18th	10,000
19th	10,000
20th	10,000

It appears that the Learner accomplished nothing in Stage 2 except to cast about at random until it hit upon a perfect program. However, one may contend that during Stage 2 the Learner was improving Herman's program in a way which did not increase immediately the frequency of success but which gradually increased the probability that further modifications would result in a perfect program. This contention receives some support from Experiments 2 and 3.

• *Experiment 2*

Problem 1 was presented after random initialization. The same Learner was used as in Experiment 1, but Herman was replaced by a slightly different computer which we may call Sherman. The latter is exactly like Herman except for some modifications in the way an instruction is executed. When $op = 0$, the conditional transfer depends on the bit in D_a instead of on that in D_n. When $op = 1$, the result of the operation is placed in D_{b+1} instead of in D_n. (If $b = 63$, $b + 1$ is taken as 0.) When $op = 3$, the result of the operation is placed in D_b and in $D_{b\pm 32}$ instead of in D_n and in $D_{a(I_b)}$.

Sherman is about as powerful a computer as Herman, but it lacks the two features—the use of the instruction location as a third address and the indirect address $a(I_b)$—which were intended to increase the likelihood that a meaningful performance record for a single instruction could be kept independently of other instructions in the program. Experiment 2 was designed to show whether these two features actually contribute to Herman's performance.

Sherman passed through Stage 1 and Stage 2 much as Herman did. Stage 1 took about 70,000 trials. During Stage 2, Sherman seemed to achieve greater average success than did Herman. The most striking result of Experiment 2 is that Stage 3 never arrived. Although the experiment was run for 800,000 trials, Sherman never succeeded in more than half of any 20,000 successive trials, whereas Herman, in Experiment 1, acquired a perfect program in 150,000 trials. One might suppose that Herman succeeded because the random initialization at the beginning of Experiment 1 was carried out with a "fortunate" set of random numbers. This hypothesis was tested by the next experiment.

• *Experiment 3*

Experiment 1 was rerun nine times, and random initialization was carried out with a different set of random numbers before each rerun. If the advent of Stage 3 in Experiment 1 were due merely to a lucky choice of random numbers for initialization, Stage 3 would probably not occur in the reruns.

The course of a rerun was not always marked by a clear division between Stage 1 and Stage 2. Sometimes the initial rise in frequency of success leveled off below 40%. Sometimes Stage 2 was so short that it could not be distinguished from Stage 1. Sometimes the initial rise was irregular instead of being smooth, as in Experiment 1.

Stage 3 arrived in every rerun except the last, during which the time allotted to the experiment ran out after 220,000 trials. In the other 8 reruns, the number of trials required for Herman to acquire a perfect program varied from 30,000 to 210,000, averaging about 100,000.

The IBM 704 was instructed in this experiment to discontinue each rerun as soon as at least 8,000 successes were obtained in a block of 10,000 trials. Four of the reruns ended with a block of 10,000 straight successes. Four ended with a block of 10,000 trials of which more than 8,000 but fewer than 10,000 were successes. It seems a safe inference that in each of the latter four a perfect program was obtained during the first 4,000 trials of the block and that the next block would have consisted of 10,000 successes. The results of every one of the 8 reruns were consistent with the supposition that Herman continued to succeed in fewer than 50% of the trials until a perfect program was found.

These results show that Herman's discovery of a perfect program in Experiment 1 was not a lucky accident. There are two ways to explain Herman's superiority over Sherman in finding programs perfect for Problem 1. Herman may surpass Sherman either in the number of perfect programs possible or in the efficiency with which the Learner can progress toward them. The features by which Herman differs from Sherman were actually designed with the latter possibility in mind, but the former cannot be ruled out.

Several more complicated problems were presented to Herman.

• *Experiment 4*

Starting from the end of Experiment 1 (that is, setting

Chart 1 **Program obtained in Experiment 1. Input datum from Location 0 transferred to Location 63.**

N	*D*	*OP*	*A*	*B*
0	**Input**	0	21	5
1	1	0	22	60
2	0	0	53	12
3	0	0	46	4
4	0	1	53	27
5	0	2	63	22
6	0	2	26	3
7	1	3	19	37
8	0	2	0	44
9	0	0	28	22
10	0	2	10	18
11	1	2	19	12
12	1	1	20	5
13	1	3	27	55
14	1	3	17	13
15	1	2	63	32
16	0	3	36	29
17	0	0	56	63
18	1	0	27	44
19	1	2	63	3
20	1	3	2	8
21	1	2	29	26
22	0	1	16	28
23	0	0	24	60
24	1	3	41	63
25	0	3	43	45
26	0	1	1	61
27	0	0	3	17
28	0	0	13	24
29	1	2	5	42
30	0	2	14	20
31	1	0	8	47
32	0	3	3	19
33	1	2	33	44
34	1	1	39	61
35	1	0	27	18
36	1	3	23	56
37	0	0	11	63
38	1	0	19	58
39	1	0	24	42
40	0	0	19	62
41	0	2	58	38
42	0	3	7	28
43	1	2	13	3
44	1	2	24	62
45	0	3	54	13
46	1	3	45	59
47	0	1	32	19
48	1	3	23	59
49	0	1	38	22
50	1	0	41	15
51	1	0	61	38
52	0	2	1	21
53	0	1	6	50
54	0	3	30	32
55	0	3	57	4
56	0	1	53	3
57	1	3	4	8
58	0	3	18	42
59	0	1	31	7
60	0	1	62	2
61	0	3	44	46
62	1	3	25	42
63	**Output**	3	11	18

Herman's program, the success numbers, and the state numbers as they were then), Herman was presented with Problem 2, which is the same as Problem 1 except that the output bit in D_{63} must be the *complement* of the input bit in D_0. Obviously a program that is perfect for Problem 1 must be phenomenally unsuccessful for Problem 2. Problem 2 was continued until a perfect program was attained (as inferred from at least 7,000 successes in a block of 10,000 trials). Then Problem 1 was given until a perfect program was attained. This was continued, with the hope that the Learner would presently be able to adapt quickly to whichever of the two problems was presented. Then it could be considered to have "learned" not just a solution to a single problem but a generalized ability to handle problems in which the input location is D_0 and the output location is D_{63}.

As may be seen from Table 2, the result was more or less as desired. It is not understood why the first adaptation was made so quickly or why the next few took longer.

Table 2 **Approximate number of trials required before perfect program was found (starting from the end of Experiment 1).**

Problems	*Number of Trials*
Problem 2	400 (est.)
Problem 1	80,000
Problem 2	140,000
Problem 1	230,000
Problem 2	20,000
Problem 1	500 (est.)
Problem 2	500 (est.)
Problem 1	200 (est.)

- *Experiment 5*

Starting from the end of Experiment 4, Herman was presented with Problem 3. In this problem, D_0 and D_5 are the input locations, D_{62} and D_{63} are the output locations, and the criterion of success is that the two-bit binary number formed by the output bits (taking the low-order bit from D_{63}) be the sum of the two input bits.

It was intended that the choice of input and output locations in the various problems follow a consistent plan, so that the Learner, faced with a new problem, would have to adapt only to the features of the problem that were really new, and not also to an arbitrary rearrangement of input and output locations. In the expectation that some problems might involve numbers as many as five bits long, the policy was laid down of letting the first input number occupy locations D_0 onward, starting with the low-order bit; letting the second input number occupy locations D_5 onward, starting with the low-order bit; and letting the output number *end* with the low-order bit in D_{63}. This policy was to be followed even in problems which, like all those discussed in this paper, involved numbers of fewer than five bits.

In 2,420,000 trials, Herman obtained 612,063 successes, or slightly more than one success in four trials, which is the expected average for a random program exclusive of time failures. The frequency of success fluctuated widely during the experiment, going below 14,000 out of 100,000 successive trials and above 33,000 out of 100,000 successive trials. It would be rash to infer from the data that there was a steady secular upward trend. The slight excess of the over-all average over ¼ should not be taken very seriously in view of the large short-term fluctuations. If the last 250,000 trials had been omitted, the over-all average would have fallen as far short of ¼ as it actually exceeded it. However, the closeness of the average to ¼ suggests that through all the fluctuations, which probably reflected changes in the program, Herman retained the habit of finishing usually within the time limit.

- *Experiment 6*

Starting from the end of Experiment 4, Herman was presented with Problem 4, in which a success is recorded if D_{63}, the only output location, finally contains the low-order bit of the sum of the input bits placed in D_0 and D_5, regardless of the final content of D_{62}.

The results followed the same pattern as Experiment 1. A perfect program, reproduced in Chart 2, was obtained after 940,000 trials.

- *Experiment 7*

Starting from the end of Experiment 6, Herman was presented with Problem 5, in which a success is recorded if D_{62}, the only output location, finally contains the high-order bit of the sum of the input bits placed in D_0 and D_5, regardless of the final content of D_{63}. It was hoped that if Herman acquired the ability to handle both Problems 4 and 5, it would not be too great a leap thence to progress to Problem 3, which combines 4 and 5.

No perfect program was obtained for this problem, although 2,740,000 trials were run. As in Experiment 5, there were large fluctuations in frequency of success. The total number of successes was 1,367,321, which falls short of half the number of trials by an amount insignificant in view of the short-term fluctuations.

It is supposed that the order in which problems are presented affects the learning process, although none of the experiments reported in this paper (Part I) show the effect clearly. Thus, the alternation of Problems 1 and 2 in Experiment 4 presumably encouraged the development of programs which were meaningful if D_0 was an input location and D_{63} an output location. This development presumably aided the subsequent learning of Problem 4; the only new location to be identified was D_5 as an input location. Once Problem 4 had been learned, the only new location to be identified in Problem 5 was D_{62} as an output location. The fact that Herman achieved more success in Problem 4 than in Problem 5 suggests three explanations:

1. that the logical function to be performed in Problem 5 (logical AND) is more difficult for Herman than that in Problem 4 (addition modulo 2);

2. that it is more difficult to identify a new output location than a new input location;

Chart 2 **Program obtained in Experiment 6. The addition modulo 2 of input data from Locations 0 and 5 is obtained in Location 63.**

N	*D*	*OP*	*A*	*B*	*N*	*D*	*OP*	*A*	*B*
0	**Input**	0	58	57	**32**	1	3	32	34
1	1	0	14	8	**33**	0	2	32	42
2	0	3	47	22	**34**	0	2	38	43
3	1	0	29	54	**35**	0	3	59	12
4	1	2	52	5	**36**	1	0	26	63
5	**Input**	0	45	24	**37**	0	2	37	18
6	0	0	23	11	**38**	0	2	11	45
7	0	3	4	37	**39**	1	3	5	56
8	0	0	18	24	**40**	1	0	18	29
9	0	3	18	57	**41**	1	3	49	23
10	0	0	2	21	**42**	0	0	2	33
11	1	0	10	34	**43**	0	2	63	9
12	0	0	32	58	**44**	0	0	54	14
13	1	0	40	56	**45**	1	1	33	47
14	0	2	53	5	**46**	0	3	1	16
15	0	0	22	35	**47**	1	3	57	19
16	0	0	14	25	**48**	1	3	16	34
17	1	2	41	59	**49**	0	1	14	29
18	1	2	23	30	**50**	0	3	38	47
19	0	3	45	51	**51**	1	0	43	39
20	0	3	41	26	**52**	1	0	41	9
21	0	1	32	44	**53**	1	0	2	21
22	1	1	38	13	**54**	1	2	1	25
23	0	3	5	56	**55**	1	1	63	23
24	1	0	46	53	**56**	0	1	5	8
25	1	0	26	63	**57**	0	0	46	24
26	1	2	34	43	**58**	1	2	35	26
27	1	1	62	5	**59**	0	1	18	20
28	0	2	10	19	**60**	1	3	15	29
29	0	1	58	61	**61**	0	2	27	24
30	0	1	17	33	**62**	0	3	11	58
31	1	0	63	32	**63**	**Output**	1	5	51

3. that Herman's experience with Problem 4 established D_5 less firmly as an input location than the previous alternation of Problems 1 and 2 had established D_0 and D_{63} as input and output locations, respectively.

The first explanation seems less promising than the other two. In any of these problems, learning to make the contents of certain locations depend on the contents of certain other locations seems a greater task than learning, once the input and output locations are identified, to make this dependence obey a certain logical function.

Experiment 8

If a large group of problems were learned, it would become cumbersome to return repeatedly to each one of the group in order to retain the ability to perform it. This could be done more easily if Herman could learn to perform different problems on successive trials. For example, if Herman could learn to perform Problem 1 in every other trial and Problem 2 in the intervening trials, the ability to perform both problems might be renewed, when necessary, by presenting this alternation of them. There is no reason why a single program could not perform a different act in successive trials, for the effect of executing a program depends not only on the instructions of which it is composed but also on the content of the data locations other than the input locations. These contents, in turn, were determined by the action of the program in the preceding trial.

As preparation, Problem 6 was presented, starting from the end of Experiment 4. This problem has no input location and one output location, D_{63}, and a success is recorded if the output bit is a 1 in an odd-numbered trial or a 0 in an even-numbered trial.

Herman achieved a perfect program for Problem 6 in fewer than 20,000 trials. This is noteworthy since the learning of time-dependent behavior is a particularly interesting phenomenon in its own right.

After a perfect program had been obtained for Problem 6, Herman was presented with the alternation of Problems 1 and 2. That is, the output bit in D_{63} was required to be the same as the input bit placed in D_0 in even-numbered trials and to be its complement in odd-numbered trials.

The frequency of success did not exceed 50% in any block of 10,000 trials, although the experiment was run for 2,130,000 trials. No perfect program was obtained. In all, 863,447 trials were successful.

Experiment 9

In Experiments 5 and 7, the frequency of success often stayed considerably above the expected fraction for a random program (¼ in Experiment 5, ½ in Experiment 7) for as many as 100,000 successive trials. It may safely be inferred that during those successful periods, programs were in use that tended to achieve success, although they did not achieve it infallibly. Since the Learner is supposed to retain the instructions comprising such programs, it is disturbing that these periods of success were often followed by periods in which the frequency of success was distinctly *below* the expected fraction. This indicates that "good" instructions were replaced by "bad" ones. In an effort to find out whether the Learner is capable at all of holding on to a "good" program which does not *always* achieve success, Problem 1 was presented to Herman after random initialization, and every tenth trial was ruled a failure no matter what Herman did. Thus even a perfect program would achieve success only in 9 trials out of 10.

Under these conditions Herman failed to retain a frequency of success higher than 45% for more than about 50,000 trials at any one time, although the experiment was run for 1,380,000 trials and often a single block of 10,000 trials yielded more than 7,000 successes. In all, 560,618 successes were obtained.

These results suggest that the Learner is seriously deficient in the ability to retain instructions that are statistically advantageous but not infallibly successful. This deficiency might be due to the random changes in the program. The Learner has features that were designed to protect "good" instructions from random changes, but perhaps the features did not work. To test this possibility, the Learner was modified so as to make no random changes. The preceding experiment was then repeated *starting from the end of Experiment 1,* so that there was at least one perfect program among the 2^{64} programs attainable by routine changes alone.

This experiment was run for 2,670,000 trials. During the first million trials, the frequency of success stayed fairly close to the expected random 45%. Although occasionally 7,000 or even 8,000 successes appeared in a single block of 10,000 trials, no three consecutive blocks each contained more than 6,000 successes. The last million trials included several successful periods, from 30,000 to 100,000 trials long, during which more than 8,000 and frequently just 9,000 successes were obtained in each block of 10,000. These successful periods were separated by normal periods, from 50,000 to 150,000 trials long, during which the frequency of success approximated 45%, as during the first million trials. The data do not suffice to show whether or not the successful periods would have grown longer and more frequent, had the experiment been prolonged. But the fact that the successful periods were absent at first and appeared after 1,700,000 trials indicates that the Learner, restricted to routine changes, is able gradually to favor particularly successful instructions even though they do not succeed more than 90% of the time.

The random changes cannot easily be dispensed with, however, for an unfavorable choice of a few key instructions can make it impossible to obtain a successful program from a given set of 128 instructions. The influential role of certain key instructions is perhaps an undesirable feature of the present scheme. For example, whenever Herman's program has been examined after a long string of successes, *op* 0 has been found in I_0 and *op* 1 or 3 in I_{63}, and reflection shows that these characteristics are almost necessary for a successful program if D_0 is used for input and D_{63} is used for output. An example of the

Learner's failure to protect "good" instructions from random changes is the fact that the key instructions in I_0 and I_{63} were changed in the course of Experiment 7.

The results obtained thus far, although fragmentary, show that in a practical number of trials a learning machine of the type described can achieve enough success to be suitable for informative experimentation. By continuing this kind of investigation, we may grow to understand the factors that influence the behavior of such a machine.

Received November 14, 1957