TRAINING SEQUENCES FOR MECHANIZED INDUCTION*

R. J. Solomonoff
Zator Company
Cambridge, Massachusetts

1. Introduction.

The present paper will not be primarily a description of work that has been done, but will emphasize plans to utilize work that has been done, to devise an "intelligent" machine capable of improving its own operation.

The machine's primary purpose is to solve problems in some particular field—for example, devising mathematical proofs, or, more generally, writing computer programs having certain desired characteristics. Along with these abilities we want the computer to be able to work constructively on the problem of speeding up, or otherwise optimizing its own operation. This machine would work on the self-improvement problem just as it would work on any other problem given to it. The criterion of its success at self-improvement need not be vacuously circular if it also has to solve problems different from self-improvement.

This work stems from an earlier paper (Ref. 1) which described a theoretical inductive inference machine meant to learn to work simple arithmetic problems, after having been given a number of correctly worked examples. The machine envisioned was to start with a certain basic set of abstract entities, which it would combine in various ways to make abstractions that could be used in predictions, since formally, any such learning problem is equivalent to a problem in prediction. The abstractions used successfully in prediction as well as the abstractions used to create them were given high values of "utility." Abstractions with high utilities were to be combined with each other to produce new abstractions to be used to make trial predictions in new problems. Again, the successful abstractions were given high utility values. This process of creating and testing new trial

*This work has been sponsored by the Air Force Office of Scientific Research through contract AF49(638)-376.

prediction abstractions was to be continued as new problems and their solutions were given to the machine.

While this approach to inductive inference seemed reasonable at one time, it soon became clear that I had no really rigorous method to compute utilities of abstractions—though I had many intuitive ideas about what properties this utility function should have.

After about a year and a half of work on the utility evaluation problem, I devised a solution that seemed to work in certain circumstances, but I wasn't at all certain that it could be applied to much more general problems. About this time I became interested in formal languages, and their apparent relationship to inductive inference. This work finally gave rise to what I believe to be a very general theory of inductive inference (Ref. 2) and it seems to have also solved the utility evaluation problem in a completely general manner.

The work and plans that I shall describe are a continuation and modification of the early work on mechanized induction in view of this more general theory of inductive inference.

It will be noted that the present work is in some ways similar in spirit to that of Simon, Newell, and Shaw (Ref. 3), in that the heuristic devices used for the projected machine are usually patterned after those used by humans. Perhaps an important difference is that I intend to use these human heuristics in only the rudimentary stages of machine development. As soon as possible, it is hoped that heuristics will be developed that will be more closely matched to the peculiarities of the machine technology used.

Similarity can also be drawn between the present work and that of Kilburn, Grimsdale and Sumner (Ref. 4), who constructed random computer programs in a successful mechanization of certain kinds of induction. Again their emphasis seemed to be on simulating certain aspects of human thinking and learning—which is a somewhat different goal from that of the present study. There was also emphasis on the supposed importance of using some random choices so as to get "original" behavior from the machine. My own feeling is that random choices can often be used to result in simpler, faster calculations than could be done otherwise, but that in general, fewer trial solutions have to be examined if optimum non-random methods are used, and the results will be more reliable and will look just as "creative" as the results obtained using partly random choices.

2. A General Description of the Machine and the Direction of Future Work.

Evaluation of arithmetic expressions is the first problem type for the machine of the present study. Expressions such as 13 - 8 or 9 + 1 - 3 or 3 - 1.2 × (1 - 18)/4 - 21 are given to the

machine as a sequence of numbers and other symbols. To start off, the machine "knows" which symbols are numbers, and which are not. The machine attempts to solve these problems by devising a program to evaluate all such input expressions.

Suppose that the first problem in the training sequence is to evaluate numerically the expression 3.5 + 9.128. This expression is given to the machine, along with the proper solution; i.e. 12.628. The machine then constructs programs at random, and tries each, in turn, on the input expression, until it finds one that gives the result 12.628. It then stops, and retains the successful program.

When a new problem, along with its solution, is given to the machine, it first tries the last successful program on the new problem. If the program is successful, it stops. If not, it makes new trial programs by modifying the old program. The new programs are tried out in turn until one is found that will correctly work the new problems as well as all of the old problems. When new problems are given, the latest successful program is tried as before, and if unsuccessful, modifications are tried, as before.

The critical question is, what modifications to try in attempts to "fix up" a formerly successful program. If very simple modifications are the only ones allowed, then the machine will only be able to work problems of training sequences in which the conceptual steps between the problems are very small, and are of the simple sort that can be dealt with using these simple program modifications.

If the programs are represented by linear sequences of symbols, then one very simple way to modify such a program, is to add on more symbols.

A significant increase in power is obtained if one of the symbols capable of being added on is to be interpreted as erasing one of the previous symbols.

The machine outlined above is to be viewed as a study for a more elaborate system. This system will, at first, consist of two machines. The problems of the first machine are to devise programs that are as optimum as possible with respect to a given criterion. Upon being given any criterion of program evaluation, it will try to write programs that will be optimum with respect to that criterion.

The second machine looks at the description of the first machine (the description being in the form of a program), and tries to optimize the first machine's operation with respect to speed, economical use of memory, or any other desired criterion.

It is clear that the job of the second machine is a special case of the general type of problem solvable by the first. As a result, the first machine *can* be given the problem of improving itself.

If the first machine does not have other problems to solve, the problem of self-improvement becomes trivial or meaningless. If it does have other problems, then self-improvement is a well defined problem. Furthermore, if the first machine has successfully gone through a training sequence containing problems of other program optimizations that are similar to the self-improvement problem, then we may expect it to be able to work with some success on its own improvement. Since most, if not all, interesting problems can be expressed in the form of the requirement of writing an optimum computer program with respect to a certain criterion, it is seen that the self-improving machine is, indeed, a very general, powerful device.

In what way is the simple arithmetic learning machine a step toward the more powerful self-improving machine? I will outline a sequence of expected developments from the simple machine that I feel are likely to culminate in this goal.

First, the heuristic devices used by the present simple machine will be expanded considerably. At the present time, the machine would take about 10^{10} trials to work some fairly simple problems. This would be decreased by a factor of about 20 by a slightly cleverer search procedure. Another large factor in speed could be obtained if the machine were able to recognize certain simple regularities in sequences of symbols, and another large speed-up could be obtained if it were able to recognize regularities such as exist in sentences that are generated by phrase structure grammars.

Various other heuristic devices must be devised until the problem of finding an acceptable program can be solved in a reasonable number of trials.

It next becomes necessary to mechanize the construction of new trial heuristics that have a reasonable probability of being useful. One way to do this is to first express the known heuristics in some sort of uniform, compact notation so that it is easy to see what characteristics the good ones have in common.

As a first approximation, it is possible to make new trial heuristics by using random combinations of the symbols occurring in the heuristics known to be good. By taking into account the frequencies with which these symbols occur, noticing certain intersymbol constraints and other regularities, it is possible to make trial heuristics that have much higher probabilities of being useful.

It will be noted that even at this simple stage of development, many of the heuristics that are useful for devising programs to solve simple problems in arithmetic and algebra, are similar to the heuristics useful in creating good trial heuristics.

We could not at this stage, however, allow the original problem-solving machine to work on the problem of improving itself,

since the problem of program optimization is somewhat different from the problem of writing programs that satisfy a certain criterion. In this latter case, a program either satisfies the criterion or doesn't. There is no "gray scale." In the case of the heuristic optimization problem there is a continuum of degrees of effectiveness for all heuristics.

It is believed, however, that in the present case changing the machine from the black-white problem-solving criterion to the more general gray scale criterion will not be very difficult. As was noted before, many of the heuristics for both types of problems appear to be identical or similar.

3. Work That Has Already Been Done.

Most recently some detailed analysis was made of the machine to learn to evaluate arithmetic expressions. As a possible input, consider the sequence of symbols

$$9 \times 8 \tag{1}$$

and the correct evaluation of it, 72. The machine must devise a program that transforms the sequence (1) into the number, 72.

In the present case, the only programs permissible are in the form of allowable substitution rules on the input expressions. The machine code for one such substitution rule is

$$A \times B \rightarrow M \times AB \tag{2}$$

Here A and B are meant to be any numbers, and "×" is a non-numerical expression. M is a special symbol to denote an operator. M × AB means "the number resulting when A and B are subject to the operation '×'"—or, more briefly, "the product of A and B."

The rule (2) means that whenever a sub-sequence of the form A × B appears in an input expression, then this subsequence may be transformed into the number which is the product of A and B. Rule (2) may also be applied to any permissibly transformed form of an input expression.

The set of rules

$$\begin{aligned} A + B &\rightarrow M + AB \\ A - B &\rightarrow M - AB \\ (A) &\rightarrow A \end{aligned} \tag{3}$$

is adequate for evaluating any expression containing numbers. +, -,), and (. The transformation rules are to be applied in arbitrary order to an input expression. When a sequence is

finally obtained to which no further transformations can be applied, this sequence is presented as output.

At the beginning, the machine will have available to it only the symbols

$$-, +, \times, /,), (, A, \rightarrow, M, \phi, \psi, n. \tag{4}$$

These symbols are at first selected at random to make strings of symbols to be used as trial sets of transformation rules. The symbol "ψ" is a termination symbol. A trial string terminates as soon as this symbol is selected.

The symbol "ϕ" indicates the end of one substitution rule and the beginning of another.

The symbol "n" is a null symbol and indicates that there is no symbol at all at the point occupied by n.

In addition to the symbols listed in (4), the machine can use positive integers. However, while the integer 1 may be used at any point in a trial sequence, the integer, i, can only be used if the integer i-1 has occurred sometime before in the trial sequence. These integers are not, however, "numbers," in the sense that M can operate on them. They are used as "subscripts" for the symbol, A. The symbol A as used in the set of rules (3) would be designated by A1, the symbol B, by A2.

The entire set of rules of (3) could be written by the machine as

$$A1 + A2 \rightarrow M + A1\ A2\ \phi\ A1 - A2 \rightarrow M - A1\ A2\ \phi\ (A1) \rightarrow A1\ \psi \tag{5}$$

If we constrain the trial sequences so that integers and only integers are allowed immediately following the symbol, A, then the probability of randomly constructing the sequence

$$A1 + A2 \rightarrow M + A1\ A2\ \psi$$

is about 10^{-10}. If we do not use this constraint on the integers, the probability is somewhat lower.

3.1. Training Sequences.

If the problems of evaluating 3 + 8, 7 - 4, and (3 + 2) are given to the machine in that order, and it is required to solve each one in turn, by devising a new substitution rule, then the set of rules, (3), would be obtained, after many, many trials for each new problem.

Suppose that we next give the problem

$$3 \times 4 + 1 \tag{6}$$

The creator of the problems will be using the convention that evaluation of multiplication and division precedes evaluation of addition and subtraction whenever possible.

The machine, of course, not knowing the convention, will first transform expression (6) into 3×5. From this point, there is no simple transformation that will yield the correct answer, 13.

It is clear that if it is not permitted to order the transformation rules, the machine must have some means of recognizing when the evaluation of 3×4 must precede the evaluation of $4 + 1$.

Context-dependent substitution does appear to be adequate for evaluation of expressions containing numbers, +, -,), (, ×, and /. Some rules that are adequate for evaluating (6) are

$$A \times B \rightarrow M \times AB \tag{7}$$

$$n\,A + B\,n \rightarrow n\,M + A\,B\,n \tag{8}$$

The rule (8) says that the substitution

$$A + B \rightarrow M + AB$$

is permitted in the context n,n.

Unfortunately, it is necessary that rules for evaluating A + B and A - B must all be context-dependent substitutions. In order to evaluate expressions like (6), the machine would first have to unlearn the rules (3). Since unlearning is a rather difficult process for humans and is even more difficult for the machine herein described, it would be well to avoid this difficulty, for the present. This can be done by suitably designing the training sequence, so that problems involving ×, /,), and (are given before those containing + and -.

If a suitable training sequence is given, the machine will discover three context-independent rules {for the symbols), (, /, and ×} and 50 context-dependent rules involving + and -.

If the machine has had no previous experience in problem-solving, it will take about 10^{10} tries to obtain a rule of the complexity of (7). After a reasonable amount of experience, it will take about one fiftieth as many trials for a rule of the same complexity, since the probabilities of choice of various of the symbols are then closer to the frequencies with which they occur in the rule.

One can identify the "a priori probability" of a rule with the product of the probabilities of its symbols. In the above examples of machine behavior, the trial rules have been selected at random, and the probability of a particular rule's being tried was

proportional to its a priori probability. In a more general case, we can set the probability of choosing a given rule equal to some function of the a priori probability of that rule. If we select this function so as to minimize the expected number of trials (these are "random trials with replacement"), then the optimum function is the normalized square root of the probability.

There is, however, some disadvantage in using this particular function. While fewer trials are indeed required, the rule obtained in this way will not have as great a probability of extrapolating properly to new problems.

A search technique that is optimum with respect to both speed and likelihood of correct extrapolation, consists of trying the rules in fixed order of a priori probability, with little or no randomness. For simply constructed rules, a search of this kind is not difficult to program, but for more complex kinds of rules, it may be very difficult to program and take more machine time than simpler search schemes that take, on the average, a larger number of trials.

In the present case, the number of trials can be reduced by a factor of about 20 if the optimum search is used. Using both optimum search and modification of symbol probabilities through previous experience, would give a speed-up factor of 1000, or about 10^7 trials for a simple rule.

A further increase in operating speed will be obtained if the machine is able to notice certain kinds of regularities in the successful rules. In particular, the 50 context-dependent rules, of which (8) is an example), are all of about the same form. They can all be compactly expressed with the notation.

$$\alpha_1 A \beta_1 B \alpha_2 \rightarrow \alpha_1 M \beta_1 A B \alpha_2 \tag{9}$$

$$\alpha = +, -,), (, n \tag{10}$$

$$\beta = +, - \tag{11}$$

To obtain any one of the 50 rules, it is necessary to select a value for α_1 (out of any of the five possible α values), then to select a value for β_1 (out of the two possibilities), and then to select a value for α_2 (out of the five possibilities).

Rule (8) can be obtained by the choices: $\alpha_1 = n$, $\beta_1 = +$, and $\alpha_2 = n$. To be able to construct a set of rules like (9), (10), (11), the machine must have a suitable notation convention. Also, it must have means for deciding that after it had discovered a few of the 50 individual rules, that it could represent them more compactly by defining, say, $\alpha = +, -,)$. Later it might find that even more compactness resulted if the definition $\alpha = +, -,), n$ was used.

A criterion for "compactness" is suggested in Ref. 2, and somewhat more relevant criteria are discussed in Ref. 5.

Evaluation of arithmetic expressions was taken as one of the simplest possible tasks for a machine to try to learn. Substitution rules are, in general, a powerful method of expressing instructions, and it has been possible to devise substitution rules that will solve linear equations. So far, a formalism of this kind has been devised for only one linear equation in one unknown.

3.2. Heuristics.

It was noted in the previous section, that a great deal of difficulty would be caused if the machine were given problems containing + and - before being given problems containing $\times$ and /. This difficulty stems entirely from the weakness of the heuristic devices available to the machine at its present state of development. In general, if the machine has few heuristic devices available to it, and these are not of great strength, then the training sequences will have to be very carefully constructed, so that the conceptual jumps between successive problems are within the capability of the machine. If the machine's heuristic devices are more powerful, then far less care need be exercised in training sequence construction.

4. Plans for Future Work.

There are no immediate plans for simulating on a computer any of the operations described. At the present time, it is felt that theoretical approximations for the number of trials required in searches for solutions are of more interest than would be the results of such simulation.

The immediate future will be spent in devising suitable heuristic devices for arithmetic evaluation and the solution of several linear equations in several unknowns. These will involve some expansion of the language in which solutions will be expressed. Attempts will be made to devise a compact, uniform notation for the heuristics, so that similarities between them may be found, as well as suggestions for the creation of new trial heuristics.

Creation of new heuristics may be viewed as the prime task of a higher order inductive inference machine, and it is in the operation of this higher order machine, that I am primarily interested. Such a machine is capable of self-improvement. Practically any learning or induction problem can be put into a form that is acceptable to a machine of this type. Whether any particular problem can, indeed, be solved by the machine will depend upon the power of the heuristics originally given to the

machine, upon the exact nature of the training sequence preceding the problem, and upon the time available for solution.

Similar remarks apply to human beings.

REFERENCES

1. R. J. Solomonoff: "An Inductive Inference Machine". IRE Convention Record, Section on Information Theory, pp. 56-62, 1957.
2. R. J. Solomonoff: "A Preliminary Report on a General Theory of Inductive Inference," Zator Technical Bulletin No. 138. AFSOR TN-50-1459, Zator Co., November 1960.
3. A. Newell, H. Simon, J. Shaw: "Report on a General Problem Solving Program." *Information Processing,* Butterworth Scientific Publications, London, 1960.
4. T. Kilburn, R. Grimsdale, F. Sumner: "Experiments in Machine Learning and Thinking." *Information Processing,* Butterworth Scientific Publications, London, 1960.
5. R. J. Solomonoff: "An Inductive Inference Code Employing Definitions." Zator Technical Bulletin No. 141, AFOSR 2214, Zator Co., 1961.