"MEMO" FUNCTIONS AND MACHINE LEARNING

By

DONALD MICHEE

"Memo" Functions and Machine Learning

by
DONALD MICHE

Experimental Programming Unit,
Department of Machine
Intelligence and Perception,
University of Edinburgh

It would be useful if computers could learn from experience and thus automatically improve the efficiency of their own programs during execution. A simple but effective rote-learning facility can be provided within the framework of a suitable programming language.

factorial (5) = 120; hcf (63, 18) = 9; member (Queen Elizabeth, the Cabinet) = false; and reverse ([Tom Dick Harry]) = [Harry Dick Tom]. The present proposals involve a particular way of looking at functions. This point of view asserts that a function is not to be identified with the operation by which it is evaluated (the rule for finding the factorial, for instance) nor with its representation in the form of a table or other look-up medium (such as a table of factorials). A function is a function, and the means chosen to find its value in a given case is independent of the function's intrinsic meaning. This is no more than a restatement of a mathematical truism, but it is one which has been lost sight of by the designers of our programming languages. By resurrecting this truism we become free to assert: (1) that the apparatus of evaluation associated with any given function shall consist of a "rule part" (computational procedure) and a "rote part" (look-up table); (2) that evaluation in the computer shall on each given occasion proceed either by rule, or by rote, or by a blend of the two, solely as dictated by the expediency of the moment; (3) that the rule versus rote decisions shall be handled by the machine behind the scenes; and (4) that various kinds of interaction be permitted to occur between the rule part and the rote part. Thus each evaluation by rule adds a fresh entry to the rote. Updating the rote by assignment from outside the function apparatus (such as from the tape reader or other input channel) is allowed, and may alter some of the consequences of future evaluations by rule. The rule

If computers could learn from experience their usefulness would be increased. When I write a clumsy program for a contemporary computer a thousand runs on the machine do not re-educate my handiwork. On every execution, each time-wasting blunder and error, each needless test and redundant evaluation, is meticulously reproduced.

Attempts to computerize learning processes date back little more than 10 yr. The most significant early milestone was A. L. Samuel's study1 using the game of checkers (draughts). Samuel devised detailed procedures both of "rote-learning" and "learning by generalization". When coupled with efficient methods of lookahead and search, these procedures enabled the computer to raise itself by prolonged practice from the status of a beginner to that of a tournament player. Hence there now exists a checkers program which can learn through experience of checkers to play better checkers. What does not yet exist is any way of providing in general that all our programs will automatically raise the efficacy of their own execution, Samuelwise, as a result of repeated "plays".

Here I outline proposals2 for enabling the programmer to "Samuelize" any functions he pleases, and so endow his program with self-improving powers—both of "rote-learning" and "learning by generalization". I shall be talking about mathematical functions. "Factorial" (of a factorial number) is a function; so is "highest common factor" (of a pair of numbers); so is "member" (of an element-set pair); so is the "reverse" (of a list). Thus

...
...
elementary level of the current "boxes" implementation, 
these functions predict no more than the expected lapse of time before the next crash. In the present context we are chiefly concerned with the use of equis in looking up a state-description on the rote and thus providing for approximation in state-space as a crude form of generalization. If we make the definition of equis program-modifiable in the light of emergent properties of the rote, then we can ensure that states (arguments) not worth distinguishing are lumped into a single rote entry, whereas those presenting difficulties of prediction are given a finer-grained representation. For example, we can have equis continuously up-dated in such a way that the result-values entered against adjacent values of those arguments which gain entry to the rote are always separated by more than a specified minimum. A thoroughgoing "lumping and splitting" facility is in this way automatically created.

Generalization by Interpolation

To see that approximation as described represents a primitive kind of generalization, we can relate it to interpolation. Figs. 2 and 3 depict the same set of data points. In the first case the gaps are partially filled by approximation. The lengths of the horizontal lines are set by the equis function, which decrees which values of the x-variable are to be regarded as equivalent to each other for look-up purposes. In the second case a polynomial is fitted to the points. The question arises as to whether in some cases a list of polynomial coefficients would not be a more powerful and condensed means of providing for quick and approximate evaluation than a rote; also whether combinations of rote and coefficient-list could be used. Just as in the rote-learning case the rote is up-dated by simple accretion of new x-y pairs, so in the case of polynomial interpolation the coefficient list can be up-dated (and the polynomial thus revised) to accommodate each newly added x-y pair. In this way a process of induction on the growing store of examples is carried out, as demonstrated and discussed elsewhere by Fredkin, Pivar and Finkelstein, and by Popplestone.

I thank the Science Research Council for their generous support for this work.

Received March 12, 1968.