

LEARNING PROGRAMS IN ARTIFICIAL INTELLIGENCE

John Mylopoulos and C. Raymond Ferrault
 Dept. of Computer Science
 University of Toronto
 Toronto Canada

Received January 21, 1979; July 27, 1979

ABSTRACT: It has been claimed of many programs developed in the area of Artificial Intelligence that their behaviour improves over time. We classify a broad range of these "learning" programs in a two-dimensional space. The structural dimension is used to describe the form of the algorithm being learned. The learning process dimension accounts for the differences in the mode of presentation to the Student of the desired input-output relation.

* * *

In this contribution to the session on "The Evolution of Artifacts", we take the term evolution in two senses, one broad and one narrow. We discuss the evolution in Artificial Intelligence (AI) of computer programs for which claims have been made that they learn, that is that their performance at given tasks improves over time. What improves and how it improves we will get to later.

First, for the hopefully few readers who are unfamiliar with Artificial Intelligence, it is the study of intelligent behaviour through computational modeling, usually but not necessarily, on a digital computer. The intelligence is often assumed to be that of a human, although other forms, below or even above, are not excluded. Some would like to draw a distinction between modeling the actions people do which require intelligence, and modeling the way people perform intelligent actions. We know so little about the latter that any model whose input/output behaviour resembles that of humans is as good a theory of human processes as anything else we may have, so we won't emphasize the distinction, except perhaps to point out that computer scientists make no claims to be qualified experimentalists.

In a few narrow areas such as playing checkers, analyzing certain classes of molecular spectra and performing symbolic integration, programs have been written to equal or better human performance. In other areas such as recognizing pictures and understanding natural language we are still far. And most importantly, no program has been constructed, or even suggested which achieves anything like the wide range of behaviour seen in humans.

The list of learning programs we will discuss is vaguely chronological, moderately representative and horrendously incomplete. If your personal favorite or even worse, your brainchild is missing, please accept our apologies.

For the purposes of the discussion that follows, a learning situation involves a teaching agent (the Teacher), and a learning agent (the Student). The Student is assumed to be an algorithm which exhibits some input-output behaviour and is an element of a collection of algorithms which constitutes the Learning Space. Learning itself is viewed as the recognition of failure by the

* * * * *

Student leading to its transformation to another algorithm in the Learning Space. Naturally the transformation of one algorithm to another must depend on the nature of the failure. The transformation process is called the Learning Strategy. Finally, the algorithm of the Learning Space which exhibits an ideal behaviour will be referred to as the Goal of a learning session.

Two main dimensions will be used to provide a simple classification of learning programs in AI. The first (the structural dimension) deals with the level of concepts in terms of which algorithms in the Learning Space are described. At one end of the scale the working hypothesis is: human intelligence resides mainly in the brain, which is made of neurons. Let us model neurons and their interconnections and use these modeling tools to describe algorithms in the Learning Space, including the Student and the Goal. Such modeling tools include concepts such as logical gates, electrical signals, threshold elements etc. We call this level neural. At the other end of the scale the hypothesis is: learning involves the acquisition of knowledge. Algorithms in the Learning Space are nothing but knowledge bases. Accordingly, the descriptive tools needed are formalisms for the representation of knowledge. We call this level epistemological and focus our discussion on one particular type of representation of knowledge called procedural (Winston (1975)). According to this representation, knowledge can be encoded usefully as procedures. Knowing an object is knowing how to recognize it, retrieve it, manipulate it, talk about it and so on. As an old sage once said: "If it waddles like a duck, quacks like a duck and otherwise behaves like a duck then you can't tell that it isn't a duck".

A third level of the dimension which should perhaps be viewed as falling between the two extremes described earlier, is the computational level. Here the algorithms of the Learning Space are viewed as computable functions describable in terms of one of the existing models of computation, e.g. Turing Machines, programs, phrase structure grammars, recursive functions, and the descriptive tools are defined accordingly. It must be emphasized here that the three levels can be shown to be in a certain sense equivalent as far as expressive power is concerned. Their difference lies more in the viewpoint they impose on the learning process, in much the same way different theories in Physics impose different viewpoints on the explanation of physical phenomena.

The second dimension, the learning process dimension, deals with the role of the Teacher in the learning process. At one end, the Teacher provides the Student with a complete description of the input-output behaviour of the Goal, which the Student then learns, directly or incrementally, without further assistance by the Teacher. We call this type of learning process learning by being told. A second possibility has the Teacher generating a sequence of sample inputs for the Student who is to respond to each one. The Teacher then judges the correctness of the responses and, if incorrect, the learning Strategy determines how the Student's behaviour should be transformed. This type of learning process we call learning from samples. In the third possibility, learning by discovery, there is no Teacher to guide the Student. The Student is responsible for his own goals.

Since the objective of the research we will be discussing is the design of programs that learn (i.e. the design of a Student), there is great difference in their complexity depending on whether we are talking about learning by being told, about learning from samples, or about learning by discovery.

The rest of this paper discusses several programs according to the nine-point space outlined above.

1. The Neural Level

We consider first the neural level of the structural dimension. Learning by being told here amounts to the problem of designing a neural net having a given input-output specification. The problem in its different facets has been treated by Bertouzos (1965) and Harrison (1965), among others, and is generally considered outside the scope of AI.

Learning from samples at the neural level has a long and illustrious history going as far back as computers themselves and featuring great successes as well as severe disappointments. Most of the work done here treats the Student as a classification device which accepts an input vector and classifies it to one of n pattern classes. The Teacher's job is to generate the samples and judge the Student's responses. The Student's job is to perform the classification and realize the learning strategy. Perhaps the best known and most successful learning program here is Samuel's Checker Player (1963). This is the first, and probably the only, program to achieve championship rating playing a game with a practically infinite game tree. Roughly, Samuel's program uses Shannon's minimax strategy and the evaluation of board positions is carried out by a simple linear threshold function whose inputs measure piece advantage, forks, etc. Learning is done by maintaining correlations of the contributions of different factors to the eventual success of a move. There is also provision in the Checker Player for changing the set of input factors used. The program can play against itself or against book games. Attempts have been made to adapt Samuel's strategy to other games such as chess without any success.

The Learning Space used for much of the research in this area includes only recognition algorithms describable in terms of a single threshold element whose outputs are weighted products of its inputs (usually assumed to take only 0 and 1 as values). Such recognition devices usually called perceptrons were first proposed by Rosenblatt (1959, 1962) (who used them to do visual pattern recognition), although earlier work had already established their merits (see Minsky and Papert (1969) for a discussion). These merits include simplicity, intuitive appeal and a simple learning strategy which can be shown to converge to a goal under very general conditions.

The image of perceptrons as simple yet powerful recognition devices which could recognize anything was shattered by the work of Minsky and Papert (1969). In one of the first treatises in the theory of computation, they proved a number of interesting results about perceptrons which illustrated convincingly their descriptive inadequacy as recognition devices. Among their results is a proof that the predicate PARITY, which is 1 if an even number of its inputs is 1 and 0 otherwise can only be

computed by a perceptron that requires a number of combinations of inputs that depends exponentially on the number of inputs. Moreover, this perceptron requires weights which grow exponentially with the size of the input set. Similar results were shown for other predicates such as CONNECTED which is 1 for any input digitized picture whose white (1) points are topologically connected. Considering the simplicity of these predicates when implemented in terms of a sequential program, it became obvious that for many pattern classes/predicates that need to be learned, the learning space is best described in terms other than those offered by perceptrons.

As mentioned earlier, perceptrons are only the simplest devices describable in neural terms. A larger class of recognition devices is describable if one allows two or more levels of threshold elements with the outputs of the i th level serving as inputs for the $(i+1)$ st. Such devices make it easier to realize predicates such as PARITY and CONNECTED but, unfortunately, no learning strategy is known whose convergence is guaranteed as was the case with perceptrons. Thus, additional complexity in the structure of the recognition devices may increase their power, but also limits our ability to study them and make general claims about their learning behaviour.

Work on learning from samples at the neural level has been traditionally classified under the heading of Pattern Recognition. Some representative examples of work in this area are described in Nilsson (1965), Arkadev and Braverman (1967) and Luda and Hart (1973), while Sampson (1976) discusses the relevance of such work to Biology and Neurophysiology.

Turning to learning by discovery at the neural level, the problem here can be reduced to the design of a program which accepts input samples, notes similarities and differences between them and proposes a set of pattern classes on the basis of its observations. This type of problem, often called "unsupervised learning", is treated in Luda and Hart (1973), among others. Holland's (1970) work on Adaptive Systems, uses quite literally the concepts of selection, reproduction, mutation, and cross-over drawn from genetics, and applies them to populations of programs in an environment. It is important to note that unsupervised learning has generally been most successful in situations where humans seem to fail because of lack of intuition.

2. The Computational Level

We move now to the computational level of the structural dimension. Learning by being told at this level involves the design of an algorithm (Turing machine, program etc.) from a specification of its input-output behaviour. When the input-output specification is given in terms of logical formulas, the problem is sometimes called Automatic Programming. Manna and Waldinger (1971) and Darlington (1976) describe representative approaches to the solution of this problem.

In the same category but with very different objectives are Newell's (1973) production system models of the behaviour of subjects in Sternberg's classic memory experiments. Sternberg showed his subjects a sequence of digits and measured the reaction time of the subjects in determining whether a probe

digit was in the list they had seen. Although not using a neural model of memory, Newell was trying to determine necessary conditions on the architecture of the brain which would explain the subjects reaction times.

Among the earliest attempts at learning at the computational level is Fogel's (1966) work on finite state machines (a restricted type of Turing machine) which are trained to predict the next element of a sequence of symbols by noting regularities in the input string observed so far. Fogel's experiments with machines involving a few states only have been quite successful in predicting cyclic environments and stationary time series, especially in comparison with humans. However, his results are not practically extendable to programs with large numbers of states.

Learning from samples at the computational level involves the development of algorithms which learn phrase structure grammars, Turing machines, or simple programs from samples. The problem of learning a grammar (known as Grammatical Inference) was first tackled by Gold (1967) who proved that even in situations where the learning space is rather limited, there may be no convergence of the learning strategy if only positive samples (i.e. samples generated by the goal) are presented to the student. Considerable work followed Gold's original results (see Eitermann and Feldman (1971) for a survey), but it is generally acknowledged today that this area of research is rather limited in its applicability by the descriptive adequacy of phrase structure grammars. Most learning situations are simply not readily expressible in terms of the framework offered by Grammatical inference.

The problem of learning a program has been tackled by Eitermann and Krishnaswamy (1974) and Bauer (1978), among others. The student is presented here with sample computations of the goal and the objective is to design a learning strategy which will eventually converge to a program that behaves in a manner that is consistent with all the sample computations. Thus, instead of presenting to the student only input-output associations, e.g. 2,3 associated with 6 if the goal is the multiplication program, we present it with traces of how the goal computes 6 when presented with 2 and 3 as inputs. Bauer's learning strategy can be shown to diverge in certain cases. However, it can also be shown that any program learned by this strategy will perform all sample computations correctly. Moreover, for any goal there is a particular sequence of sample computations which will direct the learning strategy to the goal. Thus, even though bad sequences of sample computations may lead the student astray, we at least know that there always exist good sequences that will teach it just the right program.

With the same psychological concern as Newell, Feigenbaum (1963) attempted in his EPAM program to model human behaviour in the memorization of lists of nonsense syllables. The program can handle two types of inputs - paired associations and serial anticipation. The program's memory is in the form of a discrimination net where just enough information is maintained at any time to distinguish one syllable from another, and to retrieve the second member of a pair when presented with the first. The cues used in the discrimination net are visual - the

position of a letter is important, as are the strokes of which it is composed. The program exhibits behavior comparable to what psychologists have called oscillation - the temporary forgetting of an item, following the introduction of similar items - and retroactive inhibition - the forgetting of elements of one list during the learning of the next.

3. The Epistemological Level

We turn now to the epistemological level of the structural dimension. As mentioned earlier, the objective of learning here is a knowledge base, where knowledge is represented procedurally. Learning by being told is an important problem here treated under the label of Knowledge Acquisition (see Proceedings of IJCAI-77 (1977) for several related papers). Stated simply, the problem is the development of techniques for augmenting a knowledge base. The difficulty of the problem is due to the requirement that the knowledge base remain at all times internally consistent and compatible with the organizational principles dictated by the representation being used. Since large knowledge bases are essential to the design of any kind of "intelligent" system, much effort within AI today is devoted to finding solutions to their incremental creation. An excellent example of early research on this problem is Winograd's SHRDLU program (Winograd (1972)). The program functions in a domain which has become known as the Blocks World: a table with a collection of colored cubes and pyramids, a box, and a hand to move the blocks. SHRDLU accepts English sentences as input, executes imperatives such as "Pick up the red block under the green pyramid and put it in the box", and answers questions such as "Where is the green cube?", and "Why did you move the blue pyramid?". In addition, however, the program can be told the description of new objects. For example, following the statement "A steeple is a pyramid on top of a cube", SHRDLU can respond to "Make a steeple", "Is there a steeple in the scene?" and so on.

Like SHRDLU, HACKER (Sussman (1975)) is a problem-solver in the Blocks World. However, it solves a problem by constructing a program called a plan, which is a sketch of the solution. HACKER can observe the execution of the plans it constructs, detect some errors, and transform them to avoid the errors. A correct solution to a problem can be stored away and re-used in finding a solution to a more complex one.

One of the few examples of a program that learns from samples at the epistemological level is Winston's program (1975) for recognizing visual scenes in the Blocks World. The program begins with a knowledge base of descriptions for concepts such as tower, steeple, etc. The program can learn descriptions of new concepts by being shown positive and negative instances of the concepts.

The AM program by Lenat (1977) is the first to exhibit high performance in learning by discovery at the epistemological level. AM starts from simple concepts in set theory and conjectures statements in set and number theory. It tests these statements against the facts at its disposal and, most importantly, evaluates the degree to which it believes they support other conjectures, or might lead to new ones. Lenat's

program cannot properly be called a learning program since it does not attempt to formally prove the generated conjectures to distinguish theorems from non-theorems. However, if augmented by a powerful theorem-prover, and some candidates do exist, it could become the learning program created so far in AI.

4. Conclusion

In an earlier draft of this paper written in 1976 we wrote "To the best of our knowledge there are no AI programs currently being developed which their authors would claim learn anything. Why? ... We believe that the most pressing issues in AI today are those of representation and control of co-operative processes. If we can come up with workable solutions to these problems, then we can reconsider the problem of learning ..." We are happy to report that the situation has changed, as evidenced by the works of Lenat (1977), Davis (1977), Buchanan and Mitchell (1977). We believe that this change has come about precisely because some workable solutions are being found to the problems of representing and controlling co-operative processes. We also believe that learning will become a much more important research area within AI in the future as problems of representation are further explored and understood. As the reader may have already inferred from the discussion, we view the epistemological level as the one where learning can be tackled more generally and, ultimately, more usefully.

References

1. Arkadev, A. and Fraerman, F. Computers and Pattern Recognition. Thompson Book Co., 1967.
2. Bauer M. A Basis for the Acquisition of Procedures. Ph.D. Thesis, Dept. of Computer Science, University of Toronto, 1978.
3. Biermann, A. and Feldman, J.A. "A Survey of Results in Grammatical Inference" in Proceedings of Conference on Frontiers in Pattern Recognition, Honolulu, 1971.
4. Biermann, A. and Krishnaswamy, R. "Constructing Programs from Example Computations". Technical Report CSU-CISFC-TS-74-5, Computer and Information Science Research Center, Ohio State University, 1974.
5. Buchanan, B.G. and Mitchell, T.M. "Model-Directed Learning of Production Rules". In Proceedings of Workshop on Pattern-directed Inference Systems, Honolulu, 1977.
6. Darlington, J.I. "Automatic Synthesis of SNOBCI Programs". NATO Advanced Study Institutes Series, Series E: Applied Science, 14:515, Leyden: Noordhoff, 1976.
7. Davis, E. "Interactive Transfer of Expertise: Acquisition of New Inference Rules". Proceedings of Fifth International Joint Conference on Artificial Intelligence, Cambridge Mass., 321-328, 1977.
8. Dertouzos, M.I. Threshold Logic, Cambridge: MIT Press, 1965.
9. Duda, R.C. and Hart, P. Pattern Classification and Scene Analysis, New York: Wiley Interscience, 1973.

10. Feigenbaum, E.A. "The Simulation of Verbal Learning Behavior". In Feigenbaum E.A. and Feldman J. (eds.) Computers and Thought, McGraw-Hill, 1973.
11. Fogel, I.J., Owens, A.J., and Walsh, M.J. Artificial Intelligence through Simulated Evolution, New York: Wiley, 1968.
12. Gold, M. "Language Identification in the Limit". Information and Control 10, 447-474, 1967.
13. Harrison, M. Introduction to Switching and Automata Theory, McGraw-Hill, 1965.
14. Hollard, J.E. "Outline for a Logical Theory of Adaptive Systems". In Burks, A.W. (ed.) Essays in Cellular Automata, Champaign-Urbana: University of Illinois Press, 1970.
15. Lenat, D.F. "Automated Theory Formation in Mathematics". Proceedings of Fifth International Joint Conference on Artificial Intelligence, Cambridge Mass., 833-842, 1977.
16. Manna, Z. and Waldinger, F. "Toward Automatic Program Synthesis", Communications of ACM, 14, 151-165, 1971.
17. Minsky, M. and Papert, S. Perceptrons: An Introduction to Computational Geometry. Cambridge: MIT Press, 1969.
18. Newell, A. "Production Systems: Models of Control Structures". In Chase, W.G. (ed.) Visual Information Processing, New York: Academic Press, 1973.
19. Nilsson, N. Learning Machines. New York: McGraw-Hill, 1965.
20. Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Cambridge Mass., 1977.
21. Rochester, N., Holland, J.H., Haibt, I.E., and Luda, E.C. "Test on a Cell Assembly Theory of the Action of the Brain Using a Large Digital Computer", Inst. of Radio Engineers, Trans. on Inf. Theory, 2, 80-93, 1956.
22. Posenblatt, F. "Two Theorems of Statistical Separability in the Perceptron". Proceedings of a Symposium on the Mechanization of Thought Processes, London, 421-456, 1959.
23. Posenblatt, F. Principles of Neurodynamics. New York: Spartan Books, 1962.
24. Sampson, J.F. Adaptive Information Processing: An Introductory Survey, New York: Springer Verlag, 1976.
25. Samuel, A.I. "Some Studies in Machine Learning Using the Game of Checkers II: Recent Progress". IBM J. of Res. and Dev. 11, 6, 1967.
26. Sussman, G.J. A Computational Model of Skill Acquisition. New York: American Elsevier, 1975.
27. Winograd, T. Understanding Natural Language. New York: Academic Press, 1972.
28. Winograd, T. "Frame Representations and the Declarative-Procedural Controversy". In Echrow, D. and Collins, A. (eds.) Representation and Understanding, New York: Academic Press, 1975.
29. Winston, P.H. "Learning Structural Descriptions by Examples". In Winston, P.H. (ed.) The Psychology of Computer Vision, New York: McGraw-Hill, 1975.

OPEN DISCUSSION

William Wimsatt, University of Chicago: Could you give us an example of how your approach applies to evolution?

Perrault: Okay. I'll try to give you a simple example of a situation in which these systems have been tried. The adaptive systems are the closest to the literal interpretation of evolution in the genetic sense. In most of these cases the representation being used is something like a vector of parameters, for example, being modified. The idea is that one has populations of programs competing in an environment being evaluated by some externally provided payoff function and mutation is simulated by randomly altering a parameter in the vector. Cross-over can be done by taking two parameters and chopping pieces out of one and trying to put them in the other. So we get different ways of generating processes from other processes, sending them off into the world, and usually reproducing them at a rate proportional to their success in the environment. That's the general structure of the algorithms. They seem to be most applicable to knowledge representations which look as much as possible like a chromosome. There usually has to be something like a sequence of fairly uniform-looking objects. Certainly, if you're going to try to do things like cross-over and mutation, it has to be possible to look at comparable pieces in two different programs and make up a new program out of it. It's not at all clear how one does this when you're talking about crossing two Fortran programs, for example.

Wimsatt: So programs have a great deal of looping and going from one jump to another.

Perrault: In a sense we may have to distinguish between this set of parameters and the program which generates the behavior using these parameters as part of its structure. Okay? There may be a program that has loops. That program is constant except that the set of parameters may change so that the recognition proceeds. For example, think of it as a Fortran program and twiddle its parameters. The modifications are made on the parameters but the overall structure of the algorithm remains the same. Okay?

John Maynard Smith, University of Sussex: Adaptive algorithms may be good for a wide range of complex tasks, but as a device for finding the best fit for some set of data or something, how do they actually compare in efficiency with things that have just been brought up by computer scientists for getting a good fit?

Perrault: I really can't give you a good answer to that. I just don't know that literature well enough and I also don't know the optimization literature in other aspects of computer science to be able to make good comparisons between the two of them. Off-hand, I would think that convergence rates obtained by specific methods developed by computer scientists to attack specific problems are probably graded. Of course the whole point of the adaptive algorithm is that it should be usable in a large class of problems. It is an attempt at generality.

Murray Eden, National Institutes of Health: In the area of character recognition, which is an area that has been extensively explored in computer science, there is essentially by now a general belief, and I think it's a correct belief, that almost any college freshman who knows a little programming can program a computer so that it recognizes characters with a recognition rate of the order of 60 or 70% correct. No perceptron or perceptron-like device within the environment in which the student wrote his program could do remotely as well. In a twenty-six character alphabet perceptrons barely go above chance level unless they have a lot of structure built in them. The convergence rate is much too slow whereas, as I say, just anybody who has the rudiments of the rules of the game can do a respectable job.

Perrault: By building a much more specific program than perceptrons and one which has a lot more structure to it.