The Child Machine vs The World Brain

Claude Sammut
School of Computer Science and Engineering
The University of New South Wales
Sydney, Australia
e-mail: claude@cse.unsw.edu.au

ABSTRACT

I think of machine learning research as building two different types of entities: Turing's Child Machine and H.G. Wells' World Brain. The former is a machine that learns incrementally by receiving instruction from a trainer or by its own trial-and-error. The latter is a permanent repository that makes all human knowledge accessible to anyone in the world. While machine learning began following the Child Machine model, recent research has been more focussed on "organising the world's information". Both are important endeavours, however, incremental learning has been neglected and, we argue, should be revived.

1 INTRODUCTION

In his *Mind* paper, Alan Turing (1950) famously described the imitation game and he also suggested that to build a computer system capable of achieving the required intelligence, it would have to educated, much like a human child.

> Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain. Presumably the child-brain is something like a note-book as one buys from the stationers. Rather little mechanism, and lots of blank sheets... Our hope is that there is so little mechanism in the child-brain that something like it can be easily programmed. The amount of work in the education we can assume, as a first approximation, to be much the same as for the human child.

He went on to speculate about the kinds of learning mechanisms needed for the child machine's training. The style of learning was always incremental. That is, the machine acquires knowledge by being told or by its own exploration and this knowledge accumulates so that it can learn increasingly complex concepts and solve increasingly complex problems.

Early efforts in Machine Learning adopted this paradigm. For example, Michie and Chambers (1968) BOXES program learned to balance a pole and cart system by trial-and-error receiving punishments are rewards, much as Turing described, and like subsequent reinforcement learning systems. My own efforts, much later, with the Marvin program (1981b) were directed towards building a system that could accumulate learn and accumulate concepts expressed in a form of first order logic.

At around the same time of Turing's paper on computing machines (Turing, 1936), H.G. Wells speculated on what he called "the world brain" (Wells, 1937), a permanent world encyclopædia that would provide,

> "...a sort of mental clearing house for the mind, a depot where knowledge and ideas are received, sorted, summarised, digested, clarified and compared.... any student, in any part of the world, will be able to sit with his projector in his own study at his or her convenience to examine any book, any document, in an exact replica."

Wells thought that the technology for this repository of knowledge would be microfilm. He could not have known that the World Wide Web, Wikipedia and internet search engines would do far more than he envisaged. Indeed, Google's company mission is "… to organize the world's information and make it universally accessible and useful"[1]. A core technology for such search engines is machine learning. However, it is of a form that is somewhat different form that described by Turing for his Child Machine. Unlike a human child, acquiring new knowledge incrementally, machine learning systems can access the enormous amounts of data available throughout the internet to produce, in some instances, superhuman performance. However, this is "all-at-once" and "single-concept-at-a-time" learning that must still be guided by humans.

The first programs capable of efficiently learning from moderately large numbers of examples began with Michalski's Aq (Michalski, 1973) and Quinlan's ID3 (Quinlan, 1979), both of whom were influenced by Donald Michie. Quinlan was a PhD student studying with psychologist Earl Hunt when he attended a lecture by Michie at Stanford University. Donald challenged the

[1] http://www.google.com/about/company/

students to devise a method of learning to determine a win in a chess end-game and Quinlan responded with the first version of his decision tree learner. The utility of these programs for finding patterns in large data sets encouraged new research into "batch" learning systems, somewhat to the detriment of incremental learning.

However, for machines to behave in ways that we can truly say are intelligent, they must be able to accumulate knowledge just as Turing suggested for the Child Machine. Unfortunately, apart from a few systems, like Marvin, very little research has been devoted to this style of learning. It is useful to be reminded of some important contributions from the past that deserve to be revived. We remind the reader of these in the next section.

2 CUMULATIVE ACTIVE LEARNING

A necessary component of the Child Machine's learning is interaction with the world, which may be physical or virtual, if the agent is embedded in the web. As an agent accumulates experience, it constructs a world model or theory that can be used to predict the outcome of events and to determine actions necessary to solve problems.

2.1 Background Knowledge

The first requirement for cumulative learning is a mechanism for storing and using learned concepts as background knowledge. Representing concepts in the form of expressions in first order logic makes this relatively easy. To my knowledge, Banerji (1964) was the first propose such a mechanism and the first implementation of a learning system that could use learned concepts for further learning was by Cohen (1978). With Brian Cohen, I elaborated this idea to create a learning system in which the concepts were represented as logic programs and were, thus, executable (Sammut & Cohen, 1980). This then evolved into Marvin (Sammut, 1981a) and the use of background knowledge has become an essential part of Inductive Logic Programming (Muggleton, 1991) since. However, even many ILP that use background knowledge to learn new concepts do not close the loop to allow those learned concepts to, themselves, become part of the background knowledge for future learning.

2.2 Detecting Errors

Since, in practice, a learning agent can never be exposed to all instances of a concept, it is prone to making mistakes of two kinds:

- The system may observe an event for which there is no prior knowledge about how to respond.
- The system may observe an event for which there is prior knowledge. However, the known concepts are too general and the system may respond inappropriately.

When the system is learning only one concept at a time, repairing a theory is relatively easy. We just specialise an

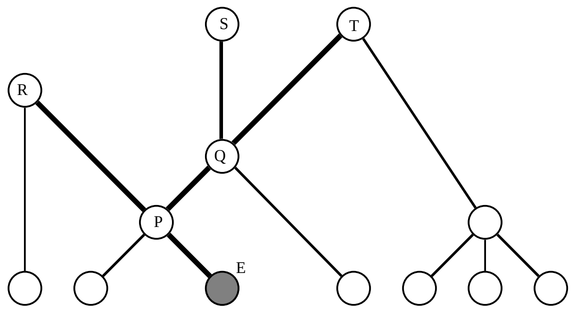

Figure 1: *A theory is a network of concepts*

over-general theory or generalise a theory that is too specific. However, in a system that accumulates knowledge over time, learning many concepts, localising the error is not so easy. We can think of a theory as being a network of interdependent concepts, as in Figure 1, where the definition of concepts *S* and *T* rely on the definition of *Q*, which depends on *P* and *E*. Suppose we discover an error in the theory while attempting to use concept, *T*, but the real error is due to an incorrect definition of concept, *E*.

When we use a logical representation of concepts, a concept description is also an executable computer program, so one way of locating the problem is to trace through the execution of the program that lead to the unexpected result, testing each concept. That is, we debug the program, as Shapiro (1981) did with MIS. To locate an error when an incorrect solution has been given (i.e. the theory contains an over-generalisation) Shapiro's debugging algorithm works backwards through the failed proof of a goal, searching for the procedure that caused the failure. In Figure 1, backtracing would begin with the last goal satisfied, that is, *T*. The debugger begins stepping back through the proof, i.e. down the dark path to node *Q*, then *P* if necessary, asking an oracle if the partial solution at each point is correct. If this is not true, then an erroneous clause has been found. Note that the algorithm assumes the existence of an infallible oracle. In a reactive environment, the learning program may do without an oracle since the program is able to perform experiments to test a concept. Thus, a failure suggests that the initial set of experiments that resulted in the formation of the concepts along the solution path was not extensive enough for at least one of the concepts.

A concept that is too specific may prevent the program from producing any solution at all. That is, the logic program that is supposed to satisfy the goal does not cover the initial conditions of the task. An attempt at debugging the theory can only be made when a correct solution has been seen, otherwise the learner has no indication that the task really is possible. A correct solution may be found, either by "mutating" the current theory in the hope that the goal can be satisfied by the mutant or the learner may observe another agent in the world performing the task. Shapiro's debugging method for programs that fail to produce an answer is equivalent to the second alternative, that is, the oracle supplies the correct solution. The debugger again tries to work backwards seeking clauses in the program which could have produced the given solution. Once such a clause is found, its body provides further goals that should be satisfied in order to arrive at the solution. The debugger considers each of these intermediate goals to see if they can also be produced by other clauses. Any goal that cannot be achieved indicates where the program or theory is deficient.

2.3 Correcting Errors and Maintaining Consistency

Detecting and repairing an error in a single concept is one thing, but repairing an entire theory is another matter. Remember that in figure 1, we envisaged a world model or domain theory as a network of interconnected concepts. Using a horn clause representation, the head of a clause corresponds to a parent node and the goals in the body correspond to the children. These goals match other clause heads and form links to the rest of the network. Also in Figure 1, we imagined that one concept, represented by the shaded node, *E,* was in error. When the concept is repaired, what effect will that have on the concepts which referred to the old concept? Since *P, Q, R, S* and *T* refer, directly or indirectly, to the erroneous node they must have been learned in the presence of the error. Are they, therefore also in error or will correcting *E* alone correct them all?

When faced with the problem of ensuring the consistency of its knowledge base, two strategies are available to the learning system.

1. After correcting E, the system may test each of the concepts that depend on E. However revising all of the concepts dependent on one that just been modified could involve a lot of work if the network of concepts is very extensive.
2. The system may wait to see if any further errors show up. In this case, each concept will be debugged as necessary. Although more economical this method requires a method for tolerating errors if the program has been assigned a task which it must continue to perform.

It should also be noted that another source of errors in planning is noise. When a learning system is connected to the physical world, as is a robot, it cannot rely on the accuracy of measurements from vision systems, touch sensors, *etc.* Thus, a program may fail because the knowledge base does not accurately reflect the outside world. This being the case, the learning system must not revise its domain theory prematurely since there may not, in fact, be any errors. Therefore, the most prudent approach to error recovery is to delay revision of a domain theory until sufficient evidence has accumulated to suggest the appropriate changes.

Let us now give an outline of an error recovery strategy.

1. The robot learner is given a task that it is required to perform. However, its domain theory may be incomplete or incorrect.
2. In the course of performing its task, the robot's plan fails.
3. If the robot is unable to proceed by adopting another plan then it must suspend working on its given task while it debugs its domain theory.
4. If an alternative plan is possible (for example, by reordering goals) then the new plan is attempted while storing the failed plan for future reference.
5. The robot cannot assume that the failed plan is incorrect since the cause of failure may have been due to noise, therefore, as each plan is executed, a history of its performance is maintained, this includes the performance of the individual concepts which formed the plan.
6. The accumulation of histories is input for a theory revision system that effectively summarises the performance of plans when it forms new concepts by generalisation.
7. Since the learning program may generate alternative descriptions for the same concept, we must be able to resolve potential conflicts so that the next time a similar plan is to be created, the appropriate description is chosen.

Richards and Mooney (1991) proposed a system that does that, based on earlier work by Muggleton (Muggleton, 1987; Muggleton & Buntine, 1988). Wrobel (1996) also proposed a first order theory refinement systems The main idea behind these methods is to consider repairing a theory as a kind of compression. That is, the set of clauses that form a theory may be rewritten so that the same or greater coverage is achieved by theory that is more compact in an information theoretic sense. An important operation for this is *predicate invention* (Muggleton & Buntine, 1988). That is, the learner must have the ability to invent its own concepts when it detects patterns that can be reused in the description of other concepts.

When an experiment does fail, we must not invalidate the concepts used in planning the experiment for, as mentioned earlier, the failure may be due to noise. Instead, we note the circumstances of the failure and augment the failed concept with a description of these circumstances. Several things could happen to the concept when this is done:

- The description of the concept is modified to the extent that it becomes correct. If an alternative, correct description already existed, then the alternative domain theories of which these concepts were components, converge.
- After several failures, there is no generalisation which covers the circumstances of failure. In this case, the failures may be either due attributed to noise or to some phenomenon not yet known to the system. In either case, nothing can be done.

An ATMS (de Kleer, 1986) maintains the network of concepts which form a domain theory and stores dependencies which, when errors are found will indicate where other potential weaknesses in the theory lie. The ATMS also allows a learning program to experiment with alternative domain theories.

3 SCALING UP

All of the references in the previous section are quite dated. The fact is, that while the methods described have the potential to build a very powerful learning system, they have not, so far, scaled to work with large amounts of data or operate in complex environments. Whereas, simple learning systems have done spectacularly well on “Big Data”. Indeed, search engine companies, like Google, have so much data and computing resources that many problems can be turned into machine learning tasks. So far, then, machine learning has facilitated the World Brain but is struggling with the Child Machine. Why is that?

It seems to me that the problem is that we still do not have effective methods for building and maintaining theories that have a deep structure. Yet, even for the World Brain, such theories will be needed. As the quantity of information rises, we will need mechanisms for digesting and summarising data that will require methods such as predicate invention and theory revision. Thus, the title of this talk is not entirely accurate. I do not see the Child Machine in competition with the World Brain. Rather, the both will eventually be needed to make large knowledge bases manageable. So a serious challenge for the future is to make incremental learning scale up to be capable, not only of imitating the learning of a child, but to work with large-scale data mining to manage knowledge on the web.

References

Banerji, R.B. (1964). A Language for the Description of Concepts. *General Systems*, **9**, 135-141.

Cohen, B.L. (1978). *A Theory of Structural Concept Formation and Pattern Recognition*. Ph.D. Thesis Thesis. Department of Computer Science, University of New South Wales.

de Kleer, J. (1986). An Assumption-based TMS. *Artificial Intelligence*, **28**(1), 127--162.

Michalski, R.S. (1973). Discovering Classification Rules Using Variable Valued Logic System VL1. In *Third International Joint Conference on Artificial Intelligence* (pp. 162-172).

Michie, D., & Chambers, R.A. (1968). BOXES: An Experiment in Adaptive Control. In E. Dale & D. Michie (Eds.), *Machine Intelligence 2*. Edinburgh: Oliver and Boyd.

Muggleton, S. (1987). Duce, An oracle based approach to constructive induction. In *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 287-292). Milan.

Muggleton, S. (1991). Inductive Logic Programming. *New Generation Computing*, **8**, 295-318.

Muggleton, S., & Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In R.S. Michalski, T.M. Mitchell & J.G. Carbonell (Eds.), *Proceedings of the Fifth International Machine. Learning Conference* (pp. 339-352). Ann Arbor, Michigan: Morgan Kaufmann,.

Quinlan, J.R. (1979). Discovering rules by induction from large collections of examples. In D. Michie (Ed.), *Expert Systems in the Micro-Electronic Age*. Edinburgh.

Richards, B.L., & Mooney, R.J. (1991). First-Order Theory Revision. In *Proceedings of the Eighth International Machine Learning Workshop*, Evanston, IL. pp. pp. 447-451.

Sammut, C.A. (1981a). Concept Learning by Experiment. In *Seventh International Joint Conference on Artificial Intelligence*, Vancouver. pp. 104-105.

Sammut, C.A. (1981b). *Learning Concepts by Performing Experiments*. Ph.D. thesis Thesis. Department of Computer Science, University of New South Wales.

Sammut, C.A., & Cohen, B.L. (1980). A Language for Describing Concepts as Programs. In J.M. Tobias (Ed.), *Language Design and Programming Methodology* (Vol. 79, pp. 111-116): Springer.

Shapiro, E.Y. (1981). An Algorithm that Infers Theories from Facts. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver. pp. 446-451: Morgan Kaufmann.

Turing, A.M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, **2**(42), 230-265.

Turing, A.M. (1950). Computing Machinery and Intelligence. *Mind*, **59**(236), 433-460.

Wells, H.G. (1937). World Brain: The idea of a permanent world encyclopaedia, Encyclopédie Française.

Wrobel, S. (1996). First Order Theory Refinement. *Advances in Inductive Logic programming*, **32**, 14--33.