Applying Inductive Logic Programming in Reactive Environments

David Hume
Claude Sammut

School of Computer Science and Engineering
The University of New South Wales
Sydney, Australia 2052

ABSTRACT

We describe an application of inductive logic programming to the task of learning action sequences in a simulated robot world. A learning agent observes sequences of actions that may change the properties or relationships of objects in the world. The observed sequence is then used to form a theory which can be generalised and tested by imitation. That is, the learner attempts to perform its own sequence of actions. If the test completes as expected then the generalisation is accepted. However, if it fails then a *regeneralisation* occurs. That is, further generalisations are found to explain the difference between expectation and reality.

We illustrate some of the problems that are encountered by learning systems when the environment is complex and reactive by examining several examples.

The Problem

We are concerned with learning in a reactive environment. By this we mean that the learning agent acquires knowledge by interacting with its environment. Thus the learner and the world are both active. This contrasts with most common applications of inductive learning where data are supplied to a passive program. Learning in a reactive environment creates a number of problems that are not normally encountered in passive learning. These problems and some suggestions about how they can be overcome are the subjects of this paper.

It is useful to understand "active" learning because of its importance in human learning and also because of the potential practical applications. For example, the reactive environment could be a manufacturing plant in which an intelligent agent is required to learn to control some part of the process. The environment could also be a space craft or robot that must understand its environment in order to function correctly. Thus, if we do not wish learning systems to continue to be disembodied entities then we must attend to the difficulties of interacting with an external environment.

The first difficulty to be faced is that this kind of learning is often weakly directed because no trainer is providing goals for learning. For example, young children often play with objects left within their reach and just as often, there is no apparent purpose to their actions. This is a form of goal-less learning where data about the world are being collected for later use. Parents sometimes provide some direction to the learning process by performing some simple task in view of the child and the child may then attempt to imitate the actions. Without having a goal, it is difficult to see which objects and actions are connected to a particular task.

The second difficulty is that a reactive environment, including our simulated one, is never ideal. A child may observe a parent's actions and then try to imitate them. However, the parent has left the world in a state that differs from the initial state, thus the child's imitation may not succeed because the initial conditions are not correct, nor does the child know what those conditions should be. In addition, each experiment that the learning agent performs changes the world. If the result of the experiment is unexpected then it may be impossible to recover. So the learning agent in a reactive environment must be able to make do with what it finds.

The third difficulty is that hypotheses can never be proved correct. Since the intelligent agent learns by experimentation and it can never perform an exhaustive set of experiments, it may believe a hypothesis that is false. Apart from causing the agent to behave incorrectly, belief in a false hypothesis will result in difficulties when trying to repair the agent's world model.

In this paper, we describe a program called CAP that uses a weakly directed learning model to explore its environment. The representation language used is first order horn-clause logic. Being a general purpose representation language, we are able to insert CAP into a variety of different kinds of environments ranging from robot worlds to the worlds of numbers and lists. The induction mechanism used by CAP is based on inverse resolution (Muggleton, 1988; Muggleton and Buntine, 1988) and on an earlier program, MARVIN (Sammut, 1981; Sammut and Banerji, 1986).

First we will give a brief overview of the CAP program and then we will illustrate the difficulties mentioned above with examples of learning tasks for CAP.

Overview of CAP

If learning in a reactive environment were totally undirected then the intelligent agent could perform an almost infinite variety of experiments to try to deduce the structure of the world from their outcomes. This could be a little time consuming, so to provide some constraints on exploration, learning is only initiated when another agent, presumed to be knowledgeable, performs some sequence of actions.

One way of characterising the motivation for CAP's learning algorithm is to think of it as trying to produce a theory that will enable it to recognise each sequence of actions it sees. A naive way of accomplishing this is to simply store all observed sequences. Obviously this is wasteful of space and the likelihood of recognising a new action sequence is low since only a sequence identical to one stored could be recognised. So CAP tries to generalise from its observations. Thus, having created an initial theory we proceed to test it and then generalise it.

Testing a theory is relatively straightforward. A theory contains actions and the world states expected to be derived from those actions. Thus, if the program attempts to perform those actions and the resulting state is *not* consistent with the final state described in the theory then the theory has been disproved. When such a failure occurs, CAP attempts to generalise the theory to account for the world as it *is* rather than as it was originally *expected* to be. Thus, it can learn from its mistakes. We will refer to this process as *theory adjustment*. Theory adjustment gives rise to unplanned generalisations, so called because they are done in order to correct a theory that failed in an unforeseen way. *Planned* generalisations occur after a theory has been tested and the experiment concludes successfully. Although not conclusive, this encourages CAP to accept the theory and generalise it further.

The description of one state of the world is extremely complex in anything other than a toy example. The larger the description, the more ways there are to generalise it. A radical generalisation would attempt to change many terms in the theory. Thus, if a failure occurred, it would be very difficult to isolate the cause of the failure so that it could be rectified by an adjustment to the theory. Therefore, CAP uses a very conservative form of generalisation. The program can be said to escalate by stages or *levels* to attempt progressively more complex generalisations. CAP terminates when it can perform no generalisations on any of its theories such that the theories remain consistent with the world.

One of CAP's distinguishing features is that it is able develop several theories concurrently. This is not only a good idea, it is necessary. Often, the program will be unable to continue testing a particular theory because the world does not contain the material required for further experimentation. So having several learning tasks running concurrently allows the program to continue operation. Sometimes, working on another problem will change the world in such as way as will enable experimentation to revert to tasks that had to be stopped previously.

In the next two sections we will discuss two aspects of CAP in more detail: how initial theories are constructed from observations and the global strategy for performing generalisations.

Constructing Initial Theories from Observations

Suppose another agent in the world has constructed an arch (yet again!). CAP would try to construct a theory that describes the preconditions and post-conditions of each action in the construction sequence. Sequences are represented by expressions in first order logic that indicate the states of objects in discrete time instants and the actions that transform one state into another. We represent the sequence as follows:

$$S_0 \xrightarrow{A_{0\to1}} S_1 \ldots S_{n-1} \xrightarrow{A_{n-1\to n}} S_n$$

Literals in the representation language are annotated with time stamps. Thus we could state that P is true at time t with the expression: *P at t*. We can also state that an action has certain duration from a start time to a finish time: *A during Start/Finish*.

Given a sequence of actions applied to states in the world, what sort of theory can we devise? CAP behaves rather like a frog trying to catch a fly. The frog's attention is attracted by movement. CAP's attention is attracted by change. Let us take arch building as an example. During the construction process, our attention will be focussed at different times on different objects. While constructing the columns, we will move one block at a time onto a column. While moving one block certain actions will be performed repeatedly. For example, to lift a block to a certain height requires us to repeat the lifting action for several time units before changing to a lateral movement to locate the block over the column. These changes in operation lead us to the following heuristics:

- Divide the original sequence into sub-sequences when the objects being affected change. For example, transfer one block to a column and then find another block.
- Divide the sub-sequences into sub-sub-sequences when actions on the same objects change. E.g. perform a sequence of lifting actions on one block and then switch to lateral movement of the same block.

For each sub-sub-sequence, $SSS_{i,j}$, a concept $H_{i,j} \Leftarrow B_{i,j}$ is created as follows:

- $H_{i,j} = P(t_1, .., t_n)$ *during* TB/TE is created by inventing a unique predicate symbol, P. The arguments $t_1, .., t_n$ are all of the objects occurring in $SSS_{i,j}$. TB is the earliest time stamp appearing in the sub-sub-sequence and TE is the latest.
- The body of the clause, $B_{i,j}$ is simply $SSS_{i,j}$.

For each sub-sequence, SS_i, a concept $H_i \Leftarrow B_i$ is created as follows:

- The head is obtained as above except that instead of deriving the arguments and times from primitive action predicates, we use the heads of the clauses, $H_{i,j}$ instead.
- Similarly, the body B_i is obtained from the conjunction of the literals $H_{i,j}$.

Finally, the initial theory $H \Leftarrow B$ is obtained by constructing a body from the conjunction of literals, H_i and the head is obtained before by extracting all of the objects referred to in the body and the earliest and latest time stamps.

If the same primitive action is performed repeatedly on the same object in succession then a recursive disjunction of clauses is created to replace the single non-recursive clause that would have been created otherwise. The result should be more compact and more general.

If the sequence, R consists of the states $S_0,..,S_n$ and primitive actions $A_{0\longrightarrow 1},..,A_{n\text{-}1\longrightarrow n}$, corresponding to n same actions on the same objects in succession, then the $n\text{-}1$ valid concepts are:

$RSA_{0\longrightarrow n} \Leftarrow S_0, A_{0\longrightarrow 1}, S1, ..., A_{n\text{-}1\longrightarrow n}, S_n$

.....

$RSA_{j\longrightarrow n} \Leftarrow S_j, A_{j\longrightarrow j+1}, S_{j+1}, ..., A_{n\text{-}1\longrightarrow n}, S_n$

.....

$RSA_{n\text{-}1\longrightarrow n} \Leftarrow S_{n\text{-}1}, A_{n\text{-}1\longrightarrow n}, S_n$

These clauses become examples for an induction phase in which they are reduced to a simple recursive concept. Finally, all constants are replaced by variables. Thus, a theory consists of a hierarchy of concepts as depicted in Figure 1.

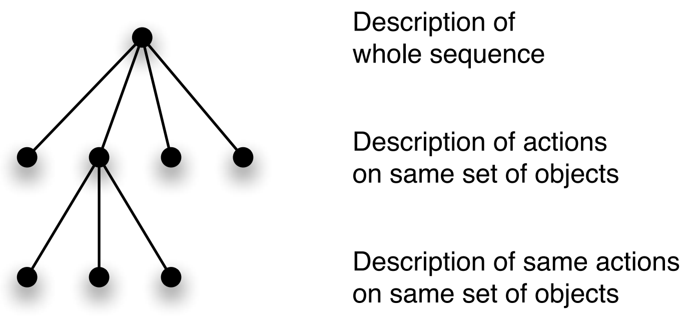

Description of whole sequence
Description of actions on same set of objects
Description of same actions on same set of objects

Figure 1: A theory consists of a hierarchy of concepts.

Before beginning to test the new theory, CAP performs one final procedure. Because state descriptions can become very complex it is useful to trim out irrelevant information. For example, when moving blocks, the spatial relationship between the moved block and *all* others changes but it is only the relationship to the column that is important. So we adopt the following heuristic to remove irrelevant terms from the theory. The program selects those predicates that have changed from one state to the next. The objects referred to in those predicates are sorted by frequency of occurrence and only those predicates which refer to the most frequently appearing objects are retained. The present implementation uses only objects with the top two frequency counts but a more general implementation will progressively loosen this condition in the search for a theory that will consistently describe observations.

Global Learning Strategy

As we mentioned previously, CAP uses a very conservative method for generalising its theories. Since the representation language is Horn-clause logic, we use inverse resolution to generalise clauses. The main generalisation operators are *absorption* and *intra-construction* with some modifications necessary for operation in the reactive environment. Absorption generalises expressions using only background knowledge while intra-construction is able to invent new terms in the description language, thus expanding its ability to describe novel situations. Details of these operations are given by Hume (1991) and the original theory behind them is described by Muggleton and Buntine (1988). In this paper we will restrict our discussion to the global learning strategy rather than go into details of the generalisation operators.

Each theory is assigned a generalisation level. Beginning at the lowest level, CAP performs experiments to test the theory in its current state and then proceeds to more and more general theories as long as they are consistent with the world. The generalisation levels are described below.

Level 1 At this level we simply try to repeat the example sequence. In fact, this is testing a generalisation since, when the observed sequence became the seed for the initial theory, constants were replaced by variables, irrelevant terms were removed and recursive expressions were introduced to replace sequences where possible.

Level 2 Here we try to generalise the primitive action predicates (e.g. move, pickup, etc.). There are two internal levels of generalisation. First we attempt absorption, i.e. trying to use concept we already know about and then we try intra-construction, i.e. inventing new concepts.

Level 3 At the next level we attempt to generalise sub-sub-sequences, i.e. sequences of the same action on the same objects. Again there are two internal levels of generalisation: absorption and intra-construction.

Level 4 At the highest level we attempt to generalise sub-sequences, i.e. sequences of different action on the same objects. The same two internal levels of generalisation exists here.

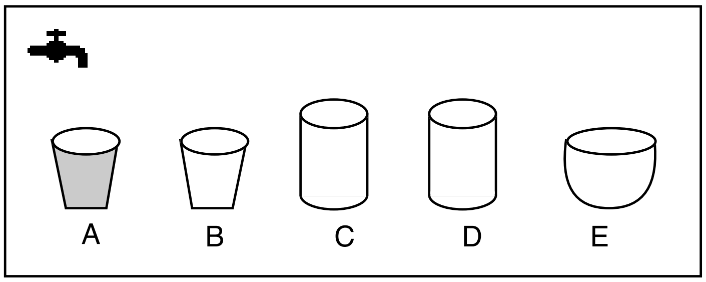
Unplanned generalisations also go through a number of levels similar to those described above.

The learning strategy is best understood by looking at an example from Hume (1991). He gives the complete program traces for the example, here we only provide enough detail to illustrate the main features of CAP.

The world contains a tap from which water may be obtained and a collection of cups and bowls which can contain water and cylinders which cannot (see Figure 2). Two types of actions are possible: pouring water from the tap into any of the objects and pouring from one object into another.

Let us assume that two sequences are observed by CAP, namely, water is poured from cup *A* into cup *B* and cup *A* is filled from the tap. The description of pouring from *A* into *B* is:

cup(a) at 0		cup(a) at 1
cup(b) at 0	pour(a, b) during 0/1	cup(b) at 1
contains_liquid(a) at 0	$\longrightarrow$	~contains_liquid(a) at 1
~contains_liquid(b) at 0		contains_liquid(b) at 1



cup(A)	contains-liquid(A)
cup(B)	~contains-liquid(B)
cylinder(C)	~contains-liquid(C)
cylinder(D)	~contains-liquid(D)
bowl(E)	~contains-liquid(E)

Figure 2: Water world

Notice that for simplicity we have omitted descriptions of objects *C*, *D* and *E*, but they too are part of the complete sequence description. Next, we transform the observation into the initial theory:

```
transfer(Source, Destination) during Initial/Final :-
    cup(Source) at Initial,
    cup(Destination) at Initial,
    contains_liquid(Source) at Initial,
    ~contains_liquid(Destination) at Initial,
    pour(Source, Destination) during Initial/Final,
    cup(Source) at Final,
    cup(Destination) at Final,
    ~contains_liquid(Source) at Final,
    contains_liquid(Destination) at Final.
```

where "transfer" would actually be an arbitrary symbol invented by the program to uniquely identify this concept and all of the object names have been turned into variables. As we noted earlier, there are other objects in the world that are not relevant to this example but which are still present. Predicates involving the irrelevant objects are trimmed out.

Let us know summarise the sequence of steps followed by CAP:

1. The two sequences of actions observed so far have been pouring water from cup *A* to cup *B* and filling cup *A* from the tap.
2. Performing a *repeat* test, CAP attempts to pour from cup *A* to cup *B*. However, there is a problem, at this stage, both cups are full and the theory expects the destination to be empty before starting. In order to force the test to go ahead, CAP performs an unplanned generalisation in which it invents the concept "may contain liquid". This is an *intra-construction* which allows CAP to match the world as it is to the world as it would like it to be in its theory.
3. Testing resumes and completes successfully since the destination contains water at the end. As far as the program is concerned, it does not matter that it already had water and that an overflow occurred. Since the program has learned something at this level, it remains at Level 1.
4. There is another theory awaiting inspection, i.e. filling from the tap. Since cup *A* is empty, this test is repeated and nothing new is learned, so the generalisation level of this theory will be raised.
5. "Transfer" is still a a lower level of generalisation so we return to it. It now tries to pour water from *B* to *A*. This succeeds with nothing new learned so the generalisation level of "transfer" increases.
6. "Transfer" and "fill from tap" are both at the same level of generalisation so either one can be chosen for the next step. We continue with "transfer". The program tests

to see if the source "may contain liquid" by pouring from cup *B* to cup *A* again. *B* is empty but the test apparently succeeds because at the end of the test, *A* still contains water. This results in an incorrect generalisation being accepted which will have to be corrected later. Since something was learned (although incorrectly) the generalisation level remains unchanged.

7. We now alternate between theories at the same level and revert to "fill from tap". We can again try generalising using "may contain liquid" and so we try filling *A* and this succeeds. Again because something was learned, the generalisation level remains unchanged.
8. Although the levels of each theory have remained unchanged, there are no more absorption generalisations possible for either of them so we move on to intra-construction. Staying with "fill from tap" we invent a new concept "cup or cylinder". So far all experiments have filled cups, now we trying filling a cylinder. Of course this fails and the generalisation is recorded as being no good.
9. Another concept that is invented by intra-construction is "cup or bowl". When this is tried with either "fill from tap" or "transfer", it is found to work.
10. As experimentation proceeds we end up having both cups empty. CAP is suspicious by nature and frequently repeats experiments when the state of the world gives it an opportunity to do so. According to its current theory, it should be able to pour from *B* in to *A* because an earlier experiment had supported the premise that the source cup "may contain liquid". After performing the experiment, it finds that the destination does not contain water as was expected. This causes the generalisations on theory to be marked as no good and they are undone.
11. CAP tries to repair the theory by searching for other generalisations. As a result, "contains_liquid(Source)" will be confirmed as necessary.

CAP will continue to look for new generalisations as long as they are consistent with the world. When all possibilities have been exhausted, the program rests until new activity from other agents in the world cause it to begin anew.

Conclusion

Given only a few examples and no background knowledge or domain theory, it is possible to effectively learn concept in a wide range of reactive environments. A representation and learning mechanism using horn-clause logic is largely responsible for this but at a cost of controlling unconstrained behaviour. In most systems with the breadth of application, the solution to this problem results either in a large amount of domain knowledge being necessary (e.g. EBL) or a large amount of intervention from an oracle.

Without domain knowledge or oracle intervention, effective learning has been demonstrated using one control heuristic, i.e. "imitate activity that you see in the environment". The categorisation of positive and negative examples of concepts is done by the environment when an experiment succeeds or fails. The "drive" to learn comes from a continual search to try to match concepts with patterns in the world where the matching is done by trying to "execute" the concepts as sequences of actions that affect the world. When concepts only partially match the world they are generalised in an attempt to complete the match and learning occurs as a result.

References

Hume, D. (1991). *Induction of Procedures in Simulated Worlds*. Ph.D. Thesis, University of New South Wales.

Muggleton, S. (1988). "A Strategy for Constructing New Predicates in First Order Logic" in *Proceedings of the Third European Working Session on Learning*, D. Sleeman (ed). Glasgow: Pitman.

Muggleton, S. and Buntine, W. (1988). "Machine Invention of First-order Predicates by Inverting Resolution" in *Proceedings of the Fifth International Machine Learning Workshop*, R.S. Michalski, J.G. Carbonell, T.M. Mitchell (eds). Ann Arbor: Morgan-Kaufmann.

Sammut, C. (1981). *Learning Concepts by Performing Experiments*, Ph.D. Thesis, University of New South Wales.

Sammut, C. and Banerji, R.B. (1986). "Learning Concepts by Asking Questions" in *Machine Learning: An Artificial Intelligence Approach (Vol. 2)*, R.S. Michalski, J.G. Carbonell, T.M. Mitchell (eds). Morgan-Kaufmann.