

Applying Inductive Logic Programming in Reactive Environments

David Hume
Claude Sammut¹

School of Computer Science and Engineering
University of New South Wales
PO Box 1, Kensington NSW
Australia 2033

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 *regeneration* 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

¹ Address all correspondence to Claude Sammut (email: claudio@spectrum.cs.unsw.oz.au)

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 to 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 a 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:

Missing MathType's Belmont font.