Learning Where To Go without Knowing Where That Is: The Acquisition of a Non-reactive Mobot Behaviour by Explicitation

Chris Thornton
Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QN
Email: Chris.Thornton@cogs.susx.ac.uk
Tel: (44)27 678856

December 14, 1994

Abstract

In the path-imitation task, one agent traces out a path through a second agent's sensory field. The second agent then has to reproduce that path exactly, i.e. move through the sequence of locations visited by the first agent. This is a non-trivial behaviour whose acquisition might be expected to involve special-purpose (i.e., strongly biased) learning machinery. However, the present paper shows this is not the case. The behaviour can be acquired using a fairly primitive learning regime provided that the agent's environment can be made to pass through a specific sequence of dynamic states.

Introduction

In reactive mobot behaviours such as wall-following and pursuit, specific stimuli evoke specific responses. As a result, the input/output profile for the behaviour shows marked correlations. The existence of these means that the behaviour can be straightforwardly acquired using any one of the wide range of reinforcement, neural-network and evolutionary methods — in fact anything capable of exploiting statistical effects. In non-reactive behaviours, responses are reactions to hidden states, i.e., states which are temporally, physically or logically inaccessible to the agents sensors. Such a behaviour's input/output profile does *not* show marked correlations and thus cannot be straightforwardly acquired using conventional techniques.

Path-imitation is a case in point. In the path-imitation task, one agent traces out a path through a second agent's sensory field. The second agent then has to reproduce that path exactly, i.e. move through the sequence of locations visited by the first agent. This is a non-trivial and clearly non-reactive task since the behaving agent's moves are 'reactions' to temporally remote (i.e., in-the-past) states of the environment. The acquisition of path-imitation behaviour might be expected to involve special-purpose (i.e., strongly biased) learning machinery. However, this turns out not to be the case.

The behaviour can be acquired using a fairly primitive learning regime provided that the behavioural environment can be made to pass through a specific sequence of *dynamic* states.

The paper breaks down into four main sections. First, I describe the acquisition experiment performed. Second, I analyze the resulting architecture of the acquisition agent. Third, I review the theoretical background and motivation for the work. (Theoretically oriented readers may prefer to read the third section first.) The fourth and final section is a discussion.

1 he acquisition experiment

The acquisition experiment involved subjecting a simulated mobot (mobile robot) to a sequence of environmental scenarios. Learning accomplished by the mobot in the earlier scenarios had an impact on the character of later scenarios. Thus, the developmental process involved an ongoing *interaction* between learner and environment. The ultimate success of the process relied on there being a tight-coupling between the 'developmental trajectory' (changes in the learner) and the 'environmental trajectory' (changes in the environment). The learner mobot was equipped with two proximity sensors (shown as dashed-lines in Figure 1). Each one of these sensed the proximity of the nearest obstructing surface along a particular ray. Proximity values were normalized in the range 0..1, with lower values indicating lower sensed proximities. All sensor inputs were noisy with noise increasing linearly with measured range. The boundary of the space, shown here as a solid line, was not sensed by the mobot.

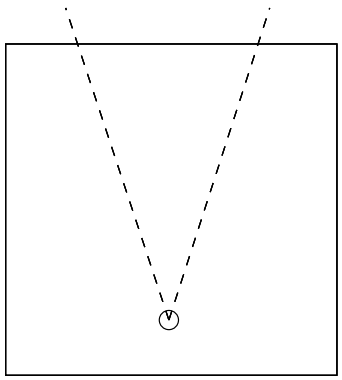


Figure 1: The learner mobot.

2 he acquisition method

For reasons that will become clear, I call the acquisition process used in this experiment 'explicitation'. It is essentially the competitive learning regime of Rumelhart and Zipser (1) with three modifications. The first modification enables the regime to operate incrementally. With this modification, competitive learning adds nodes to the network as and when they are needed. The process comes to an end when

each node is fully tuned to the inputs that it has captured.[1]

The second modification associates a significance value with each weight in the network. In standard competitive learning, each weight is equally significant in the computation of node response and nodes thus have no way to discount particular inputs. As a result there is no straightforward way for them to capture a statistical effect which does not involve all the inputs (2). In the explicitation regime, all weights for all competitive units have an associated significance value whose value reflects the accuracy of the weight as an input-value predictor. Node response is computed taking the significance values into account, and thus weights with low accuracies (significance values) are effectively discounted where appropriate.

The third modification enables the regime to operate recursively. The capture of some effects by a set of competitive nodes triggers a further round of competitive learning in which the input data are *descriptions* of the nodes themselves. For each grouping identified by this process, an internal variable is created and a link made so that the variable's value shows which of the nodes in its group is most active. New data are then derived by presenting the original data and reading off the values of the internal variables. These new data permit the re-appgrou0.10c(of5-10000.6(the99.7(the0m6putation)-(re-app)-170((9.92(alue

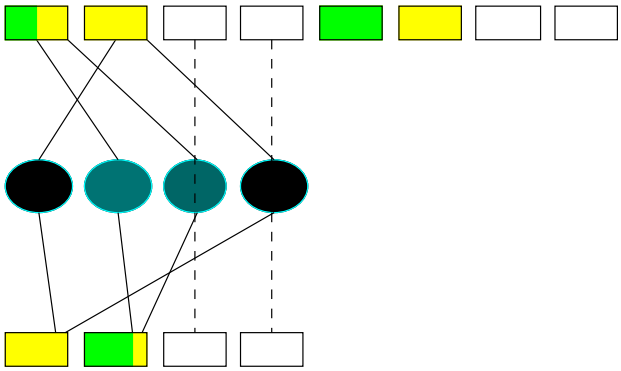


Figure 4: The structure of subnet-1.

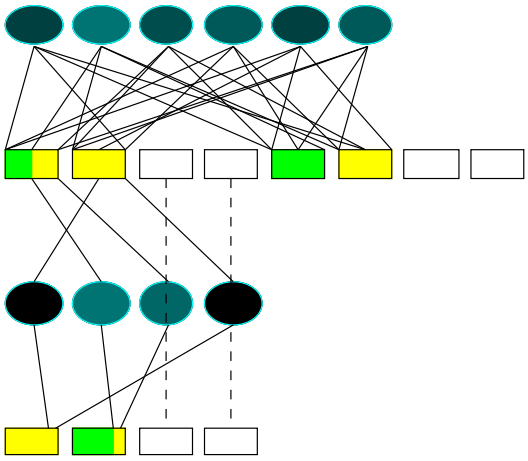


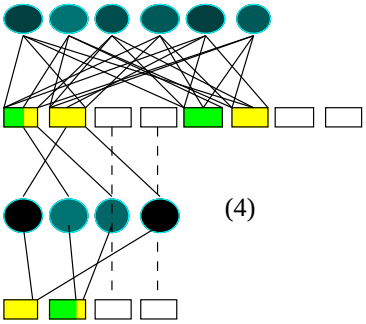
Figure 5: Competitive nodes in subnet-2.

The first node, for example, detects the situation in which the facilitator mobot is moving away from the learner on the left hand side.

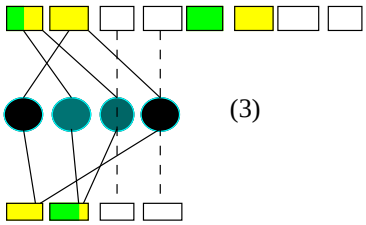
When competitive learning is applied to the descriptions of the competitive nodes in subnet-2, two main groupings are recovered. In one group we have the two nodes which detect facilitator motion away (on left and right). In the other group we have the two nodes which detect facilitator motion towards. The two internal variables generated thus form *bilateral* approach and retreat detectors. Each one measures the degree to which the facilitator is moving towards, or away from the learner on *either* side. This subnet appears in the bottom, right corner of Figure 6, which shows the sequence of subnets produced up to this point.

As we will see, these subnet-2 output variables will turn out to play a crucial role in the production of path-imitation behaviour. Their values encode the relative motion of the facilitator and can thus be straightforwardly used to drive the motors of the learner during path replication.

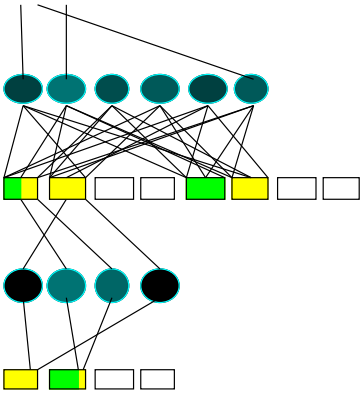
(1)



(4)



(3)



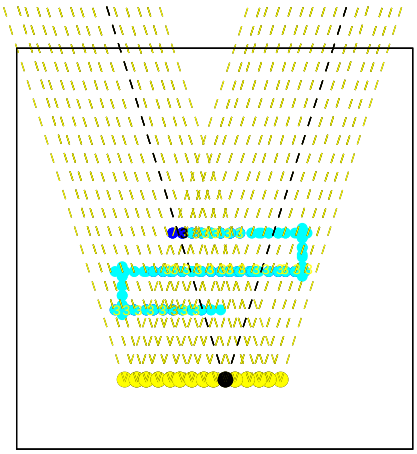


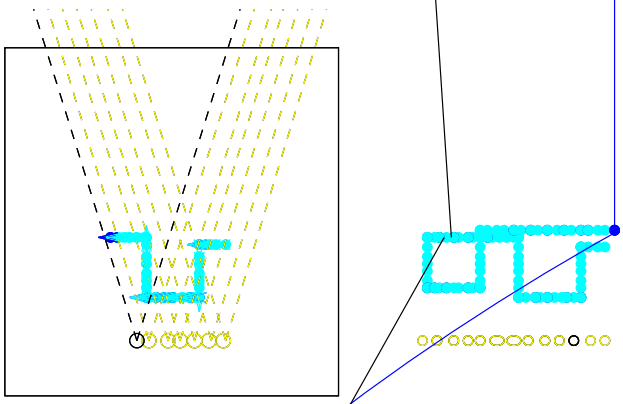
Figure 7: Environment-2.

path, the learner responds by tracking the facilitator across the space, see the upper-left box in Figure 8. As the learner tracks the facilitator across the space, the output variables in subnet-2 are instantiated with values which reflect the relative motion of the facilitator. The bilateral approach/retreat detectors encode the left/right motion of the facilitator. The raised motor variables encode the forwards/backwards motion.

By storing the sequence of instantiations produced in subnet-2's output variables, we thus acquire a sequence of relative-motion descriptions which can be used to regenerate the path executed by the facilitator. Feeding this program (after suitable post-processing) into the learner's motor system, we effectively obtain the desired path-imitation, see the lower two boxes in Figure 8 . In the lower-right box, the facilitator's path is shown using a light dashed line. The learner's imitation of it is shown using a heavier, dashed line.

3 he architecture: a guided tour

In Figure 9 we see the final network architecture produced by the learning process. The various shaded areas correspond to functional components. In the lower, left part of the architecture we have the 'Prototypical proximities detection subsystem'. Recall that this essentially serves to clean-up the relatively noisy proximity inputs. This cleaning-up process is actually an essential part of the overall acquisition process since it paves the way for the development of the unilateral motion detectors which emerge in subnet-2. It is only thanks to the lack of noise in the proximity measures (produced in subnet-1's output variables) that subnet-2's node are able to capture the relevant approach/retreat patterns. In the right-hand part of subnet-1 we have the motor-control subsystem. This is perhaps the simplest component in the entire architecture. The nodes have captured the two main



(2) $P(y|x') = 1$, where x' is either the current input or some part of it, or if

(3) $P(y|g(x)) = 1$, where g is some arbitrary function and x is the current input.

This taxonomy is derived simply by enumerating the possible syntactic forms for the justification. It is invariant with respect to the probability value selected (we do not have to use $p = 1$). It is also *exhaustive* since there is no alternative way of characterizing the conditional or unconditional probability of y being the right output for input x. The interesting consequence of this is that any learning method which produces justified output guesses, must exploit (i.e., use in the generation of guesses) some combination of these three forms of justification.

The cash value of this becomes clear when we consider the ways in which each of the three forms can be exploited. Exploiting a justification in this context means 'finding' the probability in a given distribution. Thus the complexity of exploiting a particular justification is related to the size of the relevant distribution. This prompts us to split the justification forms up according to whether the relevant distribution is *finite*. In particular, we must distinguish between what I call the **direct** forms $P(y)$ and $P(y|x)$ which are associated with finite probability distributions, and the **indirect** form $P(y|g(x))$ which is associated with an infinite one. The distribution $P(y|g(x))$ is infinite due to the infinite number of choices to be made regarding g, which is defined as any computable function.

This analysis of justification sources leads directly to a fundamental insight concerning the complexity of learning problems. Problems which involve exploiting either of the two direct forms involve the equivalent of sampling a finite distribution while problems which involve exploiting the indirect form involve the equivalent of sampling an infinite distribution. Other things being equal, the task of exploiting direct forms thus has a *lower* theoretical complexity than the task of exploiting the indirect form. Note that this is a qualitative distinction similar to the one between polynomial and exponential time complexity. However, it has a firm, mathematical basis in the enumeration of syntactic forms for probability values.

4.1 Statistical v. relational problems

It is important to note that values of the function g (which I call the **recoding function** below) must depend on *relative* argument values. In other words, the function must compute or evaluate a relational property of its arguments. Were we to have an indirect justification in which values of the recoding function depended on the *absolute* values of its arguments, then we could specify the same justification purely in terms of those absolute values, i.e., by deleting the g and associated parentheses. Thus, all indirect-form justifications necessarily involve functions which effectively compute relationships of their arguments.

In recognition of this, I call the class of higher-complexity problems which involve exploiting indirect justifications **relational** and the class of lower-complexity, direct-justification problems **statistical** (since exploiting a direct-form justification involves sampling for a statistical effect in the form of an observed probability). If it was not obvious before, the introduction of this new terminology should drive home the point that this analysis gives a foundation to the well-known Machine Learning heuristic which states that 'learning relationships is hard' [4].

5 Example

To illustrate this notion of indirect justification, consider the following training set. This is based on two input variables (x1 and x2) and one output variable (y1). There are six training examples in all. An arrow separates the input part of the example from the output part.

```
x1  x2      y1

1   2  -->  1
2   2  -->  0
3   2  -->  1
3   1  -->  0
2   1  -->  1
1   1  -->  0
```

A variety of direct justifications are to

In some cases, the absolute size of the relevant mapping may make this process rather long-winded. However, the presence of strong input/output correlations can often be demonstrated using qualitative reasoning.