Discover Relevant Environment Feature Using Concurrent Reinforcement Learning

Zhihui Luo, David Bell, Barry McCollum
School of Computer Science
Queen's University Belfast, BT9 1NN, UK
{zluo02, da.bell, b.mccollum}@qub.ac.uk

Introduction

When a robot observes its environment, there are two important characteristics of the perceived information. One is the relevance of information and the other is redundancy. The irrelevant and redundant features which commonly exist within an environment, commonly leads to state explosion and associated high computational cost within the robot's learning process.

We present a method concerning the relevance of information in order to improve the learning of a reinforcement learning robot. We introduce a new concurrent online learning algorithm to calculate the *contribution C(s)* and *relevance degree I(s)* to quantify the relevancy of features with respect to a desired learning task. Our analysis shows that the correlation relationship of the environment features can be extracted and projected to concurrent learning threads. By comparing the contribution of these learning threads, we can evaluate the relevance degree of a feature when performing a particular learning task.

We demonstrate the method on the chase object domain. Our validation results show that, using the concurrent learning method, we can efficiently detect relevant features from the environment on sequential tasks, and therefore improve the learning speed.

Environment Feature Correlation

Let x, y be the only two independent state features at the robot's state s, then s can be defined as $s = (s^x, s^y)$. The learned policy for task S using both of these features is:

$$\pi^*(s) = \pi^*(s^x, s^y) = \arg\max_{a \in A} Q(s^x, s^y, a) \quad (1)$$

Extending (Jong & Stone 2004) 's approach, we define feature x relevant if (2) is satisfied.

$$\Pr\left(\pi^*(s^x, s^y) = \pi^*(s^x)\right) > \omega \quad (2)$$

Threshold ω is used to determine whether the probability is high enough. We aim to develop a criterion to measure the correlation among features defined in equation (2). Relevancy features are defined based on the learned action policy. Hence, it is intuitive to find a method to compare the similarities among the features' policies. However, learning and comparing each feature's policy one by one is very time-consuming and resource intensive, especially for dynamic robot environment.

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Concurrent Learning

In order to compare the policies more efficiently, we introduce a new reinforcement learning method called *concurrent biased learning*. This is a multi-thread learning method, in which each learning thread refers to one feature of the environment. If an agent intentionally focuses on part of these environmental features to learn a policy of a task, we call this method a *biased learning*; otherwise, if an agent uses all features that it perceives to learn a task, we call this *unbiased learning.*

If the agent learns a policy $\pi(s^i)$ with respect to one of the environmental features i, we call this biased learning with respect to feature i. The biased Q-value can be denoted as $Q(s^i, a)$. Then the Bellman update function of the biased Q-value is:

$$Q(s^i_{t-1}, a_{t-1}) \leftarrow Q(s^i_{t-1}, a_{t-1}) + \alpha[r_t + \gamma \max Q(s^i_t, a_t) - Q(s^i_{t-1}, a_{t-1})] \quad (3)$$

To compare the policies, we equip the robot with all the biased and unbiased learning threads. When executing a task, the agent performs these learning threads simultaneously. These concurrent learning threads with different state features are updated from the experience of the same task. But each thread only concentrates on the changing aspect from its own prospect of view and ignores others.

Relevancy Analysis

The basic idea of our relevancy analysis is to compare the learned action policy among learning threads. We define the match trace of feature *i* as:

$$c^i(s) = \begin{cases} 1 & \text{if } \pi^*(s) = \pi^*(s^i) \\ 0 & \text{if } \pi^*(s) \neq \pi^*(s^i) \end{cases} \quad (4)$$

The match trace records a value of 1 when the biased learning thread provides the same greedy action as the unbiased learning thread at time t in state s. Otherwise, if they are not the same, $c(s)$ records a value 0. The *contribution* of feature *i* at state *s* is defined by the sum of match traces across a period of time:

$$C^i(s) = \sum_{t=0}^{t=t'} c^i(s) \quad (5)$$

Then we can calculate the *relevance degree* of the feature *i* based on the contribution proportion of this feature among all *n* numbers of decisions made at state *s*:

$$I^i(s) = \frac{1}{n} C^i(s) \qquad (6)$$

To automate the evaluation of the relevance degree, we combine online Q-learning with the implementation of concurrent biased learning.

Experiment Result

Chasing objects is a common scenario in the robotics area. We simulated this in a discrete grid world. This world is composed of 10*10 grid locations. There are 3 balls in this world marked ball ①, ball ②, and ball ③. All balls can move one step randomly either in vertical, horizontal or diagonal directions. The robot in this world also can move one step in each direction. It detects the exact locations of all objects.

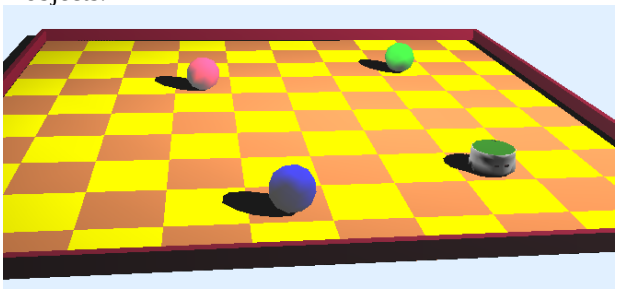

Figure 1: Simulation - Robot chases 3 moving balls

The task for the robot is to learn to follow and catch these balls one by one in a predefined sequence. We define the task for the robot to catch these balls in a sequence of ①, ②, ③. Only when the robot performs this sequential task correctly, will it receive a positive reward +10 at ball ③. In this example, the environment is highly dynamic due to the movement of three target objects and complex as there is only one delayed positive reward. We apply our algorithm to find the relevant target ball, and then compare the learning speed before and after applying our online feature discovery method.

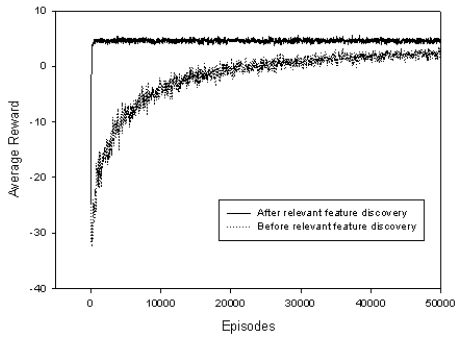

Figure 2: Learning before & after relevant feature discovery

Figure 2 compares the learning efficiency of learning the agent before relevant features discovery and after relevant feature discovery. The graph shows that after relevant features being discovered in every state, the agent can learn significantly faster than when using all features.

Task Stage	Chase ①	Chase ②	Chase ③
Average value of $I^i(s)$			
$I^1(s)$	★0.548	0.323	0.358
$I^2(s)$	0.317	★0.672	0.325
$I^3(s)$	0.291	0.307	★0.886

Table 1: Relevance degree at different stages of learning

Table 1 summarizes the average relevance degree for all three features at different stages of learning. At each stage, the maximal relevant feature is marked with a star sign. It can be clearly seen that our method successfully distinguishes the most relevant ball at each stage of learning. Ball ① has the highest relevance degree at learning stage chasing ball ①, and ball ② at stage 2 so on. Overall, ball ③ has the highest relevant degree at stage 3. The reason is that this part of the states is close to the reward and converges earlier than other states.

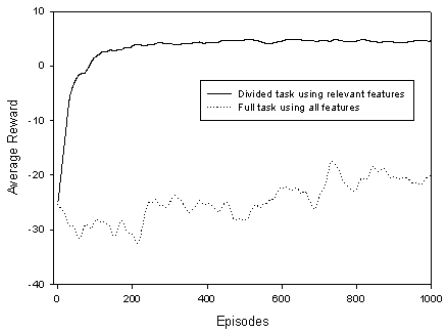

Figure 3: Compare knowledge discovery on task

If the learning robot uses the discovered knowledge in table 1 to perform the task, it can focus only on the most relevant ball at different stages of learning. The learning results are compared in figure 3. It shows that, using a relevance observing method on this task, the learning is not only faster but also smoother in nature than simple Q-learning.

Conclusion

This paper presented preliminary research on discovering relevant feature using concurrent RL. Our work is also influenced by research on concurrent hierarchal RL (Marthi et al. 2005) and research on automated state abstraction (Jonsson & Barto 2001).

References

Jong, N.K. and Stone, P. 2004. Towards Learning to Ignore Irrelevant State Variables, *The AAAI-04 Workshop on Learning and Planning in Markov Processes*, San Jose, CA.

Marthi,B.; Russell, S.; Latham, D. and Guestrin, C. 2005. Concurrent hierarchical reinforcement learning, *In Proc. IJCAI-05*, Edinburgh, Scotland.

Jonsson, A. and Barto, A.G. 2001. Automated State Abstraction for Options using the U-Tree Algorithm, *In Advances in Neural Information Processing Systems.*