Issues in putting reinforcement learning onto robots

Jeremy Wyatt

Department of Artificial Intelligence,
University of Edinburgh,
5 Forrest Hill,
Edinburgh EH1 2QL,
Scotland, U.K.

March 25, 1995

Abstract

There has recently been a good deal of interest in robot learning. Reinforcement Learning (RL) is a trial and error approach to learning that has recently become popular with roboticists. This is despite the fact that RL methods are very slow, and scale badly with the size of the state and action spaces, thus making them difficult to put onto real robots. This paper describes some work I have been doing on trying to understand why RL methods are so slow and on how they might be speeded up. A reinforcement learning algorithm loosely based on the theory of hypothesis testing is presented as are some preliminary results from employing this algorithm on a set of bandit problems.

1 Introduction

There are many ways in which we can think of categorising robot learning methods. One useful distinction is that between when the performance of the robot during the learning period matters and when it does not matter. In this paper the former will be refered to as *on-line* learning and the latter as *off-line* learning. An example of the first is when a robot must cope with unforeseen situations while operating autonomously in pursuit of a goal; an example of the second is when we use learning to try to find the best way to achieve a task as part of the process of designing a robot controller[1].

On-line learning is similar in many respects to our common-sense notion of 'learning'. An agent, presumably with a large amount of knowledge about its task and environment, encounters a new situation. It must quickly grasp the correct course of action, and do so without intervention by the operator. In such circumstances there is little room for error, and hence a trade-off exists between exploring and exploiting the environment. In *off-line* learning the criteria for success are much simpler. The agent merely needs to maximise its performance on a task by the end of the learning period. We lose the need to exploit the environment because the agent's performance *during* the learning period doesn't matter. In addition the learning process need not be entirely autonomous. The designer may intervene in a number of ways. These could feasibly include driving the robot through some example solutions [5, 11]; decomposing the task into simpler tasks and then recomposing the solutions to these [14, 15]; or providing intermediate indicators of performance, i.e. telling the robot when it's doing well even if it hasn't achieved the goal yet [16].

This paper is concerned with problems in off-line learning. In particular it looks at off-line learning using Reinforcement Learning (RL) methods. The basics of RL will now be outlined before we discuss some of the difficul-

[1] See [2] for a similar distinction.

ties encountered when employing it in robots. A reinforcement learning algorithm will then be presented which is loosely based on the theory of hypothesis testing.

2 Reinforcement Learning

Reinforcement learning is a trial and error approach to learning in which an agent operating in an environment learns how to achieve a task in that environment. The agent learns by adjusting its *policy*[2] on the basis of positive (or negative) feedback — termed reinforcement. This feedback takes the form of a scalar value generated by the agent each time step, high and low values corresponding to rewards and punishments respectively. The mapping from environment states and agent actions to reinforcement values is termed the *reinforcement function*. The agent converges to the behaviour maximising reinforcement (the *optimal policy*). In theory an appropriate reinforcement function[3] exists for all tasks, although finding such a function is typically hard [1].

It is worth noting that unlike in supervised learning procedures the error signal does not indicate which behaviour is correct, merely how good or bad the current behaviour is relative to others. This means that in order to find the best action in each state the agent must try it at least once and hence in order to *guarantee* converging to the optimal policy the agent must try all actions in all states at least once[4]. In addition the feedback received is usually delayed[5], and hence most work on RL has been on solving the problem of assigning credit (or blame) to individual actions within a sequence leading to the receipt of reinforcement. This is known as the *temporal credit assignment* problem (TCA). It is to be distinguished from the better known *structural credit assignment* problem (SCA), which is concerned with assigning credit to features of a state in order to generalise across states.

[2]The mapping from environment states to agent actions.

[3]i.e. one in which the policy maximising reinforcement corresponds to the behaviour which the *designer* considers optimal.

[4]An infinite number of times if the environment is stochastic.

[5]This is not *necessarily* the case e.g. the two-armed bandit task used here.

3 Problems putting RL onto robots

Recently a lot of work has been done trying to put RL algorithms onto real robots and there have been a number of successful implementations to date [7, 9, 14, 15, 16, 19]. There are, however, a number of difficulties associated with RL methods *per se* and these are especially pertinent to the problem of using them with real robots. In short RL makes assumptions which do not apply to real world tasks [16].

First RL assumes that the environment as perceived by the agent is a Markov Decision Process (MDP). Informally this means that the agent only need know the current state of the process in order to predict its future behaviour[6]. If the agent does not have sufficient information to predict the future process behaviour[7] then what Whitehead terms *perceptual aliasing* [26] occurs. This is when an agent cannot distinguish between two states which are significantly different with respect to their behaviour under the same policy. New algorithms have been designed to cope with this phenomena [13, 26] but are not guaranteed to converge to the optimal policy under such conditions.

The second major problem is that of slow convergence. Even in a perfectly deterministic Markovian environment RL takes time exponential in the size of the state space to converge to the optimal policy. In stochastic environments with large state spaces (e.g. a typical robot task) this means that times to convergence are prohibitively long. There are necessarily two types of solution: either make your temporal credit assignment mechanism faster or make your temporal credit assignment problem simpler. The first approach has manifested itself in two ways: work on more efficient trace mechanisms [4, 20, 21, 24], and work on the use of generalisation methods [3, 17]. The second approach includes methods such as task decomposition [6, 10, 12, 15, 22, 8]

[6]In principle any process can be represented as an MDP because an abitrary amount of information about the history of the process can be included in the description of the current state, e.g. we need to know the velocity and acceleration of a ball in order to be able to calculate its trajectory.

[7]In the case of RL this means it does not have sufficient information to predict average return accurately.

1 $t = 0$
2 $n_i = x_i = 0, \forall i$
3 $a(t) = a_j$ such that $\xi(a_j) \geq \xi(a_i), \forall i$
4 update $n_j = n_j + 1$
5 if $r(t) = 1$ then $x_j = x_j + 1$
6 $p_j = x_j \ n_j$
7 $p_{max} = \max(p_i)$, $p_{min} = \min(p_i)$, $\forall i$
8 if $\Pr(P_{max} - P_{min} \geq p_{max} - p_{min}) \leq \alpha$ then drop a_{min}
9 $t = t + 1$
10 goto step 3

Where $\xi(a)$ is the width of the $100(1-\alpha)$ confidence interval for action a; $a(t)$ is the action at time t; and n_i, x_i are the number of trials and successes for action a_i. p_i is the proportion of successes for action a_i. $P_i = X_i/n_i$, where $X_i \sim \mathrm{bin}(X_i; n_i, \pi_i)$.

Figure 1: Algorithm 1 — Based on hypothesis testing.

1 $t = 0$
2 choose $a(t)$. $\Pr(a(t) = a) = \Pi(a, t)$, $\forall a$
3 $p(t+1) = p(t) + \beta(r(t+1) - p(t))$
4 $\Pi(a, t+1) = \Pi(a, t) + \begin{cases} \alpha(r(t+1) - p(t)) & \text{if a=a(t)} \\ -\Pi(a,t) \times \alpha(r(t+1) - p(t))/(1 - \Pi(a(t), t)) & \text{otherwise} \end{cases}$
5 $t = t + 1$
6 goto step 2

Where Π is the agent's policy; $r(t)$ and $p(t)$ are the actual and predicted reinforcement values; and α, β are the learning rates for the policy and the value function respectively.

Figure 2: Algorithm2 — A variant of Sutton's reinforcement-comparison algorithm

and the construction of better reinforcement functions [16][8]. But even if we can solve all these difficulties in one robot simultaneously (solutions to one problem tend to prevent solutions to other problems [9]) we still have 'lesser' implementational difficulties such as the fact that RL algorithms are discrete when processes in the world are continuous.

While RL is a very general, and hence weak AI method, robots present problems that require strong, and therefore specific solutions. A tabula rasa approach to RL will not work in robots. Attempts to seed the robot behaviour with designer knowledge [5, 15, 16] need, if possible, to be systematised. The situation is analogous to that faced by the expert systems community in the early 1980s. How do we transfer our expert's knowledge into our expert system? In this case how do we transfer our expert knowledge about how to operate in uncertain physical environments into our robots?

One way round part of this robot knowledge acquisition problem is to teach a robot by driving it through a task along a near-optimal solution several times [11]. But unless this is merely to become a form of supervised learning a principled means of integrating the taught (supervised) and exploratory (unsupervised) approaches is required.

One method is to give an agent knowledge about how much it knows about different portions of the state space [18, 25]. In an off-line learning methodology the agent must then explore the environment so as to maximise its knowledge about the task domain rather than in order to maximise short-term rein-

[8]Even if a reinforcement function is *appropriate* in the sense outlined above it may still fail to give the agent sufficient feedback to ensure speedy learning, and so may not be a *good* reinforcement function.

[9]e.g. if we use a generalisation method then we risk over-generalising and thus exacerbating the perceptual aliasing problem.

forcement. In other words we cannot use pure exploitation policies or the agent may get trapped in the taught (and possibly sub-optimal) behaviour. Being greedy with respect to knowledge can be better than being greedy with respect to rewards.

4 Reinforcement Learning using hypothesis tests

As mentioned previously, work on speeding up RL algorithms has concentrated on speeding up temporal credit assignment i.e. the rate at which the agent constructs estimates of the mean return for each state. We can however also alter the rate at which the policy is adjusted. The problem of setting learning rates for policies can be most easily and thoroughly studied in an envionment without the need for a secondary-reinforcement mechanism. The environment chosen here is a bandit-task of a kind already widely studied in the literature [9, 23]. Reinforcement is immediate and boolean. A summary of the tasks used is given in Figure 4.

The performance of an RL algorithm such as the AHC algorithm [23] depends on being able to pick a good value for the learning rate for the policy, α, and for the value function, β[10]. Figure 3 shows that the performance varies as an inverted U-shaped function of α. The optimal value of α roughly increases with the size of the difference between the mean reinforcements for each action and decreases with its variance. Thus the optimal value of α can vary considerably across tasks. As the degree of uncertainty in an environment is usually unknown this makes RL algorithms hard to optimise.

One way to overcome this would be to set the learning rate automatically[11]. Another approach is to remove the learning rate parameter altogether. This is the approach taken here in Algorithm 1 (see Figure 1). The algorithm works in the following manner.

The agent is initially equally uncertain about all actions, so the probability distribution across actions is uniform when $t = 0$. Each time step the action is chosen about which the agent knows least (*step 3*). This is the action with the widest binomial confidence interval on p, the proportion of successes. The statistics n, x and p are updated (*steps 4* and *5*). Then the best and worst looking actions are found (*step 7*). The agent hypothesizes that actions are equally good, and calculates the likelihood of this result on the basis of the observed results. If the observed difference $p_{max} - p_{min}$ is unlikely on the 'null' hypothesis that the true rates of success for each action are the same (i.e. $\pi_{max} = \pi_{min}$) then this is rejected at the $100\alpha\%$ level of significance and the alternative hypothesis that the best looking action is indeed better is accepted (*step 8*). In this case the action a_{min} is dropped from the list of actions considered for execution. The calculations are made on the binomial distribution. Actions are not dropped in order that the environment be exploited more effectively during the learning period, but so that the environment may be explored more efficiently. The algorithm has converged when all actions but one have been dropped. The final result is an ordering on the set of all actions.

The performance of this algorithm was

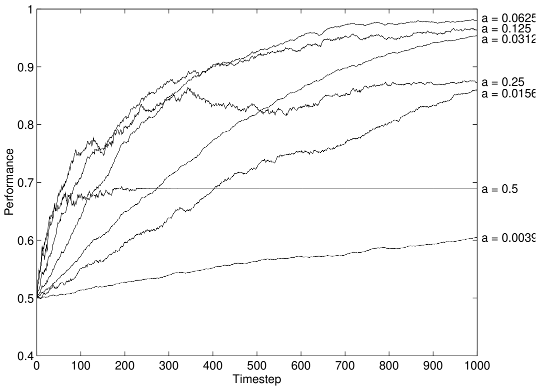

Figure 3: Performance of a variant of Sutton's reinforcement-comparison algorithm (see Figure 2) on a 2-armed bandit task. The curves of the averge performance for different values of α are shown ($\beta = \alpha$). Performance is scaled between the worst policy and the best policy: 1 is optimal, 0.5 is random. $\pi_0 = 0.55$, $\pi_1 = 0.45$. The figures were calculated from 100 trials of 1000 time steps each.

[10] Work has already been done on setting β automatically [24], but no such work has been done on setting α.

[11] e.g. by adjusting α according to the variance in the average return for each state (RC and AHC algorithms) or for each state-action pair (Q-learning).

Task	$\Pr(r=1)$	
	a_0	a_1
1	0.9	0.1
2	0.55	0.45
3	0.9	0.8
4	0.2	0.1

Figure 4: Table of the 2-arm bandit tasks used. $R = 0, 1$. Each pull on an arm is a Bernoulli trial. For convenience a_0 is always optimal.

tested against a variant of Sutton's reinforcement algorithm on the four two-arm bandit tasks described in Figure 4. The reinforcement-comparison (RC) algorithm used was optimised across α, β being set to the same value except in the case of Task 1 where $\beta = .1$ was found to be optimal. For both algorithms and all tasks 200 simulation runs were made, each of 2000 timesteps.

For each step in each run the scaled performance of the exploitation policy the agent currently considers best was recorded. Note that for the agent using Algorithm 1 this is not the actual reinforcement generated during the learning run, but an estimate of how much the agent has learned about the task in the form of a recommended exploitation policy. The recommended exploitation policy for Algorithm 1 is a uniform distribution across the remaining eligible actions. The exploration policy is the one followed during the course of the learning period. The scaled performance is calculated using equation 1.

$$p = \frac{E[R]_c - E[R]_w}{E[R]_o - E[R]_w} \qquad (1)$$

where R is the random variable denoting return, and c, o and w denote the current, optimal and worst policies respectively.

Figures 5-8 show the mean performance for each algorithm over 2000 timesteps.

5 Discussion

It can be seen from Figures 5-8 that on Tasks 2 and 3, Algorithm 1 converges significantly faster than the reinforcement-comparison algorithm. On Tasks 1 and 4 it converges at about the same rate. On Tasks 2 and 4 how-

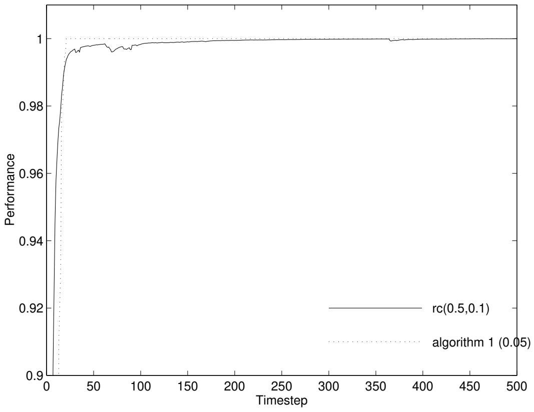

Figure 5: Curves of the mean performance of Algorithm 1 (see Figure 1) and a variant of Sutton's RC algorithm (see Figure 2) on Task 1. Performance is scaled between the worst policy and the best policy: 1 is optimal, 0.5 is random. $\pi_0 = 0.9$, $\pi_1 = 0.1$. Both algorithms were run for 200 trials, each of 2000 time steps. The parameter α of Algorithm 1 was set to 0.05. For the RC algorithm $\alpha = 0.5$ and $\beta = 0.1$.

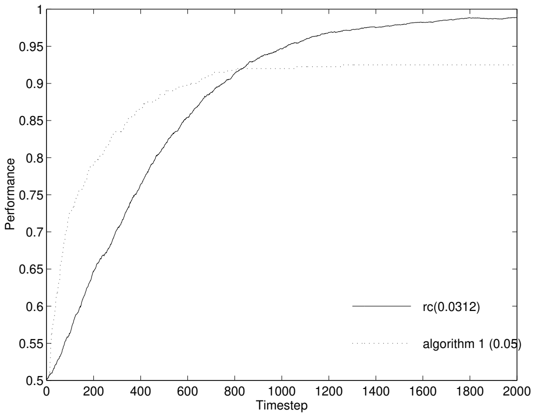

Figure 6: Curves of the mean performance of Algorithm 1 (see Figure 1) and a variant of Sutton's RC algorithm (see Figure 2) on Task 2. Performance is scaled between the worst policy and the best policy: 1 is optimal, 0.5 is random. $\pi_0 = 0.55$, $\pi_1 = 0.45$. Both algorithms were run for 200 trials, each of 2000 time steps. The parameter α of Algorithm 1 was set to 0.05. For the RC algorithm $\alpha = \beta = 0.0312$.

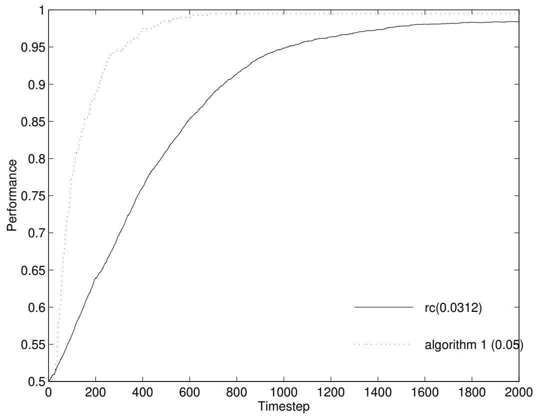

Figure 7: Curves of the mean performance of Algorithm 1 (see Figure 1) and a variant of Sutton's RC algorithm (see Figure 2) on Task 3. Performance is scaled between the worst policy and the best policy: 1 is optimal, 0.5 is random. $\pi_0 = 0.9$, $\pi_1 = 0.8$. Both algorithms were run for 200 trials, each of 2000 time steps. The parameter α of Algorithm 1 was set to 0.05. For the RC algorithm $\alpha = \beta = 0.0312$.

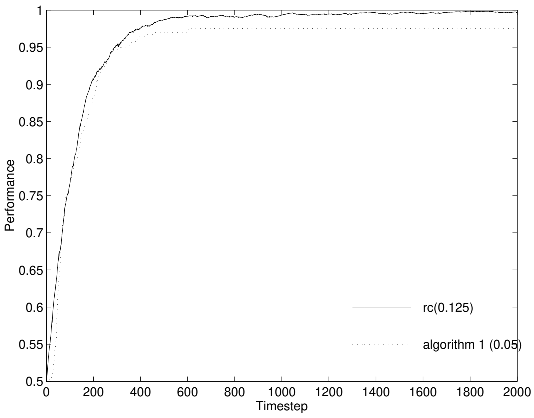

Figure 8: Curves of the mean performance of Algorithm 1 (see Figure 1) and a variant of Sutton's RC algorithm (see Figure 2) on Task 4. Performance is scaled between the worst policy and the best policy: 1 is optimal, 0.5 is random. $\pi_0 = 0.2$, $\pi_1 = 0.1$. Each algorithm was run for 200 trials, each of 2000 time steps. The parameter α of Algorithm 1 was set to 0.05. For the RC algorithm $\alpha = \beta = 0.125$.

ever it converges to a significantly poorer policy on average than the RC algorithm. Why is this?

There is a probability that at each timestep the observed results will cause the null hypothesis ($\pi_0 = \pi_1$) to be rejected in favour of the alternative hypothesis $\pi_1 > \pi_0$. The algorithm then converges to a sub-optimal policy. The probability of such a result gets larger as the difference between π_0 and π_1 decreases, and as the variance in p_0 and p_1 increases. This is why the algorithm converged to such a poor policy on Task 2. There are two possible solutions to this problem: either the significance level of the test can be altered dynamically during the learning period in order to keep the probability of an error low; or more simply, actions can become eligible again if the difference between them and the best action falls outside the critical region once more. This process of unsticking the algorithm would be strengthened by retaining a small probability each timestep of executing one of the actions that the agent has rejected. These strategies are currently being implemented and tested.

It is significant that Algorithm 1 performs reasonably on all tasks with the same value for α, whereas the RC algorithm must be optimised with respect to both α and β.

This paper has argued that RL methods making better use of the information available to them and when combined in a principled manner with teaching methods do have the potential to be fast enough to scale to learning more complex robot tasks. A new algorithm has been presented based on the theory of hypothesis testing, and some initial results presented that show some promise. While it is clearly not the case that the algorithm presented here is ideal, the results indicate that algorithms of a similar kind may have potential.

There are a number of extensions that need to be carried out to this work. First a means for reducing the error rate for Algorithm 1 must be found. Secondly more complex bandit trials should be carried out to confirm that the advantage scales to higher numbers of actions. Thirdly, although this algorithm is designed to work only on bandit problems with boolean reinforcement, (i.e. there must be no state, and no unbalanced or delayed reinforcement) the general method should be extendable to problems with unbalanced reinforcement[12] and hence to any

[12] In the manner of Kaelbling's interval-estimation

method where a value function on the state space is available.

Acknowledgements

Thanks go to Gillian Hayes, John Hallam and Martin Westhead for many valuable discussions on the topics herein, and for comments and advice on the production of the paper. Jeremy Wyatt is in receipt of SERC Research Studentship No.92314758.

References

[1] A. G. Barto. Connectionist learning for control. In W. T. Miller R. S. Sutton and P. J. Werbos, editors, *Neural Networks For Control*, pages 5–58. MIT Press, 1990.

[2] Rodney A. Brooks and Maja J. Mataric. *Real robots, real learning problems*, chapter 8, pages 193–214. Kluwer Academic Publishers, 1993.

[3] David Chapman and Leslie Pack Kaelbling. Input generalisation in delayed reinforcement learning: an algorithm and performance comparison. In *Proceedings of International Joint Conference on Artificial Intelligence*, 1991.

[4] Pawel Cichosz. Truncating temporal differences: on the efficient implementation of td(λ) learning. *Journal of Artificial Intelligence Research*, 2:287–318, January 1995.

[5] Marco Colombetti and Marco Dorigo. Training agents to perform sequential behaviour. Submitted to the Journal of Evolutionary Computation, September 1993.

[6] P. Dayan and G. Hinton. Feudal reinforcement learning. In S.J. Moody J.E., Hanson and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 5*, pages –. Morgan Kaufmann, 1993.

[7] Marco Dorigo and Marco Colombetti. Robot shaping: Developing situated agents through learning. Technical Report TR-92-040 (revised), International Computer Science Institute, April 1993.

algorithm [9], so that the confidence intervals and tests are conducted based on the mean reinforcement.

[8] Jonas Karlsson JoshTenenberg and Steven Whitehead. Learning via task decomposition. In *From Animals to Animats 2: Proceedings of the 2nd International Conference on the Simulation of Adaptive Behaviour*, pages 337–343. MIT Press, 1992.

[9] Leslie Pack Kaelbling. *Learning in Embedded Systems*. PhD thesis, Dept of Computer Science, Stanford, 1990.

[10] Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: preliminary results. In *Machine Learning: Proceedings of the 10th International Conference*, pages 167–173, 1991.

[11] Long-Ji Lin. Programming robots using reinforcement learning and teaching. In *Proceedings of the ninth national conference on artificial intelligence*, pages 781–786. AAAI Press/MIT Press, 1991.

[12] Long-Ji Lin. Hierachical learning of robot skills by reinforcement. In *Proceedings of the IEEE International Conference on Neural Networks 1993*, pages 181–186, 1993.

[13] Long-Ji Lin and Tom M. Mitchell. Reinforcement learning with hidden states. In *Proceedings of the Second International Conference on the Simulation of Adaptive Behaviour*. MIT Press, 1992.

[14] P. Maes and R.A. Brooks. Learning to coordinate behaviours. In *Proceedings of the 8th National Conference on AI*, 1990.

[15] S. Mahadevan and J.H. Connell. Automatic programming of behaviour-based robots using reinforcement learning. Research Report RC 16359 (72625), IBM Research Division, July 1990.

[16] M. Mataric. Reward functions for accelerated learning. In W.W. Cohen and H. Hirsh, editors, *Machine Learning: Proceedings of the Eleventh International Conference*, pages 181–9. Morgan Kaufmann, February 1994.

[17] Matthew A.F. McDonald and Philip Hingston. Approximate discounted dynamic-programming is unreliable. Technical Report 94/6, University

of Western Australia, Dept of Computer Science, October 1994.

[18] Andrew W. Moore. Knowledge of knowledge and intelligent experimentation for learning control. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 683–8, July 1991.

[19] Ulrich Nehmzow. *Experiments in Competence Acquisition for Autonomous Mobile Robots*. Ph.d. thesis, Department of Artificial Intelligence, Edinburgh University, 1992.

[20] Mark Pendrith. On reinforcement learning of control actions in noisy and non-markovian domains. Tech report UNSW-CSE-TR-9410, University of New South Wales, School of Computer Science and Engineering, August 1994.

[21] Jing Peng and Ronald J. Williams. Incremental multi-step q-learning. In W.W.Cohen and H.Hirsh, editors, *Machine Learning: Proceedings of the 11th International Conference*, pages 226–232, 1994.

[22] Satinder P Singh. The efficient learning of multiple task sequences. In S. Hanson J. Moody and R. Lippman, editors, *Advances in Neural Information Processing Systems 4*, pages 251–258. Morgan Kaufmann, 1992.

[23] R.S. Sutton. *Temporal Credit Assignmenment in Reinforcement Learning.* PhD thesis, University of Massachusetts, School of Computer and Information Sciences, 1984.

[24] R.S. Sutton and S.P. Singh. On step-size and bias in temporal difference learning. In *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pages 91–96, 1994.

[25] Sebastian B. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, School of Computer Science, January 1992.

[26] Steven D. Whitehead and Dana Ballard. Learning to perceive and act. Technical Report TR-331 (revised), University of Rochester, Department of Computer Science, June 1990.