Multitask Reinforcement Learning on the Distribution of MDPs

Fumihide Tanaka
Department of Computational Intelligence and Systems Science,
Tokyo Institute of Technology

Masayuki Yamamura
Department of Computational Intelligence and Systems Science,
Tokyo Institute of Technology

Abstract

In this paper we address a new problem in reinforcement learning. Here we consider an agent that faces multiple learning tasks within its lifetime. The agent's objective is to maximize its total reward in the lifetime as well as a conventional return in each task. To realize this, it has to be endowed an important ability to keep its past learning experiences and utilize them for improving future learning performance. This time we try to phrase this problem formally. The central idea is to introduce an environmental class, BV-MDPs that is defined with the distribution of MDPs. As an approach to exploiting past learning experiences, we focus on statistics (mean and deviation) about the agent's value tables. The mean can be used as initial values of the table when a new task is presented. The deviation can be viewed as measuring reliability of the mean, and we utilize it in calculating priority of simulated backups. We conduct experiments in computer simulation to evaluate the effectiveness.

1 Introduction

Reinforcement learning (RL) is a general framework of learning through trial and error. A learner called the agent interacts with its outside world, the environment, acquiring rewards. Like the discipline of dogs or cats, we can let the agent obtain some action strategies gradually through the interaction process. The point is that there we need not to design a perfect teacher, that is, the agent can learn autonomously from the environmental responses. Further, the framework is known to be able to deal with such difficulties as uncertainty in the environment and delayed rewards. These features attract researchers in many fields including robotics.

Basically, the RL framework is designed for solving a single learning task. The problem is defined only in the task, and such a concept as reuse of past learning experiences is not considered in it. But there are lots of cases where multiple tasks are imposed on the agent. Here is one example: Imagine a cleaning robot in hotels. The task of the robot is to clean up a room everyday after the customer checks out. To make the problem simple, here we assume that only one room is cleaned up in a day. The structures of all rooms are near the same, but sometimes there happen cases where customers change the layouts arbitrarily. Under these settings, the robot has to finish its task as fast as it can everyday. In this problem, there are multiple learning tasks through years. And it is essential for the agent to keep its past experiences and utilize them in the current learning task to solve it faster. Generally in RL, there costs a large number of learning trials, and then exploiting them seems to be promising as it could reduce the number significantly. But there has not been so many studies dealing with multiple tasks in RL, especially when the number of tasks is in large order (such cases as the cleaning robot above). In this paper, we try to formulize the problem on a basic RL framework, and then consider an appropriate learning method. Here multiple tasks are defined through the distribution of MDPs (Markov Decision Processes), and a learning agent maintains and exploits value statistics to improve its RL performance.

The structure of this paper is as follows: First, we briefly review the basic framework of RL (Section 2). Then, we build our arguments about multitask RL (MTRL) in Section 3. To design the MTRL agent, transfer of past learning experiences is at the core of its ability, and this is discussed in Section 4. Following this, experiments are conducted in computer simulation to evaluate proposed methods (Section 5). And finally, in Section 6 we summarize this paper with discussion about ongoing studies.

2 Basic Background [2, 7]

In this section we briefly review the basic framework of RL on which we will build our arguments.

There is a learning agent that interacts with its outside (environment) at some discrete time scale. At each time step, the agent selects an action $a_t \in \mathcal{A}$ based on its observation of the environment's state $s_t \in \mathcal{S}$. In response to the a_t, the environment returns a numerical reward r_{t+1} and the next state s_{t+1} to the agent as a consequence of a state-transition. Basically, the environmental dynamics is modeled through a finite Markov Decision Process (MDP). A particular finite MDP is defined by four sets $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} >$, where $\mathcal{P}$ is a set of transition probabilities $P^a_{ss'} = \Pr\{s_{t+1} = s' \mid s, a\}$, and $\mathcal{R}$ is a set of reward functions $R^a_{ss'} = E\{r_{t+1} \mid s, a, s'\}$. The agent's objective is to find optimal policy $\pi(s, a)$, which is a mapping from states to probabilities of taking each possible action. It is updated through learning to maximize the expected discounted future reward from each state s:

$$v^\pi(s) = E\left\{ r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \cdots \middle| s_t = s, \pi \right\} \qquad (1)$$

where γ is a discount rate. This $v^\pi(s)$ specifies the value of a state s under policy π, and v^π is called the value function for policy π. Based on this,

$$v^*(s) = \max_\pi v^\pi(s) \qquad (2)$$

```
Initialize q(s,a) arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from q
    Take action a, observe r, s'
    q(s,a) ← q(s,a) + α[ r + γ max_a' q(s',a') − q(s,a) ]
    s ← s';
  until s is terminal
```

Figure 1: Q-learning (α: learning rate) [11]

can be defined, and v^* is called the optimal value function. By using this, delayed rewards can be dealt with, and this is one feature of RL.

These value functions can be estimated through trial and error. That is, the agent can learn them directly from raw experience without a model of the environmental dynamics. Various approaches were proposed to estimate them, and we will later use Q-learning[11] among them. Figure 1 explains its basic algorithm. In Q-learning, the action-value $q(s, a)$ is used instead of $v(s)$. This method is known to be able to estimate the optimal value function under some assumptions, and is widely used in the literature of RL fields.

3 Formulizing MTRL

Generally speaking, it is natural for an intelligent agent to face multiple learning tasks within its lifetime. In contrast to conventional machine learning techniques that deal with single learning task, recently there have been some studies for multitask-oriented learning.[8, 10] Among them, a modular-approach such as Singh's CQ-learning[5] is one promising way, reusing past learning experiences by switching value function modules obtained before. But this approach is inappropriate for our problem domain where there are relatively many number of tasks, as huge amounts of memory are needed in preparing the modules. As another direction, there are lots of studies in learning maps by a navigation robot, some of which use RL techniques.[9, 1] If the environment is static or nearly static, then the maps can be reused in a new task there. This idea is in a sense near to our thought, but usually it is difficult for this approach to discuss how much environmental change it can deal with. To overcome this, we consider it very important to phrase the problem formally. In this section we describe this matter. Based on the arguments here, we will present a RL method exploiting past learning experiences in the next section.

3.1 A Task and the Distribution of Tasks

First, we define a task by MDP and its running time τ, that is, by specifying five sets $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \tau >$. As we explained in Section 2, a set of optimal values $v^*(s)$ can be calculated if we specify the dynamics of a MDP. Therefore, we can say that every task has its own optimal value set $v^*(s)$ respectively.

Next, we consider the distribution of tasks. In other words, we define a class over MDPs from which we can sample a task independently. To describe the property of the class, we use a set of $V^*(s)$ which denotes the distribution of $v^*(s)$. Specifying the $V^*(s)$ can be seen as setting a relationship between tasks. We can represent environmental features by using it. And we name the class as MDPs with Biased-Values (BV-MDPs).

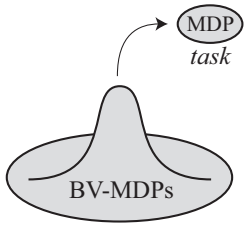

Figure 2: A task and the distribution of tasks

In this paper, we assume that every $V^*(s)$ is independent about s. Generally speaking, there seem to be cases where this condition is invalid, but currently it is very hard to develop a universal method that doesn't need any assumptions.

3.2 The Total Reward Through a Lifetime

As we mentioned in Section 2, the purpose of an agent in RL is to maximize the expected return in every state. In addition to this, here we have to consider the total reward through the agent's lifetime. Formally it is described as follows:

$$TR = \sum_{i=1}^{N} \int_{t_{i-1}}^{t_{i-1}+\tau_i} r_t \, dt \quad \left(t_i = \sum_{j=1}^{i} \tau_j, \; t_0 = 0 \right) \qquad (3)$$

where N is the total number of tasks, and τ_i is running time of task i. These two criterions (1)(3) imply a significant problem about their time scales. The former assumes infinite time though the latter finite. It suggests the need for RL in finite time. Although this is a big theme, in this paper we assume that the time scale of the former (1) is enough small compared with τ_i.

3.3 The MTRL Problem

Based on the arguments so far, now we can formulize the MTRL problem. We consider an agent that faces multiple learning tasks through its lifetime, utilizing past experiences for improving its current learning performance (Figure 3). The agent's objective is to maximize its total reward (3) in addition to the conventional criterion (1) in each task. A task is defined as an independent sample from BV-MDPs, and each is presented to the agent one by one continuously until the end of its lifetime.

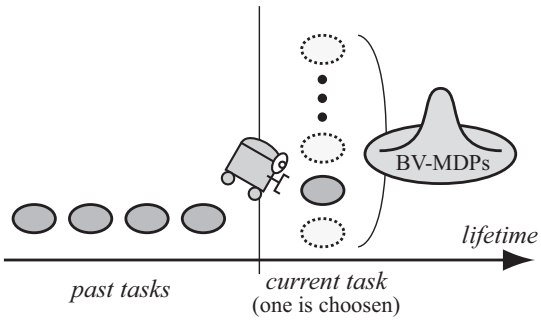

Figure 3: The MTRL agent

4 Designing the MTRL Agent

The ability to transfer past learning experiences and utilize them for improving future learning performance is at the core of the MTRL problem. In this section we discuss how it should be realized.

As we mentioned in the previous section, one objective of the agent in MTRL was to maximize its total reward (3). Currently we cannot calculate the equation directly. Then instead of it, we aim for maximizing the sub-total reward below in every task.

$$STR = \int_{t_{i-1}}^{t_{i-1}+\tau_i} r_t \, dt \quad \left(t_i = \sum_{j=1}^{i} \tau_j, \; t_0 = 0 \right) \tag{4}$$

This is valid because every task is sampled independently.

Next we explain what kind of information the agent maintains as past learning experiences, and then we describe how it is exploited in RL of current task.

4.1 Maintaining Value Statistics

What should be transferred between tasks as past experiences? The most straightforward way would be keeping the learning results, that is, value tables. In the MTRL problem, there are relationships between tasks that are defined by BV-MDPs. Therefore, there are relationships also in their value tables. We try to extract them from value statistics. After finishing each task, the agent can easily store the learning results. It calculates the statistical information such as the mean and deviation that are noted as follows:

$$\overline{Q}(s,a) = \frac{1}{n} \sum_{i=1}^{n} q(s,a)_i \tag{5}$$

$$s_{Q(s,a)} = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (\, q(s,a)_i - \overline{Q}(s,a) \,)^2} \tag{6}$$

(for all s, a) where i indicates a task index, and n is the number of previously solved tasks. These quantities represent features in the environment that are common to all of the tasks. If there is a state-action pair whose $s_{Q(s,a)}$ is relatively small compared with others, then its $\overline{Q}(s,a)$ can be viewed as more reliable than others.

4.2 Exploiting Value Statistics

To maximize (4), the agent has to estimate the optimal value function as fast as possible. In the situation where the learning speed is crucial, usually a model-based approach (Figure 4) is used in RL.[6, 7] Here, in addition to the normal learning process (by real experiences), a model and simulated experiences in it are used to estimate its values, and then its learning speed is accelerated. The basic procedure of model-based Q-learning is shown in Figure 5. The model is maintained here simply by keeping counts of the number of times each state-action pair has been experienced and of what the next states were.

Below we will discuss how previously obtained value statistics are combined with the model-based RL.

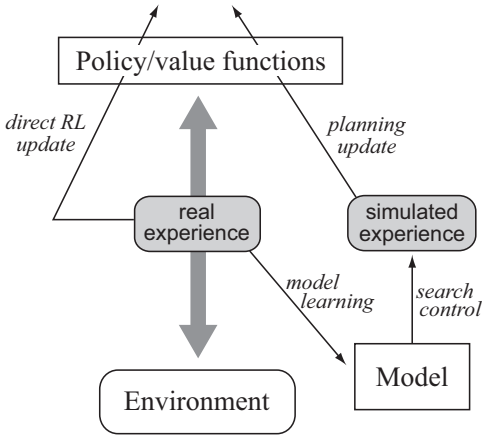

Figure 4: The general Dyna architecture [6]

```
Initialize q(s,a) and Model(s,a) for all s,a
Do forever:
  simple Q-learning
  Model(s,a) ← s', r
  Repeat X times:
    s ←random previously observed state
    a ←random action previously taken in s
    s', r ← Model(s,a)
    q(s,a) ← q(s,a) + α [ r + γ max_a' q(s',a') − q(s,a) ]
```

Figure 5: Dyna-Q algorithm [6]

Exploiting $\overline{Q}$

Usually in RL, initial values are set randomly or zero, and there is no specific standard that is considered to be the best though they affect the learning-speed very much. In the MTRL problem, however, the mean value $\overline{Q}(s,a)$ can be employed easily, and this is the best choice for them.

$$q_0(s,a) = \overline{Q}(s,a), \; \textit{for all } s,a \tag{7}$$

Exploiting s_Q

To make model-based RL effective, there are two keys in its way of simulated backups. As the number of them is fixed at X times per one real trial, it is important to let one simulated backup *as large as* and *as precise as* the agent can.

For the former theme, that is about the size of each backup, *Prioritized Sweeping* (PS) was proposed.[3, 4] In the basic algorithm shown in Figure 5, simulated experiences are obtained by random backups. But here in the PS, the order is determined by using a priority p below, and backups are done more effectively.

$$p = \left| \, q^T - q^C \, \right| \tag{8}$$

where $q^C = q(s,a)$ and q^T is various by the difference of the RL algorithm. For example, in case of Q-learning, $q^T = r + \gamma \max_{a'} q(s',a')$, or in case of SARSA[7], $q^T = r + \gamma q(s',a')$. And then, backups having larger p values are prioritized.

Now, we consider the latter theme, which is about accuracy of each backup. As we mentioned, each $\overline{Q}(s,a)$ has its own reliability represented by $s_{Q(s,a)}$ in a sense.

We make use of this feature. Many RL algorithms such as Q-learning are what we call *bootstrapping* method; each value is estimated by using other values which themselves are also estimates. Therefore, in the middle of learning phase, the values are not always accurate, that is, there often happen cases where they are far from optimal values. Now we can use the deviation information s_Q to decrease the number of wrong backups caused by the inaccurate values explained above. This idea is realized by introducing a new priority metric designed for the MTRL problem:

$$\begin{aligned} p_E &= E\left[\left(Q^T - Q^C \right)^2 \right] \\ &\simeq \left(\overline{Q^T} - \overline{Q^C} \right)^2 + \left(s_{Q^T} \right)^2 + \left(s_{Q^C} \right)^2 \end{aligned} \tag{9}$$

(See Appendix) By using this p_E, the deviation information about each $q(s,a)$ is properly exploited, and more precise estimation can be realized. Through the learning of a task, a queue is maintained of every state-action pair whose $q(s,a)$ would change nontrivially if backed up, prioritized by the size of the change. When the top pair in the queue is backed up, the effect on each of its predecessor pairs is computed. If the effect is greater than some small threshold, then the pair is inserted in the queue with the new priority. This queue maintenance is the same as the original PS.

Value statistics are updated at the end of each task's learning phase. In contrast to the mean information $\overline{Q}$ which is used only when each task's learning starts, the deviation s_Q has been hired through the task all the way. Statistics are *universal* information in that they are obtained through many past learning tasks. Therefore, if one wants to focus on the performance of convergence within a task, the information should be shifted to more *local* type from the universal one. To do this, we prepare another deviation s'_Q that represents fluctuation of q within a task, and is updated in every step of learning. (s'_Q is set to 0.0 at the beginning of each task's learning) Then, the weighted sum of s_Q and s'_Q is used, changing the weights as shown in Figure 6. By using the sum, the deviation is shifted gradually to that of the *local* one. Currently a slope parameter β is determined by trial and error.

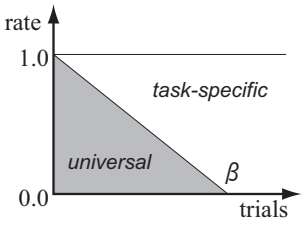

Figure 6: Adaptation of each deviation

5 Experiments

5.1 Objective

In this section we set the MTRL problem by using computer simulation. We conduct some experiments on it and evaluate the effectiveness of our method presented in Section 4. There are two points for a clear understanding about the results:

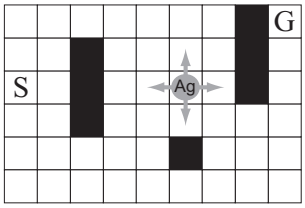

Figure 7: A simple maze problem

- We let the agent's lifetime enough long for acquiring accurate statistics. Then, in new tasks, we compare results by our method that exploits value statistics and those by normal PS.
- The purpose of RL agent is to maximize some criterions about reward. But, when we evaluate its performance, such a standard like an average reward is sometimes inappropriate because it is affected by the differences of the agent's policy (action selection). Therefore we evaluate our method by measuring accuracy of value estimation. We calculate the mean-squared error (MSE) between estimated values and the optimal values to judge the accuracy. This criterion enables us to do fair and stable evaluations. As we explained in Section 2, the value of a state is a metric which generalizes acquiring rewards, and estimating it faster leads to more rewards in τ period.

5.2 Plan

At base, the problem is designed on Sutton's simple maze.[6] Figure 7 illustrates it. The 46 cells of the grid correspond to the states. At each cell, four actions are possible: north, south, east, and west, which *stochastically* (originally deterministic) cause the agent to move one cell in the corresponding direction on the grid. The transition-probabilities are determined when a task (a maze) is given to the agent, which will be explained later. The agent cannot move toward outside and obstacles. Reward is zero on all transitions, except those into the goal state, on which it is +100. After reaching the goal state, the agent returns to the start state to begin a new episode. This problem can be represented by MDP, and there the agent aims for acquiring the minimum-length route towards the goal through trial and error. In our experiments, γ is set to 0.95 and α to 0.1 which are the same as [6, 4].

Consider a situation where multiple maze problems (MDPs) are given to the agent one by one. When a new task (MDP) is given, each transition-probability is sampled from a normal distribution. Here we form BV-MDPs by $< \mathcal{S}, \mathcal{A}, \mathcal{DP}, \mathcal{R} >$ where $\mathcal{DP}$ indicates the set of the distribution.

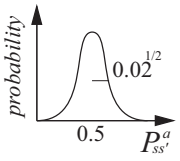

Figure 8: A probability distribution of $P^a_{ss'}$

At first, 100 tasks are sampled to calculate value statistics. Then, 10 new are sampled as well to investigate the effects of proposed methods. Results are evaluated by the averaged mean-squared error (averaged MSE) between $q(s,a)$ and $q^*(s,a)$. Regarding the agent's action selection (exploring strategy), we use ϵ-*greedy* selection where a random action is selected with probability ϵ, and in other cases an action whose value is the highest is selected.[7] In our experiments, ϵ is set to 0.5.

Two types of experiments are conducted. In Exp.1 we fix the number of states whose transition is stochastic to 25. They are determined randomly. (Experiments are done 10 times as a whole, and we evaluate the averaged results) Each transition probability represents that of success in corresponding state-transition, and in other case (: failure) the state is unchanged. Regarding the states whose transition is stochastic, all transition probabilities are sampled from a normal distribution whose mean is 0.5 and variance 0.02. In Exp.1 we focus on investigating effects of introducing $\overline{Q}$ and s_Q respectively.

Next in Exp.2, we change the number of states whose transition is stochastic. In addition to the case of 25 above, here we add those of 5 and 45. All other settings are the same as Exp.1. To reduce the number means that the environment becomes more static, and vice versa. In the former case, it is expected that the effects of introducing $\overline{Q}$ would be dominant. In Exp.2, we pay attention to how the situation changes as the environment becomes more stochastic.

5.3 Results: Exp.1

Figure 9 shows the effect of introducing $\overline{Q}$. It is comparison between a learning curve by a normal PS method and that by a PS with $\overline{Q}$ in its initial values. We can find here a typical result about using $\overline{Q}$ as initial bias. In case of using it, the performance is getting worse temporary at the beginning stage, and then it is getting better. If this experiment is continued longer, the performance is converged below the start point. There is an offset between two lines, and because of this, the biased method converges faster. This tendency was observed in all experiments conducted in this paper.

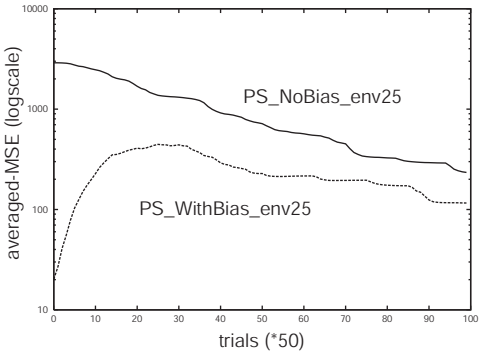

Figure 9: Learning curves for simple PS (PS_NoBias) and it with the initial bias $\overline{Q}$ (PS_WithBias).

Next, Figure 10 shows the effect of introducing s_Q. As a reference, there plotted the result of biased PS shown in Figure 9. Then we compare three results with it which are by proposed method using both $\overline{Q}$ and s_Q. (β is different in three ways, 10000, 20000, 30000; each

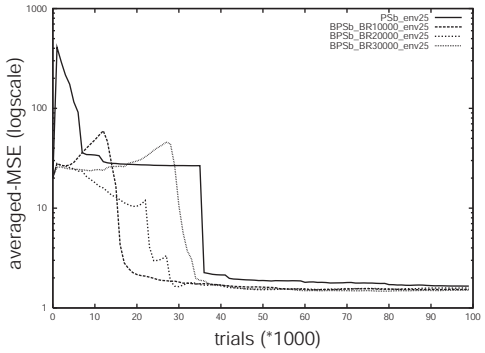

Figure 10: Learning curves for PS with $\overline{Q}$ only (thick line) and with $\overline{Q}$ and s_Q (dotted lines).

from the left of the dotted lines in Figure 10) They are all better than the reference, especially in the beginning phase. This is because by introducing s_Q the learning process is done in such a way that dispersion of reliability in $\overline{Q}$ is considered.

5.4 Results: Exp.2

Figure 11 summarizes the results of Exp.2. Experiments were conducted in three environmental settings. The number of states whose transitions are stochastic is different in three ways: 5, 25, 45. The horizontal axis means them. The vertical axis plots integral values of averaged MSE until 100000 steps. (: The smaller value means the better performance.) Three lines indicates the results by the normal PS which doesn't use any past learning information (PS), the biased PS which exploits only $\overline{Q}$ (PS_WithBias), and the proposed method which exploits both $\overline{Q}$ and s_Q (PSVS). We can see that as the environment is getting more stochastic, the effect of $\overline{Q}$ is weakened. In contrast to this, the effect of s_Q seems to be getting better as the slope of the line of PSVS is more gentle compared with that of PS_WithBias. As a consequence, we can view that the role of s_Q complements that of $\overline{Q}$.

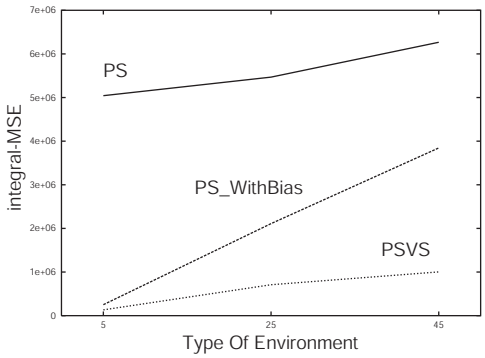

Figure 11: Performance comparison between three methods in three types of environments.

6 Conclusion

In this paper, we tried to expand the framework of RL into dealing with multiple learning tasks. We formulized

the MTRL problem on the distribution of MDPs, and presented a model-based learning method exploiting value statistics. By using it, the agent could extract relationships between tasks (which were defined with an environmental class, BV-MDPs), and utilize them to improve its RL performance. Experiments in computer simulation showed its effectiveness and properties of proposed methods.

Ongoing research topics are mainly in two directions. First, we're considering value statistics where each value is represented not by naive tables but some function approximators especially those in linear forms. Second, as we described in 3.2, we need to develop a metric that can measure or evaluate the performance of RL in finite time scales. We're trying to estimate the number of learning trials by the function of s_Q.

References

[1] J. del R. Mill'an. Rapid, Safe, and Incremental Learning of Navigation Strategies. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 26:408–420, 1996.

[2] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[3] A. W. Moore and C. G. Atkeson. Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time. *Machine Learning*, 13:103–130, 1993.

[4] J. Peng and R. J. Williams. Efficient Learning and Planning Within the Dyna Framework. *Adaptive Behavior*, 1(4):437–454, 1993.

[5] S. P. Singh. Transfer of Learning by Composing Solutions of Elemental Sequential Tasks. *Machine Learning*, 8:323–339, 1992.

[6] R. S. Sutton. Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *Proceedings of the 7th International Conference on Machine Learning*, pages 216–224, 1990.

[7] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[8] F. Tanaka and M. Yamamura. An Approach to Lifelong Reinforcement Learning through Multiple Environments. In *Proceedings of the 6th European Workshop on Learning Robots*, pages 93–99, 1997.

[9] S. Thrun and K. Möller. Active Exploration in Dynamic Environments. In *Advances in Neural Information Processing Systems 4*, pages 531–538, 1992.

[10] S. Thrun and L. Pratt. *LEARNING TO LEARN*. Kluwer Academic Publishers, 1998.

[11] C. J. C. H. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8:279–292, 1992.

Appendix

$$
\begin{aligned}
p_E &= E\left[\left(Q^T - Q^C\right)^2\right] \\
&= E\left[\left(Q^T\right)^2\right] - 2\cdot E\left[Q^T Q^C\right] + E\left[\left(Q^C\right)^2\right] \\
&\simeq \left(s_{Q^T}\right)^2 + \overline{Q^T}^2 - 2\cdot \overline{Q^T}\,\overline{Q^C} + \left(s_{Q^C}\right)^2 + \overline{Q^C}^2 \\
&= \left(\overline{Q^T} - \overline{Q^C}\right)^2 + \left(s_{Q^T}\right)^2 + \left(s_{Q^C}\right)^2
\end{aligned}
$$