Reinforcement Learning for Locomotion of a Two-linked Robot Arm

Hajime Kimura and Shigenobu Kobayashi *

**Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology, gen@fe.dis.titech.ac.jp, kobayasi@dis.titech.ac.jp*

Abstract. Reinforcement learning (RL) tasks of real world can be characterized by two difficulties: Function approximation and hidden state. For large and continuous state or action space, RL agents have to incorporate RL algorithms with some form of generalization. One way to do it is to use general function approximators to represent value function or control policies. Hidden state problems arise in the case that RL agents cannot observe the state of the environment perfectly owing to noisy or insufficient sensors, partial information, etc. Partially observable Markov decision processes (POMDPs) are an appropriate model for hidden state problems. We have presented a RL algorithm in POMDPs, that is based on a stochastic gradient ascent. It uses function approximation to represent a stochastic policy, and updates the policy parameters. In this paper, we apply the algorithm to a robot control problem, and show the features in comparison with Q-learning and Jaakkola et al.'s method. The result shows that the gradient method achieved good results in terms of handling hidden state, performance sensitivity to increasing the number of variables in the observation space, and possibility to use together with function approximation.

1 Introduction

Robotics are potentially a fruitful area for the application of reinforcement learning (RL) techniques. RL is a promising method for robots to improve the behavior themselves or make up for lack of knowledge about the tasks. RL can be seen as an online learning of an input-output mapping to maximize some performance measure. Since the controller, called an agent, may not know the environment ahead of time, the learning occurs through trial-and-error experimentation with the environment. For every state and every action, a reward is provided to the agent. But, on sequential decision making tasks, the rewards may be intermittent and considerably delayed. Many theoretical works in RL adopt Markov decision processes (MDPs) for environment models. MDP is an excellent model for delayed reinforcement tasks with uncertainty and discrete state transition. TD method (Sutton 88) and Q-learning (Watkins et al. 92) are representative of dynamic-programming-based RL algorithms to solve MDPs. Unfortunately, it needs a non-realistic assumption that RL agents observe the state of the environment perfectly. RL tasks of real world can be characterized by two difficulties: *Function approximation* and *hidden state* problems.

For large and continuous state or action space, agents have to incorporate RL algorithms with some form of generalization. One way to do it is to use general function approximators, such as neural networks, to represent value function or control policies. However, the generalization often causes non-Markovian effects. It can be caused easily in the case that continuous state-spaces are partitioned into multi-dimensional coarse grids (Moore et al. 95). *Perceptual aliasing* (Whitehead et al. 95) can be seen as a similar non-Markovian problem.

Hidden state problems arise in the case that agents cannot observe the state of the environment perfectly owing to noisy or insufficient sensors, partial information, etc. Partially observable MDPs (POMDPs) (Singh et al. 94) are an appropriate model for hidden state problems.

The work described in this paper is focused on a RL algorithm that learns memory-less policies in POMDPs with using function approximation. It is motivated by the following principles:

- Robots (agents) should learn tasks themselves. It would be very easy for us to give the reward function to show the robots what to do. The robots have to learn how to do the tasks.
- Robots should be able to handle uncertainty of perceptions, actions, and rewards.
- Robots should be able to learn within a given finite memory and given finite computational power. In many practical situations, the computational resources required to find an optimal solution are infeasible or economically undesirable.
- Robots should use some function approximators to generalize state space. It is concerned with the bound of resources.
- Robots should be able to handle non-Markovian problems owing to hidden state or function approximation.
- We would prefer incremental simple algorithms, like as TD or Q-learning. Also we want robots to learn as few trials as possible.

We take up a small but significant RL task in robotics that can be seen as a RL in POMDPs with function approximation. We have presented a RL algorithm based on a stochastic gradient ascent (Kimura et al. 96), that can be used to try to improve a stochastic policy in POMDPs. It uses function approximator to represent a stochastic policy, and updates the policy parameters. We apply the algorithm to the task, and show its features in comparison with Q-learning and Jaakkola et al.'s method.

2 The Problem Domain

Consider a planar two-link manipulator in a gravitational environment as shown in Figure 1. The agent does not know the environment ahead of time. At each time step, the agent observes noisy sensor-readings of the joint angles, and outputs turning direction of the joint motors. Since we want the robot to move to the front, the immediate reward, that is given to the agent in every time step, is defined as the length that the body moved forward in the current step. When the body moves backward, a negative reward is given proportionally to

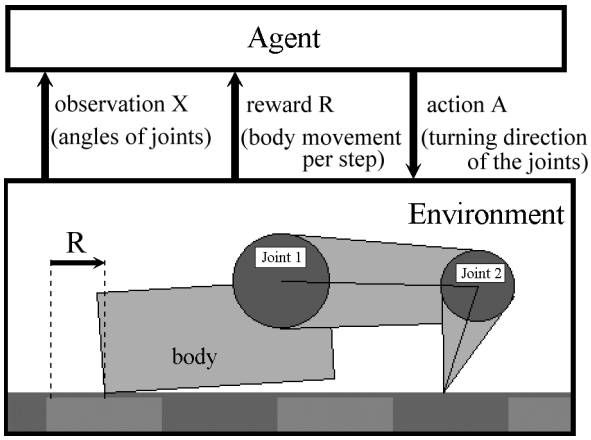

Figure 1: Learning a motion to move to the front.

the distance moved. In order to move the body to the front, the arm should be controlled as if it is crawling. Through trial and error, the agent has to learn such a control policy that maximizes a reward function. The environment is considered to be continuous state POMDPs. It has the following 3 difficulties.

(1) Delayed Reinforcement: The agent should learn the movement as Figure 2 (1). When the arm is touching the ground and moving as A→B→C, then positive immediate reward is given, so the agent can learn this movement easily. On the other hand, the movement C→D→A gives zero immediate reward. If the arm is moving wastefully like

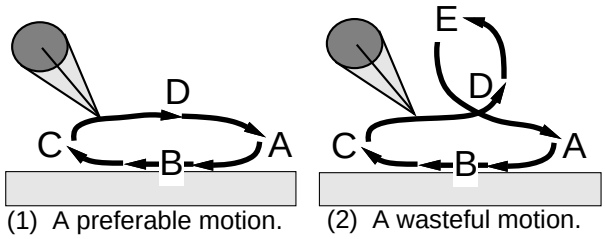

Figure 2: Crawling motions of the robot arm.

C→D→E→D→A as shown in Figure 2 (2), then the agent is given the same zero immediate reward. Accordingly, the agent has to learn adequate actions by delayed reinforcement.

(2) Hidden state problems arise from imperfect state observation owing to noisy or insufficient sensors. In this paper, POMDPs represent the hidden state problems.

(3) Function approximation: In order to gen-

eralize large and continuous state space, the agent has to use function approximators. There are many ways to do it; neural networks, fuzzy logic systems, etc. The simplest way is quantizing: partitioning the continuous state space into multidimensional grid, and treating each cell as an atomic object. The grid approach has a number of dangers. Increasing the resolution costs computational resource and physical amount of data exponentially (Moore et al. 95). In addition, the cells that are roughly partitioned have hidden state. The task in Figure

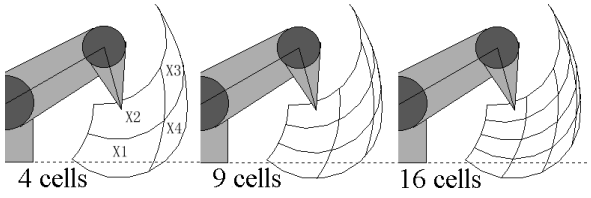

Figure 3: Partitioning continuous sensor-readings into discrete cells.

1 has the same POMDP problem. Assume that the continuous sensor-readings are partitioned into 4 discrete cells as shown in Figure 3. The observation $X1$ includes two hidden states: one is that the arm top is touching the ground, the other is not.

The difficulties of this example can be theoretically generalized to a RL problem in POMDPs with function approximation.

3 Stochastic Gradient Ascent

3.1 Previous Work

Most previous RL algorithms for non-Markovian decision tasks have adopted approaches that combine some form of state estimation with Q-learning. The state estimator attempts to generate Markovian state representation, thereafter Q-learning module finds value function and optimal policies. Recurrent-model (Lin et al. 92), (Whitehead et al. 95), Perceptual Distinctions Approach (Chrisman 92) and Utile Distinction Memory (McCallum 93) are based on building a predictive model of the environment. Window-Q (Lin et al. 92), (Whitehead et al. 95) and Utile Suffix Memory (McCallum 95) use a history of recent observations as a state representation. Recurrent-Q (Lin et al. 92) (Whitehead et al. 95) uses recurrent neural networks. We should notice that most previous approaches adopt Q-learning to find policies over internal state space. The first issue we must consider is that all these techniques cannot guarantee the state estimator to build a perfect Markovian state representation, especially in the resource-bounded agents. The second is a non-Markovian effect owing to function approximation to handle the enormous internal state variables. This is not to deny using the above state estimators, but that in some cases the agents should adopt the other adequate RL components for learning memory-less policies in POMDPs, instead of Q-learning.

As discussed in (Singh et al. 94), memory-less *stochastic policies* can be considerably better than memory-less deterministic policies in the case of POMDP's. (Jaakkola et al. 94) have proposed a policy improvement method based on a Monte-Carlo policy evaluation in POMDPs. The algorithm, we call JSJ method, operates in the space of stochastic policies. A drawback of JSJ method is that the agents cannot use arbitrary function approximator except for quantizing.

We have presented a RL algorithm in POMDPs, that is based on a stochastic gradient ascent on discounted reward. It uses function approximator to represent a stochastic policy, and updates the policy parameters. We believe it is the most hopeful approach in that case.

3.2 Function Approximation for Stochastic Policies

The objective of the agent is to form a stochastic policy, that assigns a probability distribution over actions to each observation, in order to maximize some reward function. In this paper, $\pi(a, W, X)$ denotes probability of selecting action a under the policy π in the observation X (Figure 4). The policy is represented by a parametric function approximator using an internal variable vector W. The agent can improve the policy π by means of modifying W. In short, W corresponds to synaptic weights where the action selecting probability is represented by neural networks, or W means weight of rules in classifier systems. The advantage of the parametric notation of π is that computational bounds and mechanisms of the agent can be specified simply by the form of the function. It can provide a sound theory of learning algorithms for arbitrary types of agents.

3.3 General Form of the Algorithm

Figure 5 shows a general form of the algorithm based on a stochastic gradient ascent. The notation $e_i(t)$ in the 4th procedure is called *eligibility* (Williams 92), that is considered to be information

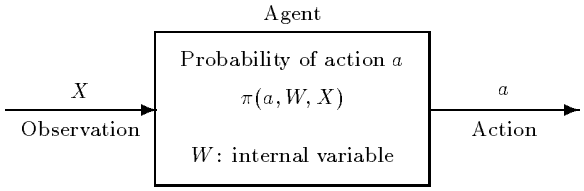

Figure 4: Stochastic policy; The agent can improve a policy π by modifying the vector W.

1. Observe X_t in the environment.
2. Execute action a_t with probability $\pi(a_t, W, X_t)$.
3. Receive the immediate reward r_t.
4. Calculate $e_i(t)$ and $D_i(t)$ as

$$e_i(t) = \frac{\partial}{\partial w_i} \ln\Big(\pi(a_t, W, X_t)\Big) ,$$
$$D_i(t) = e_i(t) + \gamma D_i(t-1) ,$$

where γ $(0 \leq \gamma < 1)$ denotes the discount factor, and w_i does the i^{th} component of W.

5. Calculate $\Delta w_i(t)$ as

$$\Delta w_i(t) = (r_t - b)\, D_i(t) ,$$

where b denotes the reinforcement baseline.

6. Policy Improvement: update W as

$$\Delta W(t) = (\Delta w_1(t), \Delta w_2(t) \cdots \Delta w_i(t) \cdots) ,$$
$$W \leftarrow W + \alpha(1-\gamma)\Delta W(t) ,$$

where α is a nonnegative learning rate factor.

7. Move to the time step $t+1$, and go to step 1.

Figure 5: General form of the proposed algorithm.

about the executed action. $D_i(t)$ is a discounted running average of eligibility, called *eligibility trace*, that accumulates the agent's history of the executed actions. When a positive reward is given, the agent updates W by the procedure 5 and 6 as if the agent reinforces the probability of executed actions recorded in the eligibility trace. Some theorems have shown in (Kimura et al. 95) and (Kimura et al. 96), we have proved the following result in (Kimura97):

Theorem 1 *For all $t \geq 0$, $a \in \mathcal{A}$, $X \in \mathcal{X}$ and W, we assume that $|r_t|$ and $|\frac{\partial}{\partial w_i} \ln \pi(a, W, X)|$ are bounded from above. In ergodic POMDPs, when the agent keeps a stationary policy π, then ΔW follows*

$$\lim_{N\to\infty} \frac{1}{(1-\gamma)N} \sum_{t=0}^{N-1} \Delta w_i(t) = \sum_{s\in S} U^{\pi}(s) \frac{\partial}{\partial w_i} V_{\infty}^{\pi}(s), \quad (1)$$

where s denotes a state in the underlying MDP, $V_{\infty}^{\pi}(s)$ denotes expected discounted reward, $U^{\pi}(s)$ denotes the probability of occupying state s under π. The discount factor used in $V_{\infty}^{\pi}(s)$ is identical to γ used in the algorithm.

This result means that the average of $\Delta w_i(t)$ is equivalent to the gradient of the expected discounted reward biased by the state occupancy probability. It says that the agent can use this algorithm to change the policy in a direction for increasing discounted reward. It is interesting in the point that discounting (or forgetting) past experiences relates to discounting future rewards. Although there are several issues about defining the optimality of discounted reward criteria in POMDPs (Singh et al. 94), it can be used to try to find an approximately optimal policy with respect to the average reward by moving the discount factor γ close to 1. This algorithm can be seen as an extension of episodic REINFORCE algorithms (Williams 92).

4 Experiments

4.1 Details of the Robot

We specify the details of the robot shown in Figure 1. The upper arm length is 34, the fore arm length is 20. The joint of the body and the arm is located on the height = 18, the horizontal distance = 32 from the body's bottom left corner. The angle from horizontal of the first joint connected to the upper arm is constrained such that $-4 \leq Joint1 \leq 35$ degree. The angle of the second joint from the axis of the upper arm to the fore arm is constrained such that $-120 \leq Joint2 \leq 10$ degree. The motor of the first joint moves the arm to 12 ± 4 degree in the commanded direction with uniform distribution. Also the motor of the second joint moves 12 ± 4 degree to the commanded direction. When the arm is touching the ground, the arm does not slip, but the body slips easily.

4.2 Implementation of the Agent

4.2.1 Stochastic Gradient Ascent (SGA)

(1) Partitioning the continuous sensor-readings into discrete cells: The vector $(x_1, x_2, x_3, x_4, \cdots)$ shown in Figure 6 denotes a discrete observation X in Figure 3. The vectors are the unit basis vectors of length = (number of cells), that is, one component is 1, and the others are 0 with the one appearing at a different component for each observation, e.g., X_1 is $(x_1, x_2, x_3, x_4) = (1, 0, 0, 0)$.

(2) Continuous input case: Each joint's degree is normalized as $0 \leq \theta_i \leq 1$ (Figure 7).

Table 1: The action probability and the eligibility.

action a_j $= (y_1, y_2)$	action probability $\pi(a_j, W, X)$	eligibility $e(t) = \frac{\partial}{\partial w} \ln \pi(a_j, W, X)$
$a_1 = (0,0)$	$(1-f_1)(1-f_2)$	$\frac{-1}{1-f_1}\frac{\partial f_1}{\partial w} + \frac{-1}{1-f_2}\frac{\partial f_2}{\partial w}$
$a_2 = (0,1)$	$(1-f_1)\,f_2$	$\frac{-1}{1-f_1}\frac{\partial f_1}{\partial w} + \frac{1}{f_2}\frac{\partial f_2}{\partial w}$
$a_3 = (1,0)$	$f_1\,(1-f_2)$	$\frac{1}{f_1}\frac{\partial f_1}{\partial w} + \frac{-1}{1-f_2}\frac{\partial f_2}{\partial w}$
$a_4 = (1,1)$	$f_1\,f_2$	$\frac{1}{f_1}\frac{\partial f_1}{\partial w} + \frac{1}{f_2}\frac{\partial f_2}{\partial w}$

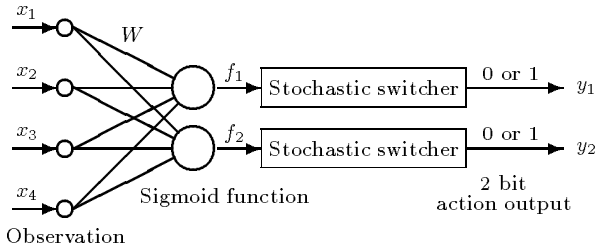

Figure 6: The discrete input implementation.

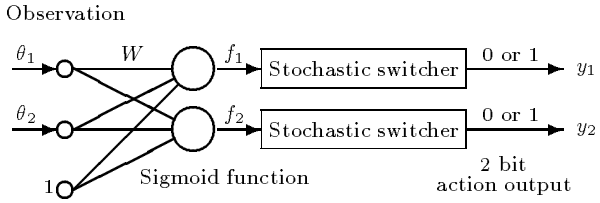

Figure 7: The continuous input implementation.

In this section, w_{ij} denotes a synaptic weight that connects i^{th} input unit with j^{th} sigmoid function. Similarly, the associated eligibility is $e_{ij}(t)$, and the eligibility trace is $D_{ij}(t)$. The input x_i and variables w_{ij} are transformed into probability f_i yielding

$$f_j = \frac{1}{1+\exp\left(-\sum_{i=1}^{4} x_i\, w_{ij}\right)} \ . \qquad (2)$$

The output y_i denotes a random variable that takes 0 or 1. f_i denotes the probability of $y_i = 1$. y_1 and y_2 correspond to the turning directions of the motors in the joint 1 and 2 respectively. Since the policy and eligibility are shown as Table 1, the agents shown in Figure 6 and 7 can calculate it by f_i and y_i as the following.

$$\begin{aligned} e_{ij}(t) &= \begin{cases} \dfrac{-1}{1-f_j}\dfrac{\partial f_j}{\partial w_{ij}} & \text{, where } y_j = 0, \\ \dfrac{1}{f_j}\dfrac{\partial f_j}{\partial w_{ij}} & \text{, where } y_j = 1. \end{cases} \\ &= \frac{y_j - f_j}{f_j(1-f_j)}\frac{\partial f_j}{\partial w_{ij}} \\ &= x_i(y_j - f_j)\ . \qquad (3) \end{aligned}$$

Figure 8 shows the concrete algorithm for Figure 6 and 7. It is obtained by using Equation 3 and the general form shown in Figure 8. The learning mechanism is very simple and very easy to implement on a parallel hardware because each sigmoid function can execute the calculation separately.

1. Observe X_t in the environment.
2. Execute action $a_t = (y_1, y_2)$ with probability $\pi(a_t, W, X_t)$.
3. Receive the immediate reward r_t.
4. Calculate $e_i(t)$ and $D_i(t)$ as
$$\begin{aligned} e_{ij}(t) &= x_i(y_j - f_j)\ , \\ D_{ij}(t) &= e_{ij}(t) + \gamma D_{ij}(t-1)\ , \end{aligned}$$
where γ $(0 \le \gamma < 1)$ denotes the discount factor, and w_i does the i^{th} component of W.
5. Calculate $\Delta w_i(t)$ as
$$\Delta w_{ij}(t) = (r_t - b)\, D_{ij}(t)\ ,$$
where b denotes the reinforcement baseline.
6. Policy Improvement: update W as
$$w_{ij} \leftarrow w_{ij} + \alpha(1-\gamma)\,\Delta w_{ij}(t)\ ,$$
where α is a nonnegative learning rate factor.
7. Move to the time step $t+1$, and go to step 1.

Figure 8: The implemented SGA algorithm both for discrete case and for continuous case. Note that it can be executed separately on each sigmoid function.

The learning rate is fixed to $\alpha = 0.4$, reinforcement baseline $b = 0.01$. All weight variables, w_{ij}, are initialized to random values between ± 0.05.

4.2.2 Q-learning

(1) Discrete input case: Q-learning (one-step Q-learning) (Watkins et al. 92) is a simple incremental algorithm based on dynamic programming. The Q-function is updated in the following way:

$$\begin{aligned} \Delta Q &= r_t + \gamma \max_{u \in A} Q(u, X_{t+1}) - Q(a_t, X_t), \\ Q(a_t, X_t) &\leftarrow Q(a_t, X_t) + \alpha\, \Delta Q. \end{aligned}$$

We use learning rate $\alpha = 0.4$ and discount factor $\gamma = 0.9$ because it may converge very slowly if the discount factor is very close to one. We adopt the action-selection method based on the Boltzmann distribution used in (Barto et al. 95). The probability of action is given in the ratio of $\exp(Q/temp)$, and the temperature scheduling is given by $temp = \frac{1000}{100+step}$.

(2) Continuous input case: Figure 9 shows the implementation for Q-learning, and the algorithm is given by

$$\begin{aligned} \Delta Q &= r_t + \gamma \max_{u \in A} Q(u, X_{t+1}) - Q(a_t, X_t) \ , \\ \Delta w_{ij} &= \Delta Q \cdot \frac{\partial Q(a_t, X_t)}{\partial w_{ij}} \ , \\ w_{ij} &\leftarrow w_{ij} + \alpha \, \Delta w_{ij} \ . \end{aligned}$$

In this experiment, $\partial Q(a_t, X_t)/\partial w_{ij} = x_i$. We used same parameters as the discrete input case, i.e., $\alpha = 0.4$, $\gamma = 0.9$ and $temp = \frac{1000}{100+step}$.

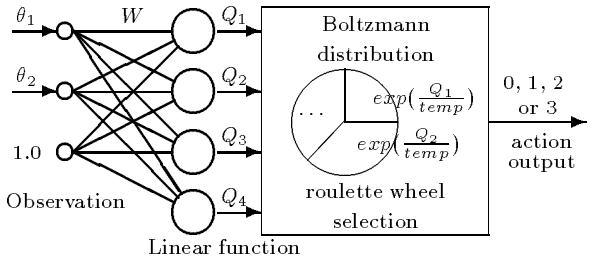

Figure 9: The continuous input implementation for Q-learning.

4.2.3 JSJ Method

Policy improvement based on a Monte-Carlo policy evaluation method (JSJ method) (Jaakkola et al. 94) is composed of two parts: Policy evaluation and policy improvement.

Policy evaluation: Calculate $Q(X,a)$ values that are defined by the average reward criteria. When the current taking action $a_t = a$, then

$$\begin{aligned} \beta_t(X,a) &= (1 - \frac{1}{k_t(X,a)}) \, \gamma_t \, \beta_{t-1}(X,a) + \frac{1}{k_t(X,a)} \\ Q_t(X,a) &= (1 - \frac{1}{k_t(X,a)}) \, Q_{t-1}(X,a) \\ &\quad + \beta_t(X,a) \ (r_t - \bar{r}) \\ \beta_t(X) &= (1 - \frac{1}{k_t(X)}) \, \gamma_t \, \beta_{t-1}(X) + \frac{1}{k_t(X)} \\ V_t(X) &= (1 - \frac{1}{k_t(X)}) \, V_{t-1}(X) + \beta_t(X) \ (r_t - \bar{r}) \end{aligned}$$

, else when $a_t \neq a$,

$$\begin{aligned} \beta_t(X,a) &= \gamma_t \, \beta_{t-1}(X,a) \\ Q_t(X,a) &= Q_{t-1}(X,a) + \beta_t(X,a) \ (r_t - \bar{r}) \\ \beta_t(X) &= \gamma_t \, \beta_{t-1}(X) \\ V_t(X) &= V_{t-1}(X) + \beta_t(X) \ (r_t - \bar{r}) \, , \end{aligned}$$

where $k_t(X,a)$ is the number of times (X,a) has occurred, $k_t(X)$ is the number of times X has occurred, $\bar{r}$ denotes the average of r_t, γ_t is a discount factor that is preferable to converging to one in the limit. In this experiment, we set $\gamma_t = 0.95$ fixed. A policy improvement step follows this calculation step.

Policy improvement: It is achieved by increasing the probability of taking the best action as defined by $Q(X,a)$. The new policy is guaranteed to yield a higher average reward as long as for some X, $\max_a [Q(X,a) - V(X)] > 0$.

In this experiment, the probability of taking action is directly proportional to the ratio of w_i, and the policy refinement is executed by adding 0.005 to the designated action's w_i and the weights are normalized thereafter. Initial weights are set to 0.25.

4.3 Results

(1) Discrete input case: Figure 10 shows the average performance of SGA in 4, 9, 16, 49 cells. The performance is not so good in rough partitioning condition because of the hidden state, however, the agent can move ahead by using a stochastic policy in the critical area. The cells are more increased, the performance grows better, because the higher resolution alleviates non-Markovian effects. But too many cells (49 cells) cause wasteful increasing of state variables. It has a bad influence on early performance.

In the case of discrete observation, SGA can compare with Q-learning (Watkins et al. 92) or JSJ method (Jaakkola et al. 94). Figure 12 to 15 present the results of the comparison. All algorithms use fixed parameters that are roughly best tuned on 16 cells.

The SGA method achieved good results in terms of both handling hidden state in the rough partitioning condition and insensitivity to increasing the number of state variables. In the rough partitioning condition (Figure 12), it is intractable for Q-learning, however, the SGA and the JSJ method converged to approximately the same positive expected reinforcement. Although non-Markovian effects remained strongly in 9 cells (Figure 13), Q-learning found an adequate deterministic policy. The performance of the Q-learning was relatively improved by increasing the cells. The Q-learning dominated others in Figure 15. The reason for this

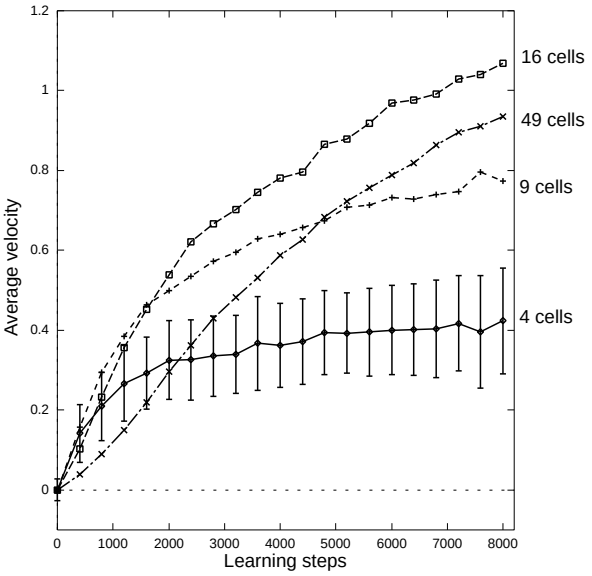

Figure 10: The average performance of 100 trials in the case of discrete input.

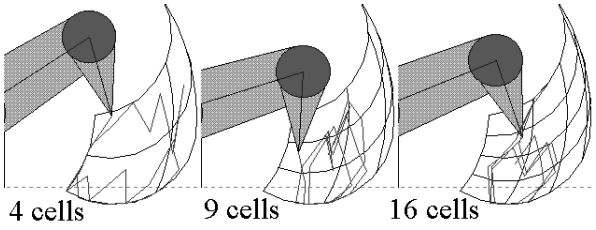

Figure 11: Sample trajectories of the policies that agents learned. (discrete input case)

is that partitioning the continuous observation into large number of cells makes the task a reasonably approximated MDP, and Q-learning takes advantage of Markovian property (because it is based on dynamic programming), but SGA and JSJ methods do not.

In contrast, the performance improvement of the JSJ method was more sensitive to increasing the number of cells than SGA. The reason for this may be a difference of the number of variables to form the policy, that is, the search space, between JSJ and SGA methods. In this experiment, the JSJ method needs more variables than the SGA although the agents have the same number of cells. But we should notice that reducing the policy parameters often results in defective function approximation that causes non-Markovian effects.

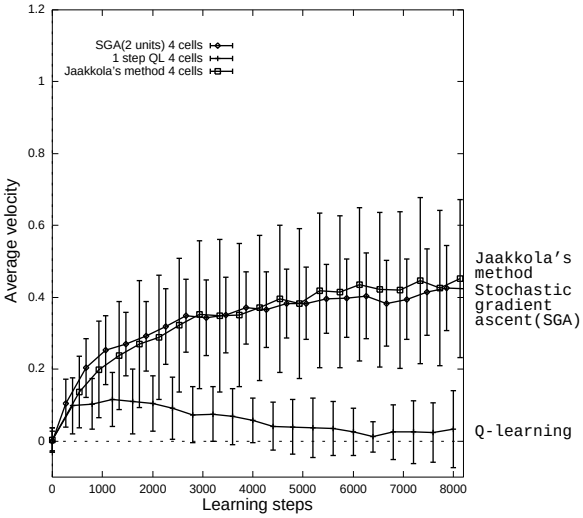

Figure 12: Comparison in the case of 4 cells.

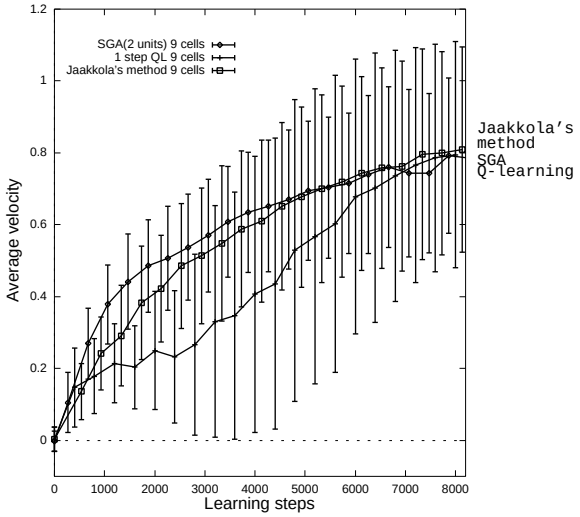

Figure 13: Comparison in the case of 9 cells.

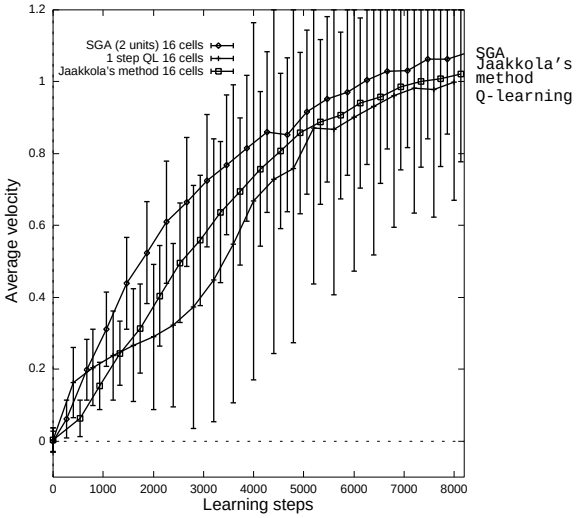

Figure 14: Comparison in the case of 16 cells.

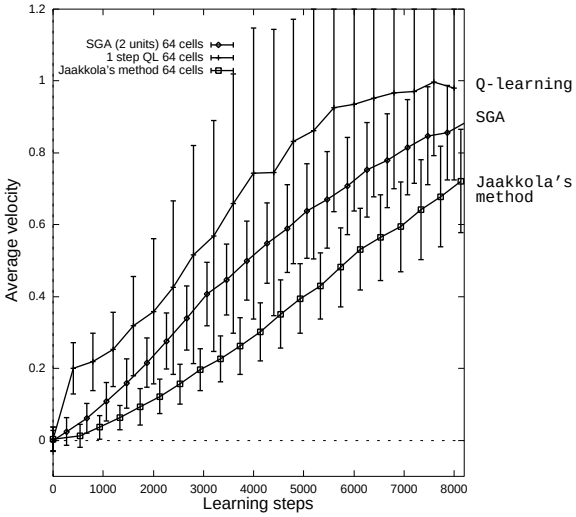

Figure 15: Comparison in the case of 64 cells.

(2) Continuous input case: In this experiment, we compared the gradient method with only Q-learning, because JSJ method cannot be applied with an arbitrary function approximator.

Figure 16 shows the average performance of 100 trials. Although the gradient method costs more 2 times of learning steps than the case that the continuous sensor-readings are partitioned into sufficient cells, it learns the policy to move to the front certainly. It is difficult to learn the policy for Q-learning. We tried Q-learning with another scheduling parameter $temp = \frac{10000}{1000+step}$, but the performance was not so different. The reason for this is that the function approximator shown in Figure 9 is insufficient to approximate the value function of this task. Q-learning needs richer resources for function approximation.

Figure 17 shows a trajectory of the gradient method after 30,000 steps. It was not converged to a fixed policy. Some stochastic action remained.

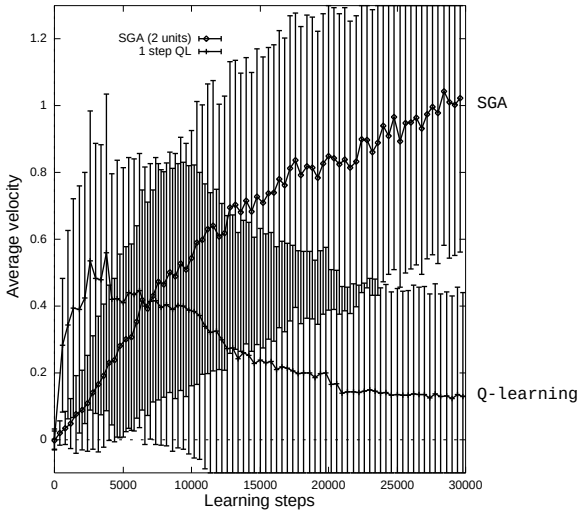

Figure 16: The average performance of 100 trials in the case of continuous input.

5 Discussion

Robustness: The Q-learning was more sensitive to hidden state, and the JSJ method was more sensitive to increasing the number of variables in the observation space than the SGA method. Also in the continuous input case, the SGA method obtained policies adequately with using the same learning parameters, whereas Q-learning resulted in failure. These results show that the stochastic

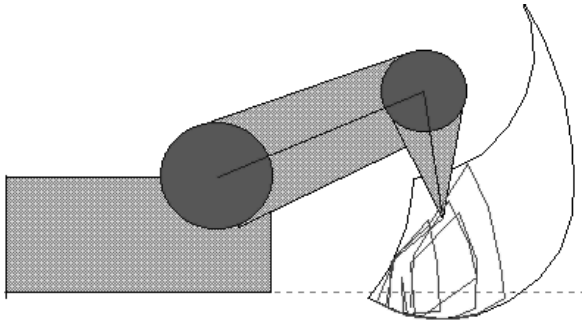

Figure 17: A trajectory after 30,000 steps in the case of continuous input.

gradient method is robustest in terms of handling hidden state, increasing the observation space, and handling function approximation.

Exploration strategy: The algorithm makes progress in the policy improvement proportionally to the state-visiting frequency. As a result, in a rarely visiting state, the agent tends to take random action. It can be considered to be a rational exploratory behavior to deal with the tradeoff of exploration and exploitation.

Advantages in function approximation: SGA method can be used together with many types of function approximator to the policy representation only if $\pi(a, W, X)$ is differential with W. Moreover, the SGA method can be executed in strongly resource-bounded conditions. In this experiment, the SGA agents learned policies with using only 2 sigmoid units, 6 weight variables and 6 variables for eligibility trace. If any non-Markovian effects are caused owing to function approximation, SGA method may learn better policies than random action at least.

SGA agents can also adopt fuzzy logic systems or neural networks as the function approximator, and SGA agents can handle continuous-valued action by providing $\pi(a, W, X)$ as a probability density function. We believe it is a promising approach for reinforcement learning to make use of expert's knowledge, because experts tend to indicate only input/output relations, not value function. When the input/output relations are given, we can easily modify initial weights to move an initial policy close to the expert's knowledge by using traditional supervised-learning techniques.

Stochastic policy: SGA method can maintain essential random action even if the others are converged in approximately deterministic actions. Such a behavior is difficult for Q-learning. Using stochastic policies is closely related to multi-player games. In Markov games (Littman 94), random actions are used to deal with the agent's uncertainty of the opponent's move.

Parallel processing: SGA agents can execute a localized computation in some cases. This feature is the same as REINFORCE algorithms (Williams 92), also related to probabilistic networks (Russel et al. 95) (called belief networks or Bayesian networks). It is interesting that the computational scheme of these methods are very similar, i.e., all these techniques make use of the gradient of the natural logarithm of some probability. We think the combination of the SGA method and probabilistic networks is fairly attractive.

6 Conclusion

This paper discussed that RL agents for practical tasks have to solve decision problems in POMDPs with function approximation, and resource-bounded agents need RL components that learn memory-less stochastic policies in POMDPs with using an arbitrary function approximator. We applied a stochastic gradient ascent algorithm (SGA method) to a robot control problem and showed its features by comparison with Q-learning and Jaakkola et al.'s algorithm (JSJ method). The task is small but significant because it can be seen as a RL problem in POMDPs with function approximation, also the computational resources are strongly restricted. The SGA method achieved good results in terms of handling hidden state, performance sensitivity to increasing the variables in the observation space, and possibility to use together with function approximation. The combination of SGA method and some state estimation (finite history windows, Bayesian networks, etc.) is a future work.

References

Barto, A. G., Bradtke, S. J. & Singh, S. P. (1995): Learning to act using real-time dynamic programming, *Artificial Intelligence 72*, pp. 81-138.

Chrisman, L. (1992): Reinforcement learning with perceptual aliasing: The Perceptual Distinctions Approach, *Proceedings of the 10th National Conference on Artificial Intelligence*, pp. 183-188.

Jaakkola, T., Singh, S. P., & Jordan, M. I.(1994): Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems, *Advances in Neural Information Processing Systems 7*, pp.345-352.

Kimura, H., Yamamura, M., & Kobayashi, S. (1995): Reinforcement Learning by Stochastic Hill Climbing on Discounted Reward, *Proceedings of the 12th International Conference on Machine Learning*, pp.295-303.

Kimura, H., Yamamura, M. & Kobayashi, S. (1996): Reinforcement Learning in Partially Observable Markov Decision Processes: A Stochastic Gradient Method, *Journal of Japanese Society for Artificial Intelligence*, Vol.11, No.5, pp.761-768 (in Japanese).

Kimura, H. (1997): Policy Improvement by Stochastic Gradient Ascent: A New Approach to Reinforcement Learning in POMDPs, *PhD thesis*, Tokyo Institute of Technology, Japan (in Japanese).

Kimura, H., Miyazaki, K., & Kobayashi, S. (1997): Reinforcement Learning in POMDPs with Function Approximation, *Proceedings of the 14th International Conference on Machine Learning*, to appear.

Lin, L. J. & Mitchell, T. M. (1992): Reinforcement Learning With Hidden States, *Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior (ICSAB)*, pp. 271-280.

Littman, M. L. (1994): Markov games as a framework for multi-agent reinforcement learning, *Proceedings of the 11th International Conference on Machine Learning*, pp. 157-163.

McCallum, R. A. (1993): Overcoming Incomplete Perception with Utile Distinction Memory, *Proceedings of the 10th International Conference on Machine Learning*, pp. 190-196.

McCallum, R. A. (1995): Instance-Based Utile Distinctions for Reinforcement Learning with Hidden State, *Proceedings of the 12th International Conference on Machine Learning*, pp. 387-395.

Moore A. W. & Atkeson, C. G. (1995): The Partigame Algorithm for Variable Resolution Reinforcement Learning in Multidimensional Statespaces, *Machine Learning 21*, pp. 199-233.

Russel, S., Binder, J. & Kanazawa, K. (1995): Local learning in probabilistic networks with hidden variables, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 1146-1152.

Singh, S. P., Jaakkola, T., & Jordan, M. I. (1994): Learning Without State-Estimation in Partially Observable Markovian Decision Processes, *Proceedings of the 11th International Conference on Machine Learning*, pp. 284-292.

Sutton, R. S. (1988): Learning to Predict by the Methods of Temporal Differences, *Machine Learning 3*, pp. 9-44.

Watkins, C. J. C. H., & Dayan, P. (1992): Technical Note: Q-Learning, *Machine Learning 8*, pp. 279-292.

Whitehead, S. D., & Lin, L. J. (1995): Reinforcement learning of non-Markov decision processes, *Artificial Intelligence 73*, pp.271-306.

Williams, R. J. (1992): Simple Statistical Gradient Following Algorithms for Connectionist Reinforcement Learning, *Machine Learning 8*, pp. 229-256.