Hierarchical Reinforcement Learning of Low-Dimensional Subgoals and High-Dimensional Trajectories

Jun Morimoto †and Kenji Doya††
Email: jun-m@is.aist-nara.ac.jp
†Graduate School of Information Science, Nara Inst. of Science and Technology
Takayama, Ikoma, Nara, 630-0101, Japan
†ATR Human Information Processing Research Labs.
††Kawato Dynamic Brain Project, ERATO, JST
Seika, Soraku, Kyoto, 619-0288, Japan

ABSTRACT

In this paper, we propose a hierarchical reinforcement learning method which enables a learner to learn tasks in a high-dimensional state space. In the upper level, the learner coarsely explores the low-dimensional state space. In the lower level, the learner finely explores the high-dimensional state space. Specifically, the learner learns to set up appropriate subgoals for the task in the upper level, and learns to achieve the subgoals in the lower level. As an example task, we choose a stand-up task involving a two-joint three-link robot. This robot has a ten-dimensional state space. The robot learns to find subgoal postures in the upper level, and to achieve these subgoal postures in the lower level. Simulation results show that the hierarchical architecture acceralates the learning of the robot to stand up.

KEYWORDS: Reinforcement learning, Hierarchical, Stand up, Robotics

1. Introduction

Reinforcement learning does not require explicit knowledge on the desired trajectories, only a signal that evaluates if the action taken by the learner is "good" or "bad". One of the most difficult problems in applying reinforcement learning to real-world control tasks is how to deal with a high-dimensional state space. Most previous studies have applied reinforcement learning to tasks in a discrete low-dimensional state space, such as maze exploration [1] or linear pole balancing in a discretized state space [2]. Some studies have applied reinforcement learning to tasks in a continuous low-dimensional state space, such as pendulum swing-up task [3, 4, 5].

However, many real world problems involve a continuous high-dimensional state space. For example, humans learn to control their own body which includes tens or hundreds of degrees of freedom. As an example for studying methods for applying reinforcement learning to real world problems, we choose a stand-up task using a two-joint three-link robot, as shown in Figure 1. The robot is not fixed to the ground and has a ten-dimensional state space. The goal of the task is to find a path that links a lying state to an upright state under the constraint given by the robot dynamics.

How can we apply reinforcement learning to such a high-dimensional state space? One answer is to employ a divide and conquer strategy using a hierarchical architecture. In this paper, we introduce a two-level learning architecture, with one upper layer and one lower layer. The upper learning modules explore a large area using a representation of the low-dimensional state space. In contrast, the lower learning modules explore small areas in the original high-dimensional state space.

Hierarchical reinforcement learning using subgoals has been applied to discrete maze search tasks [6, 7] and to the sequential control of a two-link robot arm [8]. In [7], the subgoals were movement directions in a coarse-grain state space. In [6] and [8], the subgoals were specific points in a lower-level state space. One problem in applying a hierarchical method to high-dimensional problems is that even a very coarse division of the state space, e.g., two in each variable, can result in a very large number of states in the upper level. Therefore, it is essential to consider only a low-dimensional subspace in the upper level by using an appropriate projection from the original high-dimensional state space.

In our stand-up task, we take only the joint and pitch angles of intermediate postures as the subgoals in the upper level, while other variables, such as angular velocities, are left unspecified. This makes possible the learning of subgoals in a low-dimensional state space. The exact values of the remaining variables are determined by a lower-level optimization process. The task for the upper-level process is to find an appropriate sequence of subgoal postures attainable by the lower-level processes.

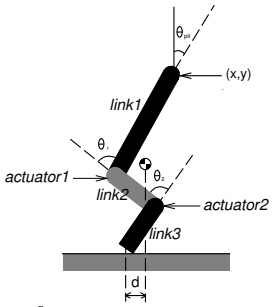


Figure 1: Robot configuration.

The robot has a ten-dimensional state space:
$\mathbf{x} = (\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2, \theta_{pit}, \dot{\theta}_{pit}, x, \dot{x}, y, \dot{y})$, where θ_1 denotes the hip angle, θ_2 denotes the knee angle, and d gives the horizontal displacement of the center of mass from the center of the foot.

Table 1: Physical parameters of the robot

		link1	link2	link3
length	experiment 1	1.25 m	0.5 m	0.5 m
	experiment 2	1.0 m	0.5 m	0.5 m
width		0.2 m	0.2 m	0.2 m
weight		20 kg	20 kg	20 kg

Table 2: Parameters of servo controllers

	actuator1	actuator2
max torque	260 $N \cdot m$	700 $N \cdot m$
position gain k_τ	100 $\frac{N \cdot m}{deg}$	300 $\frac{N \cdot m}{deg}$
velocity gain b_τ	10 $\frac{N \cdot m}{deg/s}$	30 $\frac{N \cdot m}{deg/s}$

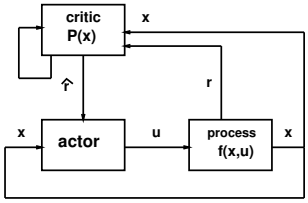


Figure 2: Actor-critic architecture

2. Stand-up task and learning method

The goal of the task is to find a path that links a lying state to an upright state under the constraint given by the robot dynamics. We apply reinforcement learning, continuous temporal difference (TD) learning [3] in particular, to this task. We use the actor-critic method [2] to implement the continuous TD learning (see Fig. 2). The robot configuration and state variables are shown in Fig. 1.

The torque τ_i of the actuator ($i = 1, 2$) is given by a feedback controller

$$\tau_i = k_{\tau_i}(\theta_{di} - \theta_i) + b_{\tau_i}(\dot{\theta}_{di} - \dot{\theta}_i) \quad (1)$$

where k_τ is the position gain , and b_τ is the velocity gain (see Table 2).

2.1. Plain Reinforcement Learning

In a previous study [9], we accomplished a stand-up task with a plain, non-hierarchical architecture. Namely, we select the most essential variables as the input to the actor and critic networks. The variables are $\mathbf{x} = (\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2, d, \dot{d})$. The actor works as a control function that takes the six state variables as the input, and provides the desired joint angles $\mathbf{u}(\mathbf{x}) = (\theta_{d1}, \theta_{d2})$ as the output. The critic works as a predictor of the value function $P(\mathbf{x})$ which also takes the six state variables and makes a prediction of the value of the state [3]. The function approximation method and the representation of the state space for implementing the actor $\mathbf{u}(\mathbf{x})$ and the critic $P(\mathbf{x})$ are critical in applying the actor-critic method to high-dimensional tasks.

We have been successful in the task by using an efficient function approximation method, i.e., Adaptive Gaussian Softmax Basis Function Network (Adaptive GSBF Network). Unfortunately, our previous experiments have shown that the Adaptive GSBF network works sufficiently only up to about a six-dimensional state space. Therefore, an alternative solution is necessary for higher dimensional problems.

3. Hierarchical Reinforcement Learning

We consider a divide and conquer strategy. The basic idea is to divide a highly non-linear problem in a high-dimensional space into two levels: a nonlinear problem in a lower-dimensional space and nearly linear problems in the high-dimensional space.

3.1. Upper level learning

In the upper level, the learner explores the whole relevant area of the low-dimensional state space. In order to facilitate the global search, the state and action spaces are coarsely discretized. We then use the $Q(\lambda)$-learning method [10] to learn the subgoal sequence.

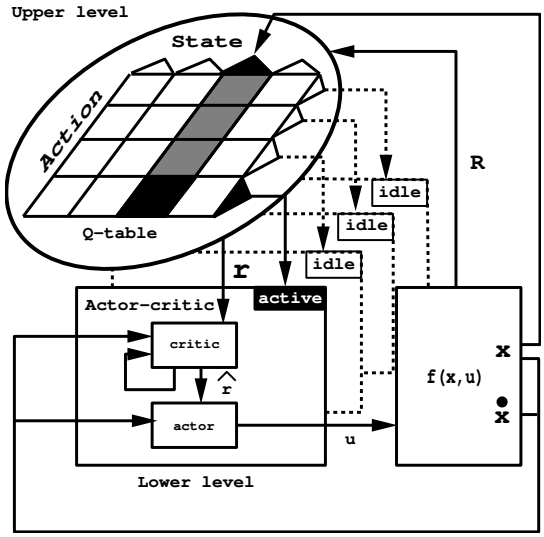


Figure 3: Hierarchical architecture: The gray part shows current the state, the black part shows the action chosen by the upper level, and r_{up} denotes the reward from the upper level to the lower level.

In a stand-up task, we choose the posture of the robot $\mathbf{X}$=$(\theta_1, \theta_2, \theta_{pit})$ as input variables, and choose the desired posture of the robot $\mathbf{U}$=$(\theta_{s1}, \theta_{s2}, \theta_{spit})$ as an action command sent to the lower level.

3.2. Lower level learning

In the lower level, the learner explores limited areas of the high-dimensional state space without discretization. Note that a lower level learner will learn to achieve the subgoal specified as the upper level action. We use the continuous $TD(\lambda)$-learning method [3] to learn the control command sequence.

In a stand-up task, we choose $\mathbf{x}$= $(\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2, \theta_{pit}, \dot{\theta}_{pit})$ as input variables, and choose the desired joint angle $\mathbf{u} = (\theta_{d1}, \theta_{d2})$ as output variables.

3.3. Combining the upper and lower levels

The lower-level modules learn to achieve a subgoal from a given initial state. Because each subgoal is defined in the low-dimensional state space in the upper level, the subgoal is not a point but a hyper-plane in high-dimensional state space in the lower level. A reward is given to the upper level by the success of the entire task. A reward is given to the lower level by the achievement of the subgoal specified by the upper level (See Fig 3) [7].

In a stand-up task, we use the height of the head as a reward for the upper level, and the distance from the current posture to the desired posture given by the upper level as a reward for the lower level.

3.4. Switching Subgoals

We recognize that a subgoal is achieved when the system state reaches within a certain neighborhood of the subgoal. After the completion of one subgoal, the state of the upper level is updated and the next action is taken, that is, the next subgoal is given to the lower level.

Table 3: Permissible states and actions in the upper level

	State	Action
Stage 1	$\{S_0\}$	$\{S_1, S_2, S_3\}$
Stage 2	$\{S_1, S_2, S_3\}$	$\{S_5 \cdots S_{12}\}$
Stage 3	$\{S_5 \cdots S_{12}\}$	$\{S_{13}\}$
Success	$\{S_{13}\}$	
Failure	$\{S_{-1}\}$	

Summary of the hierarchical architecture

Upper level
Input: $\mathbf{X} = (\theta_1, \theta_2, \theta_{pit})$
Output: Subgoal posture $\mathbf{U} = (\theta_{s1}, \theta_{s2}, \theta_{spit})$
Reward: Head height $R = F(y)$

Lower level
Input: Joint and pitch angles, and their derivatives
$\mathbf{x}$= $(\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2, \theta_{pit}, \dot{\theta}_{pit})$
Output: Desired joint angle $\mathbf{u} = (\theta_{d1}, \theta_{d2})$
Reward: Distance to the subgoal posture
$r(t) = F(|\theta_1 - \theta_{s1}|, |\theta_2 - \theta_{s2}|, |\theta_{pit} - \theta_{spit}|)$

4. Experiments

Each trial is started with the robot lying on the ground, and is continued for T=10 seconds in simulated time. When the robot falls down and hits its hip or head on the ground (**Failure**), the trial is terminated and the next trial is started.

4.1. Experiment 1: Plain architecture

We use an adaptive GSBF network to approximate the control functions (actor) and the value function (critic).
The reward is given by the head height y:

$$r(y) = \begin{cases} (\frac{y}{l})^2 - 1 & (during\ trial) \\ -1 & (on\ failure) \end{cases}$$

where l is the total length of the robot. Each simulation continues up to 2,000 trials.

4.2. Experiment 2: Hierarchical architecture

In this experiment, we utilize our prior knowledge about stand-up movements in the design of the hierarchical architecture. Stand-up movements include three sequential component movements.

Stage: 1 Pull up the upper body.
Stage: 2 Bend the knees to get on one's feet.
Stage: 3 Straighten the hip and knees to stand up.

Using the above component movements, we select a set of permissible states and actions in the upper level. Each upper level action is executed by a separate actor-critic module in the lower level as shown in Figure 3.
We define the upper level states and actions using a finite set of postures S_i $(i = -1 \cdots 13)$ (See Fig. 4). On each stage, one of the permissible actions is taken, as shown in Table 3.

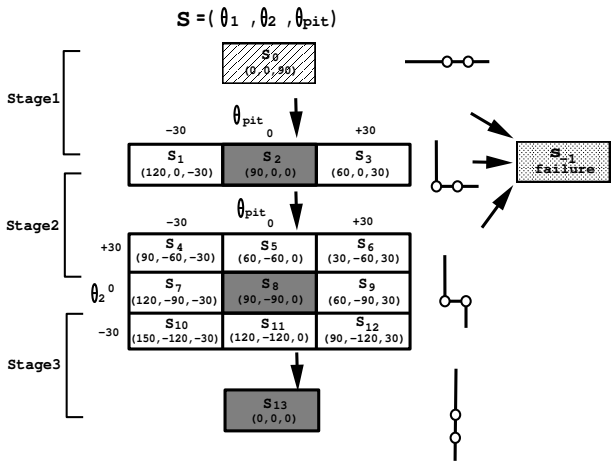


Figure 4: Upper level states and actions: The hatched part shows the initial state, the gray parts and robot figure on right side show default actions in each state, and the dotted part shows failure state.

The reward function for the upper level is given by

$$R(r_h) = R_s + \frac{\int_0^t r_h(t)dt}{T_s}$$

$$R_s = \begin{cases} 1 & (on\ \ success) \\ 0 & (on\ \ failure) \end{cases}$$

$$r_h(y) = \begin{cases} (\frac{y}{l})^2 & (in\ \ stage3) \\ 0 & (in\ \ stage1, 2) \end{cases}$$

where $T_s = 1[sec]$ is a scaling constant for the reward. y is the head height of the robot, and l is the total length of the robot.

The reward function for the lower level critic is given by

$$r(\theta_1, \theta_2, \theta_{pit}) = e^{-\left(\frac{|\theta_1 - \theta_{s1}|^2}{\sigma_{r1}^2} + \frac{|\theta_2 - \theta_{s2}|^2}{\sigma_{r2}^2} + \frac{|\theta_{pit} - \theta_{spit}|^2}{\sigma_{rpit}^2}\right)} - 1$$

where σ_r is the width of the reward function. A penalty of $r = -1$ is given when the robot falls to the floor.

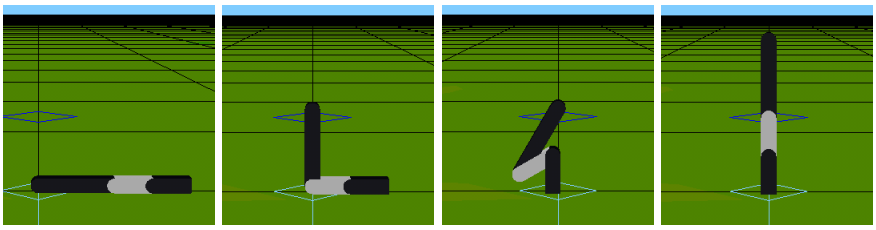
Stages 1 and 2 are completed when the reward is $r > -0.1$, that is, when the robot reaches within a certain neighborhood around the subgoal posture. The next stage is started from that point with the next subgoal. Each simulation run continues up to 500 trials.

5. Results

We tested above methods with the following two cases.

Experiment 1: First, we tested the plain architecture with the adaptive GSBF Network. The robot successfully learned to stand up within 2,000 trials in five out of 15 simulation runs. Table 4 summarizes the results of the five successful runs.

Experiment 2: Next, we tested the hierarchical architecture with the adaptive GSBF Network. The robot successfully learned to stand up within 500 trials in five out of 24 simulation runs; the hierarchical architecture was about two times faster than with the plain architecture in the average number of trials, as shown in Table 4.



(a) Initial state (b) Subgoal 1 (c) Subgoal 2 (d) Subgoal 3

Figure 5: Upper level state sequence

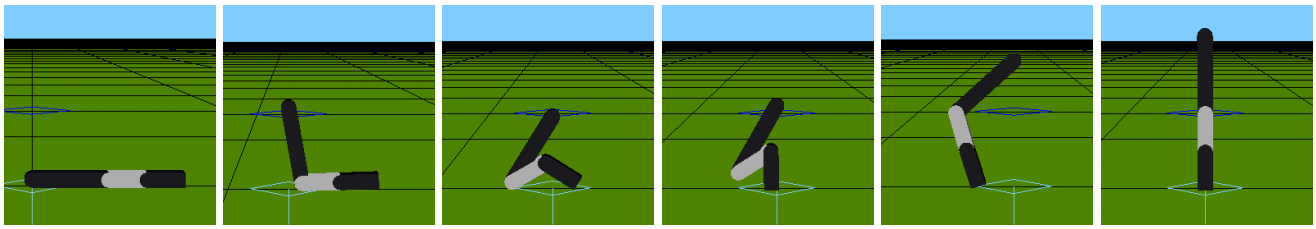


Figure 6: Lower lever movement sequence (learned stand-up trajectory)

Table 4: The number of trials and the CPU time needed for learning to stand-up (CPU: SGI Indigo 2). The table gives the average over five successful runs.

Method	trials	CPU time
Plain	734	39min.
Hierarchical	331	15min.

Figures 5 and 6 show examples of learned the upper level state sequence, which is subgoal sequence, and learned the lower level state sequence, which is the stand-up trajectory, respectively.

6. Conclusion

In this paper, we proposed a hierarchical reinforcement learning method that enables a learner to learn a task in a high-dimensional state space. Successful results were obtained for a two-joint three-link robot stand-up task. These results shows that the proposed hierarchical architecture can successfully learning the stand up task in fewer trials. In this hierarchical learning method, we can introduce prior knowledge naturally into the upper level learning as a set of subgoals. Upper level learning with little or no prior knowledge will be a subject in future work.

References

[1] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.

[2] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:834–846, 1983.

[3] K. Doya. Temporal difference learning in continuous time and space. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 1073–1079. MIT Press, Cambridge, MA, 1996.

[4] K. Doya. Efficient nonlinear control with actor-tutor architecture. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 1012–1018. MIT Press, Cambridge, MA, 1997.

[5] S. Schaal. Learning From Demonstration. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 1040–1046. MIT Press, Cambridge, MA, 1997.

[6] M. Wiering and J. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6(2), 1997.

[7] P. Dayan and G. E. Hinton. Feudal reinforcement learning. *NIPS 5*, pages 271–278, 1993.

[8] C. K. Tham. Reinforcement learning of multiple tasks using a hierarchical CMAC architecture. *Robotics and Autonomous Systems*, 15:247–274, 1995.

[9] J. Morimoto and K. Doya. Learning "stand-up" trajectories using reinforcement learning. In *Technical Report of IECE*, volume NC97-28(1997-07), pages 25–32. The Institute of Electronics, Information and Communication Engineers, 1997.

[10] J. Peng and R. Williams. Incremental multi-step Q-learning. *Machine Learning*, 22:283–290, 1996.