Reinforcement learning of dynamic motor sequence: Learning to stand up

Jun Morimoto

jun-m@is.aist-nara.ac.jp
Graduate School of Information Science
Nara Inst. of Science and Technology
ATR Human Infomation Processing Lab.
Takayama, Ikoma, Nara, 630-0101, Japan

Kenji Doya

doya@erato.atr.co.jp
Computational Neurobiology Group
Kawato Dynamic Brain Project, JST
Nara Inst. of Science and Technology
Seika, Soraku, Kyoto, 619-0288, Japan

Abstract

In this paper, we propose a learning method for implementing human-like sequential movements in robots. As an example of dynamic sequential movement, we consider the "stand-up" task for a two-joint, three-link robot. In contrast to the case of steady walking or standing, the desired trajectory for such a transient behavior is very difficult to derive. The goal of the task is to find a path that links a lying state to an upright state under the constraints of the system dynamics. The geometry of the robot is such that there is no static solution; the robot has to stand up dynamically utilizing the momentum of its body. We use reinforcement learning, in particular, a continuous time and state temporal difference (TD) learning method. For successful results, we use 1) an efficient method of value function approximation in a high-dimensional state space, and 2) a hierarchical architecture which divides a large state space into a few smaller pieces.

1 Introduction

Recently, there have been many attempts at controlling humanoid robots in a human-like way [6]. In the case of upper-body movements and steady walking, for example, the desired trajectories and the control laws to implement them have been well analyzed theoretically [5, 8]. On the other hand, trajectories for transient sequential movement such as a dynamic stand-up are difficult to derive theoretically, intuitively, or empirically. In this paper, we investigate methods for a robot to achieve a dynamic stand-up by itself through learning. We take the reinforcement learning approach which does not require explicit knowledge of the desired trajectories, but only requires a signal that evaluates if the action taken by the robot is "good" or "bad". The robot learns stand-up trajectories through trial and error, in the same way as humans learn motion patterns. We use a two-joint, three-link robot, as shown in Figure 1, which is not fixed to the ground and has a ten-dimensional state space.

The application of reinforcement learning to such a high-dimensional system is difficult, especially if a grid-like representation of the state is used. Therefore, we explore the use of 1) an efficient method of value function approximation in a high-dimensional state space, and 2) a hierarchical architecture which divides a large state space into smaller pieces using subgoals in a lower-dimensional space. Using these methods, we have obtained successful results where the two-joint, three-link robot learned the the dynamical stand-up motion by computer simulation.

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

	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 $\frac{N \cdot m}{deg/s}$

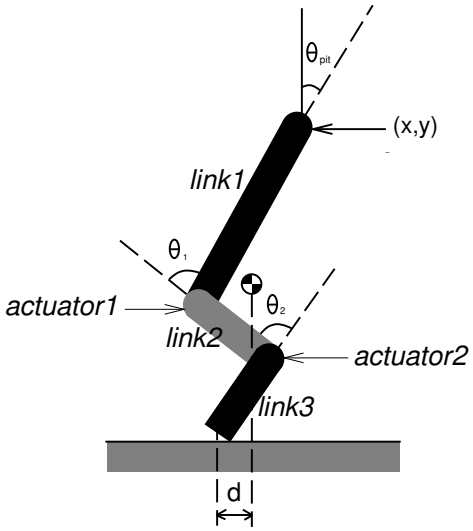


Figure 1: Robot configuration.
The robot has ten dimensional state space :
$\mathbf{x} = (\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2, \theta_{pit}, \dot{\theta}_{pit}, x, \dot{x}, y, \dot{y})$.

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, especially continuous temporal difference (TD) learning [4] to this task. We use the actor-critic method [1] to implement the continuous TD learning (see Fig. 2). The robot configuration and state variables are shown in Fig. 1. We select the most essential variables as the input to the actor and critic networks. The variables are $\mathbf{x} = (\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2, d, \dot{d})$, 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 (see Fig. 1).

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 torque τ_i of the actuator $(i = 1, 2)$ is calculated from control law

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

where k_τ is the position gain , and b_τ is the velocity gain (see Table 1).

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 [4].

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 consider these points in the next section.

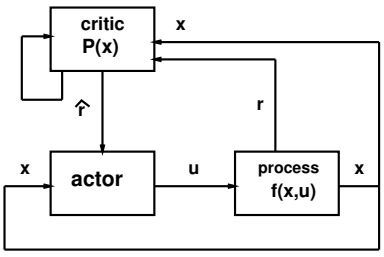


Figure 2: Actor-critic architecture

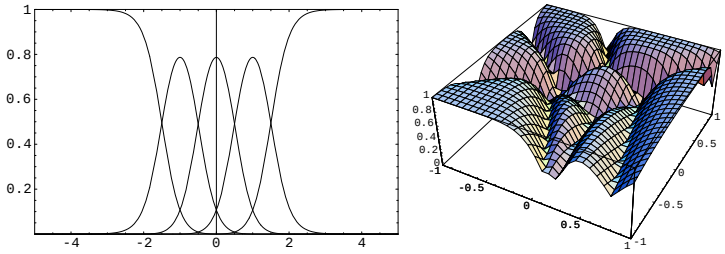
3 Value function approximation

A common approach in reinforcement learning of control tasks is to discretize the state space using "boxes" [1] or CMAC [9] to approximate the value function. Problems with such an approach are the lack of generalization and perceptual aliasing. Furthermore, when we have to deal with a high-dimensional state space, simple direct product of each discretized state variable will produce a huge number of states. Therefore, we consider a continuous value function approximation method that is capable of dealing with high-dimensional input.

In many control tasks, the part of the state space that is relevant for the task is very limited. In such a case, we should use precise interpolation in that part of the state space that is important for the task while using coarse extrapolation in the rest of the state space. Now, we consider how to implement that idea below.

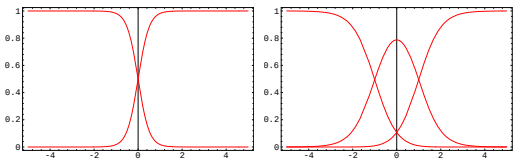
A network with sigmoid basis functions is widely used for non-linear function approximation. Global function approximation using sigmoid basis functions provides a good nature of generalization and works well for off-line learning. However, sigmoid basis function approximation has the problem of catastrophic interference when used for on-line learning [7]. When applied to temporal difference (TD) learning, the parameters of sigmoid networks can, in some cases, diverge [2].

Local function approximation methods like radial basis function (RBF) network are suitable for on-line learning. However, RBF has the drawback of limited generalization and problems caused by the curse of dimensionality. In this paper, we investigate a function



a b

Figure 3: Extension of Gaussian softmax basis functions. (a) One dimensional example. (b) Two dimensional example.



a b

Figure 4: Adaptation of the shape of basis functions. (a) If there is no neighboring basis function in the middle, the shapes of all basis functions extended in a sigmoidal fashion. (b) When a new basis function is allocated, the shapes of neighboring basis functions change adaptively.

approximation method using Gaussian softmax basis functions (GSBF) which can implement both local and global basis functions in a natural way.

3.1 Gaussian Softmax Basis Function Network (GSBFN)

Figure 3 (a) shows examples of scalar Gaussian softmax basis functions given by

$$b_k(x) = \frac{e^{-\frac{1}{2}|\frac{(x-c_k)}{s_k}|^2}}{\sum_l e^{-\frac{1}{2}|\frac{(x-c_l)}{s_l}|^2}}$$

where c_k and s_k are the center and the size of the k-th basis function. The basis functions in the middle are local but those at the ends are extended in a sigmoidal fashion by the effect of normalization. The same characteristics can be seen in a higher-dimensional space, as illustrated in Figure 3 (b) for the two-dimensional case.

For a given n dimensional input vector $\mathbf{x} = (x_1, ..., x_n)^T$, the *activation function* of the k-th unit is calculated by

$$a_k(\mathbf{x}) = e^{-\frac{1}{2}||M_k(\mathbf{x}-\mathbf{c}_k)||^2},$$

where $\mathbf{c}_k$ is the center of activation and M_k is a matrix that determines the shape of the activation function. The softmax *basis function* is then given by

$$b_k(\mathbf{x}) = \frac{a_k(\mathbf{x})}{\sum_{l=1}^{K} a_l(\mathbf{x})},$$

where K is the number of basis functions. The output is given by the inner product of the basis functions and the weights w_k

$$y(\mathbf{x}) = \sum_{k=1}^{K} w_k b_k(\mathbf{x}).$$

For a given target output $\hat{y}(\mathbf{x})$, the weights are updated by the LMS rule

$$\Delta w_k = -\eta_w (y(\mathbf{x}) - \hat{y}(\mathbf{x})) b_k(\mathbf{x}) \qquad (k = 1, ..., K).$$

where η_w is the learning rate.

3.2 Adaptive GSBFN

In an adaptive GSBF, a new unit is allocated if the error is larger than a criterion $e_{\max}$ and the activation of all existing units is smaller than a threshold $a_{\min}$, that is,

$$|y(\mathbf{x}) - \hat{y}(\mathbf{x})| > e_{\max} \quad \text{and} \quad \max_k a_k(\mathbf{x}) < a_{\min}.$$

The new unit is initialized with $w_k = \hat{y}(\mathbf{x})$, $\mathbf{c}_k = \mathbf{x}$, and $M_k = \mathrm{diag}(\mu_i)$, where μ_i is the inverse of the radius of basis function.

Note that when a new basis function is allocated, the shapes of neighboring basis functions also change because of the nature of GSBF (see Fig. 4).

The shape and the center of the activation can be further optimized by the following learning rules.

$$\begin{aligned}
\Delta M_k &= -\eta_M (y(\mathbf{x}) - \hat{y}(\mathbf{x})) w_k \frac{\partial b_k}{\partial a_k} \frac{\partial a_k}{\partial M_k} \\
&= -\eta_M (y(\mathbf{x}) - \hat{y}(\mathbf{x})) w_k (b_k(\mathbf{x}) - 1) b_k(\mathbf{x}) M_k (\mathbf{x} - \mathbf{c}_k)(\mathbf{x} - \mathbf{c}_k)^T, \\
\Delta \mathbf{c}_k &= -\eta_c (y(\mathbf{x}) - \hat{y}(\mathbf{x})) w_k \frac{\partial b_k}{\partial a_k} \frac{\partial a_k}{\partial \mathbf{c}_k} \\
&= \eta_c (y(\mathbf{x}) - \hat{y}(\mathbf{x})) w_k (b_k(\mathbf{x}) - 1) b_k(\mathbf{x}) M_k^T M_k (\mathbf{x} - \mathbf{c}_k),
\end{aligned}$$

where η_M and η_c are the learning rates.

4 Dividing a task using a hierarchical architecture

Although we have achieved modest success in the stand-up task using the adaptive GSBFN (experiment 1, see Section 5.1), we consider a *divide and conquer* strategy to further accelerate learning (experiment 2, see Section 5.2). The basic idea is to divide a highly non-linear task in a high-dimensional space into two levels: local optimization in the high-dimensional space and global exploration in a lower-dimensional space.

Hierarchical reinforcement learning using subgoals has been applied to discrete maze search tasks [10, 3] and to sequential control of a two-link robot arm [9]. In [3], the subgoals were movement direction in a coarse-grain state space. In [10] and [9], the subgoals were specific points in the lower-level state space.

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

4.1 Stand-up task using a hierarchical architecture

In the hierarchical architecture, the upper level process works as a discrete reinforcement learning system whose actions are discrete sequence of subgoal postures. The lower level process takes the subgoal posture as the peak of the reward function and learns to maximize it, that is, to find a trajectory to reach the subgoal.

Summary of the hierarchical architecture

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

Lower level
Input: Joint angle, joint angular velocity, pitch angle, pitch angular velocity
$\mathbf{x} = (\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2, \theta_{pit}, \dot{\theta}_{pit})$
Output: Desired joint angle $\mathbf{u} = (\theta_{d1}, \theta_{d2})$
Reward: Function of subgoal posture
$r(t) = F(\theta_1, \theta_2, \theta_3; \theta_{s1}, \theta_{s2}, \theta_{spit})$

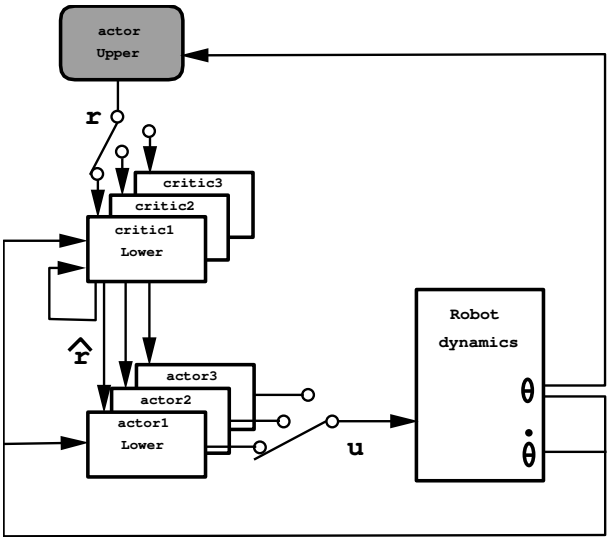


Figure 5: The hierarchical architecture used for the stand-up task.

5 Experiments

Each trial was started with the robot lying on the ground, and was continued for ten seconds in simulation time. When the robot fell down and hit its hip or head on the ground, the trial was terminated and was restarted again.

5.1 Experiment 1: Adaptive GSBF

We used an adaptive GSBF network to approximate the control functions (actor) and the value function (critic). We used $e^2_{max} = 0.001$ and $a_{min} = e^{-1.0}$ when a new unit was allocated. We used $\mu_i = (2.3, 0.18, 2.3, 0.18, 8.0, 0.5)$ for initialization of basis functions. In the current experiment, the shape and the center of the activation functions were not updated for simplicity.

The reward was 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 was continued up to 2,000 trials.

5.2 Experiment 2: The hierarchical architecture

In the hierarchical architecture, we divided the task into three sub-tasks using two intermediate postures

Table 2: Sequence of subgoal postures.

	$(\theta_{s1}, \theta_{s2}, \theta_{spit})[deg]$
Initial state	$(0, 0, 90)$
Subgoal 1	$(90, 0, 0)$
2	$(150, -120, -30)$
3	$(0, 0, 0)$

as the subgoals, as shown in Table 2 and Figure 6. Each subtask uses a separate actor-critic pair at the lower level as shown in Figure 5.

The reward function for the lower level critic was

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

where σ_r was the width of the reward function.

A subtask except for the last one was completed when the reward was $r > -0.1$, that is, when the robot reached to a tube in the state space around the subgoal posture. A new subtask was started from that point with the next subgoal. The entire task was terminated after ten seconds from the start or when the robot fell down to the floor, in which case a penalty of $r = -1$ was given for the subtask. Each simulation run was continued up to 300 trials.

5.3 Results

In a preliminary experiment, we tested the performance of GSBFN with fixed basis functions allocated in a grid shape. Each of the six input variables $(\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2, d, \dot{d})$ was represented by (7, 3, 7, 3, 10, 3) basis functions, respectively, the product of which resulted in the total of 13,230 basis functions. In this case, it took about 8,000 trials and 9 hours to learn to stand-up.

We tested above methods with the following two cases.

- **Experiment 1**
  Plain: A plain architecture without subgoals using adaptive GSBFN.

- **Experiment 2**
  Hierarchical: A hierarchical architecture using adaptive GSBFN.

First we tested the plain architecture with the adaptive GSBFN. The robot successfully learned to stand up within 2,000 trials in five out of 15 simulation runs. Table 3 summarizes the results of the five successful runs. Learning was about ten times faster than with fixed GSBF in the preliminary experiment. This result shows an efficient function approximation ability of adaptive GSBF.

Table 3: The number of bases, trials, and CPU time needed for learning to stand-up (CPU: SGI Indigo 2). Average over 5 successful runs.

Method	bases	trials	CPU time
Plain	actor: 246 critic: 261	734	39min.
Hierarchical	actor1: 25 critic1: 46	106	7min.
	actor2: 26 critic2: 46		
	actor3: 84 critic3: 150		

Next, we tested the hierarchical architecture with adaptive GSBFN. The robot successfully learned to stand up within 300 trials in five out of 13 simulation runs, and about seven times faster than with the plain architecture in number of trials, as shown in Table 3.

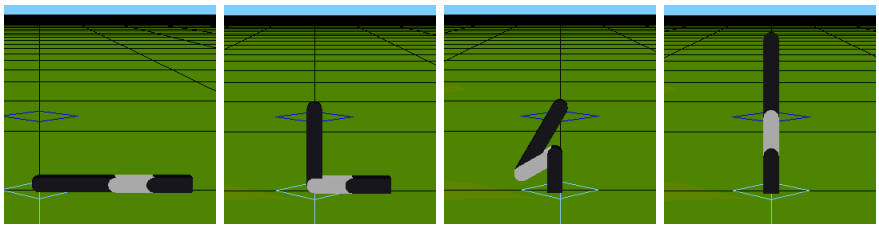
Figures 6 and 7 show the subgoal sequence and the stand-up trajectory learned by hierarchical architecture.

6 Conclusion

In this paper, we proposed a learning method that enables a two-joint, three-link robot to stand up in a high-dimensional state space. Continuous temporal difference (TD) learning with adaptive GSBF and a hierarchical architecture enables the robot to learn to stand up in a relatively small number of trials.

The results show that adaptive GSBF is suitable for approximating functions in high-dimensional state space, and that a hierarchical architecture enables the robot to explore efficiently in high-dimensional state space. Furthermore, the hierarchical architecture enables us to easily incorporate our physical knowledge or the data from demonstration into the learning process.

In this paper, we fixed the action sequence of the upper level. Simultaneous learning of both upper and lower level actions is the subject of future work.



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

Figure 6: Upper level action sequence

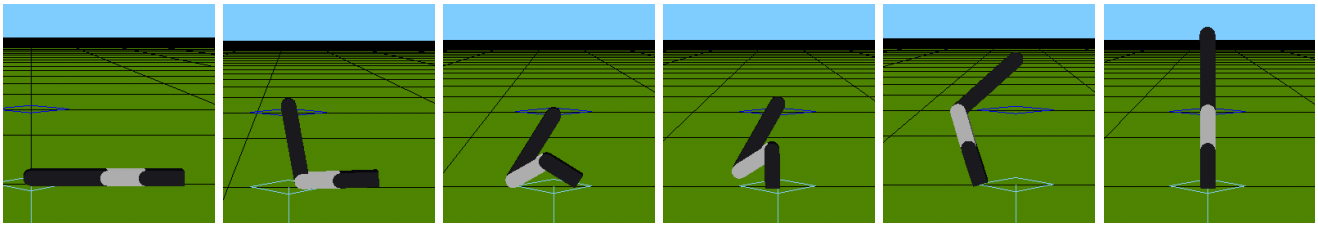


Figure 7: Lower lever action sequence (learned stand-up trajectory)

References

[1] 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.

[2] J. A. Boyan and A. W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 369–376. MIT Press, Cambridge, MA, USA, 1995.

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

[4] 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.

[5] J. Furusho and A. Sano. Sensor-Based Control of a Nine-Link biped. *International Journal of Robotics Research*, 9(2):83–98, 1990.

[6] M. Inaba, I. Igarashi, K. Kagami, and I. Hirochika. A 35 dof humanoid that can coordinate arms and legs in standing up, reaching and grasping an object. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 29–36, 1996.

[7] S. Schaal and C. G. Atkeson. From isolation to cooperation: An alternative view of a system of experts. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 605–611. MIT Press, Cambridge, MA, USA, 1996.

[8] A. Takanishi, M. Ishida, Y. Yamazaki, and I. Kato. The Realization of Dynamic Walking by the Biped Walking Robot WL-10RD. In *Proceedings of 1985 International Conference on Advanced Robotics (ICAR'85)*, pages 459–466, 1985.

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

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