Off-Policy Natural Actor-Critic

Takeshi Mori, Yutaka Nakamura, and Shin Ishii
Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0192, Japan
{tak-mori,yutak-na,ishii}@is.naist.jp

Abstract

Recently-developed Natural Actor-Critic (NAC) [1] [2], which employs natural policy gradient learning for the actor and LSTD-Q(λ) for the critic, has provided a good model-free reinforcement learning scheme applicable to high-dimensional systems. Since NAC is an on-policy learning method, however, a new sample sequence is required for estimating sufficient statistics in the policy gradient after the current policy or the policy's parameterization is modified, or the gradient is estimated as biased. Moreover, the control of exploration and exploitation should be performed by a direct operation on the policy, which may be a large constraint on introducing an exploratory factor. To overcome these problems, we propose an off-policy NAC in this article, in which the policy gradient is estimated without any asymptotic bias by using past system trajectories under the control by past policies. In addition, this method allows the exploration to be controlled from the outside of the policy optimization; this is more effective than the direct policy operation. We also propose a hierarchical parameterization of the policy and an off-policy NAC learning for that policy. Computer experiments using a snake-like robot simulator show our new off-policy NAC is so effective that the number of required trajectories is much smaller than that by the on-policy method. Moreover, even when the policy parameterization is high dimensional, our hierarchical off-policy NAC learning successfully obtains a good policy, by controlling the policy's effective dimensionality dynamically. These improvements allow a more efficient application in high-dimensional control problems.

Keywords

reinforcement learning, off-policy method, natural policy gradient, least squares policy evaluation, actor-critic

1 Introduction

Reinforcement learning (RL) is a machine learning framework for reward-related autonomous acquisition of control policies using trial-and-error [3]. Value-based methods like the well-known TD-learning have been successfully applied to various Markov decision problems with state and action spaces. When applied to high-dimensional problems, however, the learning of value functions becomes hard and hence it is difficult to obtain a good controller. Moreover, especially in applications of value-based methods to high-dimensional control problems, the convergence is not guaranteed, because small changes of the value function may cause drastic change in the policy [4] [5].

On the other hand, stochastic policy gradient methods [6] [7] [8] have nice convergence properties, but the estimation of the policy gradient can be difficult for systems with noise and regular policy gradients have been observed to be rather slow in the learning process. Recently, an efficient learning algorithm based on stochastic policy gradient, called natural actor-critic (NAC), was developed [1] [2]. In this method, natural policy gradient learning [9] was employed in the policy improvement step (or actor learning step), and LSTD-Q(λ) [1] [2], which is a kind of least squares policy evaluation methods such as LSTD [10], LSTD(λ) [11], LSTD$Q(\lambda)$ [12], was employed in the policy evaluation step (or critic learning step). NAC is a model-free RL method which has been successfully applied to high-dimensional control problems such as policies learning for biped locomotion [13] and baseball [2].

However, there are two diffculties remain in NAC. First, because LSTD-Q(λ) is a policy evaluation algorithm for a fixed policy, samples generated by previous policies cannot be used for the improvement of subsequent policies, while a lot of samples under the current policy's control are necessary to estimate the action-value function (Q-function). When the number of samples is small, the approximation error of the Q-function becomes large and is no longer useful for policy updates. In [1] and [13], the authors assumed a small change of the policy causes a little change of the value function, and then past trajectories were reused for estimating the current Q-function using forgetting factors in order to decay older samples. Unfortunately, such a procedure can introduce bias into the estimation of policy gradient, whereas decreasing the estimating variance. This is a typical bias-variance problem in the RL field.

Furthermore, it is difficult to deal with the "exploration-exploitation problem" [14] [15]. In stochastic policy gradient RL methods, a parameter representing the level of exploration can be defined such to determine the randomness of stochastic policy, and be controlled according to the performance maximization criterion. However, it is difficult to appropriately tune such a parameter, because when it becomes small in order to exploit many rewards, the stochastic policy makes the state transition non-ergodic. In policy gradient RL methods, non-ergodicity such as assigning zero probability to possible actions, may cause the divergence of the policy gradient, leading to the instability of learning.

These problems can be partly overcome by using off-policy RL methods in which the 'target' policy to be evaluated may differ from the 'behavior' policy by which the sample trajectories are generated [16] [17]. In this study, we then propose an off-policy natural actor-critic (off-NAC) learning method.

In section 2, as a generalization of the existing derivation [12] of LSTD$Q(0)$, we derive LSTD$Q(\lambda)$ from the Bellman equation[1]. Then, we obtain off-policy LSTD$Q(\lambda)$, based on similar derivation to the existing off-policy TD(λ), with a linear function approximator [18]. Our off-policy LSTD$Q(\lambda)$ employs weighted importance sampling [16], which is a new formulation as an off-policy actor-critic learning method. Then we derive off-policy version of LSTD-Q(λ) [1] [2], which uses advantage function [19] and more efficient in actor-critic

[1]Although, in NAC learning, LSTD-Q(λ) [1] [2] is used for the critic learning step, it is rather restrictive because the basis functions which approximate advantage function is necessary. On the other hand, LSTD$Q(\lambda)$ is more general (simple) method, since the update rule is similar to well-known SARSA(λ) [3], where the basis functions are arbitrary. So, we derive LSTD$Q(\lambda)$ at first for wider applicability.

learning scheme. In section 3, we propose an off-policy NAC (off-NAC) learning which uses the off-policy LSTD-Q(λ) for the value learning (critic learning) in NAC. Because off-NAC is able to estimate the policy gradient using effectively past trajectories without introducing any asymptotic bias, the RL process becomes faster and more stable. In addition, because our proposal method is based on an off-policy method, trajectories under the control by an arbitrary behavior policy can be used in principle. Then, we can realize an appropriate control of exploration and exploitation by controlling the generation process of behavior policies. Although off-policy sampling techniques have been applied to policy-gradient-based RL by Shelton [16] and Meuleau et al. [17], those studies do not consider applications to policy-gradient-based actor-critic RL [8] [7] [9] [1]. Our approach is more suitable for applications to higher-dimensional and more complex problems than those by existing off-policy methods. Moreover, we show a novel and interesting application of our off-NAC. Motivated by the biological developmental process, we present a hierarchical parameterization of the target policy and its off-policy NAC learning scheme which incorporates a dynamic freezing-releasing mechanism of policy's dimensionality; a quick learning for low-dimensional and coarse policy is done when the data number is small, while a precise learning for high-dimensional and fine policy is done with a small estimation variance as the data number increases. In section 4, we apply our method to an automatic acquisition problem of locomotion by a snake-like robot simulator. Computer simulations show that our method is more efficient and stabler than NAC, by effectively reusing past trajectories. In addition, our hierarchical off-NAC is effective in an automatic acquisition problem of a high-dimensional policy, which is an important feature when applied to automatic control problems of high-dimensional dynamical systems.

2 Off-policy LSTD$Q(\lambda)$

In this section, we derive LSTD$Q(\lambda)$ and off-policy LSTD$Q(\lambda)$, and off-policy LSTD-Q(λ).

2.1 On-Policy LSTD$Q(\lambda)$

Let the action-value function (Q-function), which indicates the goodness (value) of a pair of a state $s \in S$ and an action $a \in A$ under a certain policy π, be represented as a linear function approximator (critic):

$$Q^\pi(s,a) = \phi(s,a)' w^\pi, \tag{1}$$

where w^π is a q-dimensional parameter vector and $\phi \equiv \phi(s,a)$ is a q-dimensional basis function for the linear function approximator. $\cdot'$ denote a transpose. For all LSTD-based methods [10] [11] [12] [1], the parameter w^π is obtained by the least squares method for given sequences of reward observation.

When introducing an eligibility trace λ [3], there are two points of view, called the forward-view equation and the backward-view equation. The forward-view equation unifies the TD learning ($\lambda = 0$) and the Monte Carlo learning ($\lambda = 1$) as its two extremes, and is based on a theoretical interpretation in which current update is carried out under the prospection of reward and state in the future. In contrast, the backward-view equation is based on a technical interpretation in which a set of eligible state-action pairs is modified by the current TD-error by going back to the past. Usually, an RL algorithm with an eligibility trace is defined as a forward-view equation, and implemented as a backward-view equation after transformed approximately.

According to the backward-view equation of the LSTD$Q(\lambda)$, w^π is calculated as

$$w^\pi = A^{-1}b. \tag{2}$$

Matrix A and vector b are given by

$$b = \sum_{t=0}^{T-1} z_t r_{t+1}, A = \sum_{t=0}^{T-1} z_t(\phi_t - \gamma\phi_{t+1})', \tag{3}$$

where T is the number of steps (samples) in an episode, z is the eligibility vector, and γ is the discount factor [11] [1].

Since there has been no forward view for the λ-based LSTD methods such as LSTD(λ) [11] and LSTD-Q(λ) [1] [2], we here derive the forward-view equation of LSTD$Q(\lambda)$ from the Bellman equation, which provides almost the same result as the backward-view equation (equations (2) and (3)). This derivation is an extension of Lagoudakis' derivation of $\lambda = 0$ [12] to $0 \leq \lambda \leq 1$. First, the Bellman operator T_π is defined as

$$T_\pi Q^\pi(s,a) \equiv r(s,a) + \gamma \int_{x\in\mathbb{S}} P(x|s,a) \int_{u\in A} \pi(u|x) Q^\pi(x,u)\ dudx, \tag{4}$$

where r is the reward function for a state-action pair (s,a), $P(x|s,a)$ is the state transition probability of reaching a next state x from a state s by an action a, and π is a probabilistic policy. By assuming the state and action spaces are finite, equation (4) is represented as a vector form:

$$T_\pi \boldsymbol{Q}^\pi \equiv \boldsymbol{r} + \gamma \boldsymbol{P} \Pi_\pi \boldsymbol{Q}^\pi, \tag{5}$$

where $\boldsymbol{Q}^\pi$ is an $|\mathbb{S}||\mathbb{A}|$-dimensional vector, $\boldsymbol{P}$ is a state transition probability matrix ($|\mathbb{S}||\mathbb{A}| \times |\mathbb{S}|$), Π_π is an action selection probability matrix ($|\mathbb{S}| \times |\mathbb{S}||\mathbb{A}|$), and $\boldsymbol{r}$ is an $|\mathbb{S}||\mathbb{A}|$-dimensional reward vector. Next, we define a λ-Bellman operator T_π^λ as

$$T_\pi^\lambda \boldsymbol{Q}^\pi \equiv \boldsymbol{R}^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} T_\pi^{(n)} \boldsymbol{Q}^\pi, \tag{6}$$

where $T_\pi^{(n)} \boldsymbol{Q}^\pi$ is given by

$$T_\pi^{(n)} \boldsymbol{Q}^\pi = \sum_{k=1}^{n} (\gamma \boldsymbol{P} \Pi_\pi)^{k-1} \boldsymbol{r} + \gamma^n (\boldsymbol{P}\Pi_\pi)^n \boldsymbol{Q}^\pi, \tag{7}$$

using equation (5).

The true Q-function should be a fixed point with respect to the λ-Bellman operator T_π^λ:

$$T_\pi^\lambda \boldsymbol{Q}^\pi = \boldsymbol{Q}^\pi. \tag{8}$$

When employing a linear function approximator like equation (1), however, the approximate Q-function lies within the space spanned by the set of basis functions:

$$\Phi = \begin{pmatrix} \phi(s_0, a_0)' \\ \vdots \\ \phi(s_{|S|}, a_{|A|})' \end{pmatrix},$$

where Φ is an $(|\mathbb{S}||\mathbb{A}|\times q)$ matrix. Because the fixed point of equation (8) is not in the spanned space in general, we approximate the Q-function as an orthogonal projection $(\Phi(\Phi'\Phi)^{-1}\Phi')$ onto the spanned space, such to minimize the L^2-norm from the true Q-function, and then obtain a new fixed point equation [12]:

$$\Phi(\Phi'\Phi)^{-1}\Phi'\boldsymbol{R}^{\lambda} = \Phi w^{\pi}, \tag{9}$$

where

$$\boldsymbol{G}^{(n)} \equiv \sum_{k=1}^{n} (\gamma \boldsymbol{P}\Pi_{\pi})^{k-1}\boldsymbol{r}, \boldsymbol{H}^{(n)} \equiv -\gamma^{n}(\boldsymbol{P}\Pi_{\pi})^{n}\Phi,$$

and

$$\boldsymbol{G}^{\lambda} \equiv (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\boldsymbol{G}^{(n)}, \boldsymbol{H}^{\lambda} \equiv (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\boldsymbol{H}^{(n)}.$$

According to the definition above and equations (6) and (7), equation (9) becomes

$$\begin{aligned} & \Phi(\Phi'\Phi)^{-1}\Phi'(\boldsymbol{G}^{\lambda} - \boldsymbol{H}^{\lambda}w^{\pi}) = \Phi w^{\pi} \\ \Longleftrightarrow \quad & w^{\pi} = (\Phi'(\boldsymbol{H}^{\lambda} + \Phi))^{-1}\Phi'\boldsymbol{G}^{\lambda} \equiv A^{-1}b. \end{aligned}$$

This is the solution of the LSTD$Q(\lambda)$. It was suggested that the LSTD$Q(\lambda)$ is stabler than a least squares method employing a non-orthogonal projection, in a particular case of $\lambda = 0$ [12].

Similar to the characteristics of the usual eligibility trace, the fixed point of the Bellman operator agrees with that of the λ-Bellman operator, when there are infinite number of samples in finite state and action spaces [4]. As λ approaches 0, w^{π} approaches the estimator which provides an accurate value function for the maximum likelihood estimation of the underlying MDP [3]. In a non-Markovian environment which cannot be represented accurately as an MDP model, on the other hand, a λ value close to 1 will work better so that the learning tries to minimize the error with respect to the training data [3]. The LSTD$Q(\lambda)$ is advantageous, because it can handle both of these two cases.

According to the derivation above, the forward-view equation that obtains the matrix A and the vector b at a time step t is given as

$$b_t = \phi_t G_t^{\lambda}, A_t = \phi_t(H_t^{\lambda} + \phi_t)', \tag{10}$$

where G_t^{λ} and H_t^{λ} are

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}, H_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}H_t^{(n)}. \tag{11}$$

G_t^λ and H_t^λ are the summations of

$$G_t^{(n)} = \sum_{l=1}^{n} r_{t+l}\gamma^{l-1}, H_t^{(n)} = -\gamma^n \phi_{t+n}, \tag{12}$$

weighted by the eligibility λ. It should be noted that the λ-return R_t^λ and the n-step return $R_t^{(n)}$ are given using these notations as [3]

$$R_t^\lambda = G_t^\lambda - w' H_t^\lambda, R_t^{(n)} = G_t^{(n)} - w' H_t^{(n)}. \tag{13}$$

The sequence of (r_{t+1}, ϕ_t) has been produced by the current policy π. This forward-view equation is approximately represented as the following backward-view equation (Appendix C), corresponding to equation (3). An RL algorithm is implemented based on this backward-view equation.

$$b \approx \sum_{t=0}^{T-1} b_t, A \approx \sum_{t=0}^{T-1} A_t. \tag{14}$$

2.2 Off-policy LSTD$Q(\lambda)$

Here, we propose an off-policy version of the LSTD$Q(\lambda)$, in which the target policy $\pi(a|s)$ to be evaluated and optimized may be different from the behavior policy $b_p(a|s)$ which conducts trial control to generate sample trajectories. Based on the importance sampling technique, the forward-view equation of the off-policy LSTD$Q(\lambda)$ is given by

$$\bar{b}_t = \phi_t \bar{G}_t^\lambda \prod_{m=0}^{t} \rho_m, \bar{A}_t = \phi_t(\bar{H}_t^\lambda + \phi_t)' \prod_{m=0}^{t} \rho_m, \tag{15}$$

where $\prod_{m=0}^{t} \rho_m$ is an importance sampling weight (ISW), i.e., the reproducibility of a trajectory from $m = 0$ to $m = t$, given by the likelihood ratio of the appearance of this trajectory between by the target policy and by the behavior policy. Corresponding to this modification, equation (12) is replaced by

$$\bar{G}_t^{(n)} = \sum_{l=1}^{n} r_{t+l}\gamma^{l-1} \prod_{k=1}^{l-1} \rho_{t+k}, \bar{H}_t^{(n)} = -\gamma^n \phi_{t+n} \prod_{m=1}^{n} \rho_{t+m}. \tag{16}$$

The ISW terms in equation (16) denote the trajectory reproducibility toward the future. It should be noted that a sequence of (r_{t+1}, ϕ_t) is sampled according to the behavior policy b_p which is arbitrary in principle.

In our importance sampling, the ISW for each sample (each time step) is calculated as well as the Precup's per-decision method [18], based on the observation that an ISW at a certain time step on a trajectory is not dependent on ρ values for subsequent time steps. Although a naive importance-sampling-based method sometimes calculates an ISW for a whole trajectory [3], such a method is not very efficient especially at an early learning stage; when a trajectory becomes long, the ISW of the trajectory often becomes small, leading to the learning inefficiency of the constituent samples by ignoring the proper credit assignment problem. The per-decision method overcomes this problem and is thought to exhibit good learning efficiency.

Proposition 1 Let $A(\bar{A})$ and $b(\bar{b})$ be the summations of $A_t(\bar{A}_t)$ and $b_t(\bar{b}_t)$, respectively, during a certain learning episode with the on-policy (off-policy) LSTD$Q(\lambda)$. Then,

$$E_{b_p}\{\bar{A}|s_0\} = E_\pi\{A|s_0\},\ E_{b_p}\{\bar{b}|s_0\} = E_\pi\{b|s_0\}. \tag{17}$$

This proposition is proved in Appendix A. From proposition 1, replacing equation (10) by equation (15) does not introduce any bias to the estimation of the matrix A and vector b.

Proposition 2 Let $w_K^\pi(\bar{w}_K^\pi)$ be the least squares solution based on the average sufficient statistics, $A^k, b^k(\bar{A}^k, \bar{b}^k)$, over K learning episodes with the on-policy (off-policy) LSTD$Q(\lambda)$. Then, $\bar{w}_K^\pi$ converges to w_K^π with probability 1. This proposition is proved in Appendix B.

In the existing off-policy RL methods, such as the off-policy TD(λ) [18] and the off-policy method for Monte Carlo return [16], the occurrence probability of a target variable was corrected by an ISW so that the estimator by the corresponding on-policy method was obtained without any bias. In the off-policy LSTD$Q(\lambda)$, on the other hand, $E_{b_p}\{\bar{w}_K^\pi|s_0\} = E_\pi\{w_K^\pi|s_0\}$ holds when $K \to \infty$, while a bias due to the sampling bias remains when $K < \infty$. This is because the least squares method performs the optimization based on the given data set. In a general weighted least squares method:

$$(XWX)^{-1}XWY, \tag{18}$$

with a diagonal positive weight matrix W, a data point (X_i, Y_i) with a large weight W_i has a high priority in the optimization. Since both of the numerator XWY and the denominator (XWX) include W, however, the effect of W is normalized. The off-policy LSTD$Q(\lambda)$ is a kind of this weighted least squares method, where W corresponds to the ISWs for the samples.

An off-policy estimation method based on the weighted importance sampling:

$$\frac{\sum_k^K f_k \frac{\pi_k}{b_{p_k}}}{\sum_k^K \frac{\pi_k}{b_{p_k}}} \tag{19}$$

has been proposed [16] [17] [3] [20]. This method realizes an unbiased estimation when $K \to \infty$, but biased when $K < \infty$, because of the ISW normalization. It is known, however, that importance sampling methods of this type work more efficiently than unbiased estimation methods for many RL tasks. Since an unbiased estimation method based on importance sampling obtains an estimation which is reliable only around the observed samples, the policy improvement based on the discontinuously-estimated return may be stuck. When using a weighted estimation method, on the other hand, the policy improvement works relatively well, because the weighted estimator tends to approximate over the data area where no sample has been observed by an extrapolation (smoothing) effect [16]. Moreover, although the unbiased estimator sometimes diverges, the weighted method is likely to suppress the variance and hence stable [20].

Our off-policy LSTD$Q(\lambda)$ has a similar form (equations (15), (16), (14), (2)) to that of the weighted importance sampling method above, and then we expect it enjoys similar merits. In addition, our study shows an incremental algorithm of a weighted importance sampling method.

The next proposition guarantees the convergence of our off-policy LSTD$Q(\lambda)$.

Proposition 3

(a) For every state-action pair, the stationary distribution with the behavior policy has positive probability.

(b) The target and behavior policies are proper; namely, a control sequence with each of these policies reach a terminal state after infinite time.

(c) Φ has a full rank.

(d) A real number χ exists, such that $E_{p_b}\{\prod_{t=0}^{T} \rho_t^2\} < \chi$ for any $s_0 \in \mathbb{S}$.

Under these conditions, the learning sequence of the critic parameter w_t converges to the optimal one w^* with probability one, for any $\lambda \in [0, 1]$.

The proof is possible by extending the convergence proof of the LSTD(λ) by [21] to the one employing basis functions in state and action spaces. Conditions (a) and (b) are satisfied by restricting the target and behavior policies such to make the system ergodic, which is an often-applied restriction. Condition (c) is not practical because it needs a memory amount comparable to a table-look-up representation. However, it may not be satisfied strictly; in the experiments shown later, good results are actually obtained by using a fairly small number of basis functions. Condition (d) is satisfied in usual finite episode tasks.

2.3 Quasi-active sampling

In off-policy methods, the sampling is no longer determined by the policy; therefore it is crucial for the success of off-NAC to have an appropriate sampling method. Since the learning accuracy of a state-action pair is dependent on the number of visiting the pair, it is important to sample state-action pairs to which we put some special attention. Such a sampling way regarding the attention is called an active sampling, and is an important issue in RL [22].

In real world problems, however, it is difficult to visit repeatedly state-action pairs of interest within trajectories, because the underlying system dynamics cannot be ignored, then the active sampling suffers from a large restriction. Since the requirement for visiting each sample should change after the target (behavior) policy is updated and hence the system dynamics is changed, it is difficult to determine the requirement before sampling.

In [18], 'restarting' technique is proposed, which modifies the initial state distribution for learning into the one different from the distribution of initial states of the system. However this method is only used for avoiding slightly the ISW reduction through trajectory [18]. Therefore we extend this technique to several usage, called quasi-active sampling.

In this method, a non-negative random variable g_t which prescribes the initial state distribution of learning is introduced, and equation (10) is rewritten as

$$\tilde{b}_t = \phi_t \bar{G}_t^\lambda \sum_{k=1}^{t} g_k \prod_{m=k}^{t} \rho_m, \tilde{A}_t = \phi_t(\bar{H}_t^\lambda + \phi_t)' \sum_{k=1}^{t} g_k \prod_{m=k}^{t} \rho_m. \tag{20}$$

When $g_0 = 1$ and $g_t = 0(\forall t > 0)$, it is consistent with the standard off-policy LSTD$Q(\lambda)$ which uses the initial state distribution of the system. With another setting, the initial state distribution of learning is different from that of the system, such that the t-th step state with a positive g_t value is paid an attention.

When we have some *a priori* knowledge about interesting samples, the initial state distribution of learning, g, can be set based on the knowledge. For example, when the interest amount of states is represented as a specific measurement function $f(s)$, the algorithm can be modified by setting $\{g_0, \dots, g_{T-1}\} = \{f(s_0)/Z, f(s_1)/Z, \dots, f(s_{T-1})/Z\}, Z = \sum_{t=0}^{T-1} f(s_t)$ for given set of samples $\{s_0, s_1, \dots, s_{T-1}\}$.

The quasi-active sampling technique can also be used for putting special attention to events occurring on trajectories. By assigning large g values to the samples just before interesting events have occurred, the learning that puts emphasis on those events is promoted. In a maze task, for example, the states just before reaching the goal state are important and should be emphasized. In a navigation task of a snake-like robot simulator shown in the simulations later, the states just before running into the wall is important. Since these events which may occur during an episode are dependent on the behavior policy and it is difficult to define the states of interest before a trajectory is actually produced, we cannot place such priority into the initial state distribution of the system.

In addition, the control of g can be used for the reduction of the bias of the initial state distribution of the system. In real problems such as robot control, it is usually difficult to set an unbiased initial distribution of the system. When the distribution of desirable trajectories can be represented as $f(s)$, for example, setting g like above could reduce the bias of the original initial distribution.

Moreover, according to off-policy learning methods, an ISW becomes small and the learning becomes inefficient as the trajectory gets long; an appropriate re-setting of g at 1 during the trajectory (episode) recovers the ISW so that the problem above can be overcome to some extent. The last application is proposed in [18].

By transforming equation (20) into a backward-view equation, we obtain Algorithm 1 (for transformation, see Appendix C). In Algorithm 1, a superscript and subscript index a single trajectory and a time step in a trajectory, respectively ($t = 0, \dots, T^k$ on the k-th trajectory). In line 13, δ is a positive scalar and I is the identity matrix. This is a regularization term that stabilizes the inverse calculation of matrix A, and this regularization term introduces no bias to the parameter obtained by the off-policy LSTD$Q(\lambda)$. A sequential calculation method of the inverse of matrix A is described in Appendix D.

Algorithm 1. Off-Policy LSTD$Q(\lambda)$

Input: $K, \pi, b_p^{1:K}, T^{1:K}, \phi_{0:T^K}^{1:K}, r_{1:T^K}^{1:K}, g_{0:T^K-1}^{1:K}$
// K: number of stored trajectories, π: target policy, $b_p^{1:K}$: behavior policies
// $T^{1:K}$: number of time steps in episodes, $\phi_{0:T^K}^{1:K}$: basis functions
// $r_{1:T^K}^{1:K}$: rewards, $g_{0:T^K-1}^{1:K}$: start state variables
1 for $k = 1$ to K {
2 for $t = 0$ to T^k {
3 $\rho_t = \pi_t^k / b_{p_t}^k$ //ρ_t: stepwise-responsibility of k-th trajectory
4 }
5 $c_0 = \rho_0 g_0^k, z_0 = c_0 \phi_0^k$ //z_0: k-th initial eligiblity ($q \times 1$) vector
6 for $t = 1$ to $T^k - 1$ {
7 $c_t = \rho_t c_{t-1} + g_t^k$
8 $z_t = \gamma\lambda\rho_t z_{t-1} + \phi_t^k c_t$ //z_t: k-th eligibility ($q \times 1$ in each t) vector
9 }
10 $A^k = \sum_{t=0}^{T^k-1} z_t(\phi_t^k - \gamma\rho_{t+1}\phi_{t+1}^k)'$ // A^k: k-th A ($q \times q$) matrix
11 $b^k = \sum_{t=0}^{T^k-1} z_t r_{t+1}^k$ //b^k: k-th b ($q \times 1$) vector
12 }
13 $A = \sum_{k=1}^{K} A^k + \delta I, b = \sum_{k=1}^{K} b^k$ //A ($q \times q$) matrix and b ($q \times 1$) vector
14 $w = A^{-1} b$ //w: critic parameter ($q \times 1$) vector
Output: w

2.4 Off-policy LSTD-Q(λ)

Here, we drive off-policy LSTD-Q(λ).

LSTD-Q(λ) [1] [2] is the least squares policy evaluation method which needs the basis functions for approximating the advantage function, and it is more efficient than LSTD$Q(\lambda)$. Q-function can be represented by the summation of the state value function V, which indicates the goodness of a state s, and the advantage function$f$ [19], which indicates that of selecting an action a in s: $Q^\pi(s,a) = V^\pi(s,a) + f^\pi(s,a) = w'\psi(s) + \hat{w}'\hat{\psi}(s,a)$. Because the expectation of the advantage function with respect to the action is 0, it is necessary for the basis functions $\hat{\psi}(s,a)$ to satisfy $\int_a \pi(a|s)\hat{\psi}(s,a)\, da = 0$ [1] [2]. Here, new basis functions $\tilde{\phi}_t^k = [[\psi_t^k]', [\hat{\psi}_t]']'$ and $\bar{\phi}_{t+1}^k = [[\psi_{t+1}^k]', 0']'$ are defined, where $\tilde{\phi}_t$ is used for approximated Q-function at t and $\bar{\phi}_{t+1}^k$ is used for that in one-step-ahead eligibility prediction at t. $\bar{\phi}_{t+1}^k$ does not have $\hat{\psi}_{t+1}'$, because the expectation of TD error with respect to the next action a_{t+1} can be marginalized out in advance. This operation can reduce variance of the TD estimation [1] [2].

In off-policy LSTD-Q(λ), $\tilde{\phi}_{0:T^k-1}^{1:K}$ and $\bar{\phi}_{1:T^k}^{1:K}$ are used as input instead of $\phi_{0:T^k}^{1:K}$ in algorithm 1. Then, we substitute $c_0 = \rho_0 g_0^k, z_0 = c_0\tilde{\phi}_0^k$ for line 5, and $z_t = \gamma\lambda\rho_t z_{t-1} + \tilde{\phi}_t^k c_t$ for line 8, and $A^k = \sum_{t=0}^{T^k-1} z_t(\tilde{\phi}_t^k - \gamma\bar{\phi}_{t+1}^k)'$ for line 10. Because of the expectation operation, the basis function vector $\bar{\phi}_{t+1}$ is not multiplied by the importance weight ρ_{t+1} due to a_{t+1} (line 10).

3 Off-policy natural actor-critic

In this section, we propose an off-policy natural actor-critic (off-NAC) algorithm (Algorithm 2), which enables us to estimate the matrix A and the vector b without introducing any asymtotical bias by using past trajectories under the control by past policies. This reusage of past trajectories enlarges the number of effective samples and hence makes the estimation variance small in comparison to the case that uses only trajectories with the current policy. Then, it is expected that a good policy parameter can be obtained stably and efficiently. Moreover, our method can incorporate explicitly an exploration-exploitation control by introducing a 'meta-' control of behavior policies from the outside of the optimization of the target policy.

3.1 Off-NAC algorithm

This algorithm provides an episodic learning scheme; the policy parameter θ is fixed within a single episode. A learning episode terminates at time T or when at time T_{s_E} a terminal state s_E is observed. The stochastic policy is parameterized as π_θ with a parameter θ, and θ_0 denotes the initial policy parameter. In each learning episode, an initial state s_0 is sampled from a fixed distribution of initial states, $p(s_0)$ (line 4). The variable for setting initial state distribution, g_t, is determined from the past trajectory $\{s^1_{0:T^1}, a^1_{0:T^1-1}, r^1_{1:T^1}\}$ according to the quasi-active sampling (line 9).

$D^{1:K} \equiv \{D^1, D^2, \dots, D^K\}$ is the set of K trajectories, each of which is controlled by the current policy or a past policy: $D^k \equiv \{T^k, s^k_{0:T^k}, a^k_{0:T^k-1}, r^k_{1:T^k}, g^k_{0:T^k-1}, b^k_{p_{\Xi_{l-k+1}}}\}$, where $b_{p_{\Xi_{l-k+1}}}$ is the behavior policy that has been used for generating the k-th trajectory (lines 2 and 10). Ξ denotes the parameter of a behavior policy. K_0 is the number of trajectories to be stored before learning, and its setting at an appropriate positive value is useful for suppressing the variance of initial estimation of the policy gradient (line 11). M is the iteration number of policy improvement during a single (control) episode (line 13).

At the onset of a single learning episode, parameter Ξ of the behavior policy is drawn from a normal stochastic process:

$$\Xi \sim \mathcal{N}(\theta, \sigma), \tag{21}$$

around the current parameter θ of the target policy added by a noise with a standard deviation (s.d.) σ; exploration and exploitation are controlled by tuning the s.d. σ (lines 3 and 33). A large variation (large σ) of generation of behavior policies, especially at an early learning stage promotes the exploration, while a small variation that makes b_p similar to the target policy π_θ, especially at a later learning stage, leads to the exploitation. This meta-control of the stochastic generation process (21) of behavior policies, instead of controlling the target policy itself, is beneficial for avoiding the divergence of the policy gradient, if we employ a positive s.d. σ. Here, we introduced a simple example of meta-control, but another meta-control is of course possible.

Algorithm 2. Off-Policy Natural Actor-Critic

Input: K,K_0,M,θ_0,π_{θ_0},T,σ,$D^{1:K}$
// K: maximum number of trajectories to be stored during learning
// K_0: number of trajectories to be stored before learning
// M: iteration number of policy improvement during a single (control) episode
// θ_0: initial policy parameter ($q_\theta \times 1$) vector, π_{θ_0}: initial stochastic policy
// T: maximum time steps in a episode, σ: exploration factor
// $D^{1:K}$: set of K trajectories
for $l = 0, 1, 2, \dots,$ {
  $D^{2:K} = D^{1:K-1}$ //dismiss oldest sample trajectory
  $\Xi_l \sim \mathcal{N}(\theta_l, \sigma)$ //generate behavior policy
  $s_0^1 \sim p(s_0)$ //generate initial state
  while $t \neq (T$ or $T_{s_E})$ {
    $a_t^1 \sim b^1_{p_{\Xi_l}}, s^1_{t+1} \sim P_t, r^1_{t+1} \sim R_{t+1}$ //generate sample trajectory
  }
  $T^1 = \min(T, T_E)$ //determine terminal time
  Determine g^1 from $\{s^1_{0:T^1}, a^1_{0:T^1-1}, r^1_{1:T^1}\}$ //determine start state variable
  $D^1 = \{s^1_{0:T^1}, a^1_{0:T^1-1}, r^1_{1:T^1}, g^1, b^1_{p_{\Xi_l}}, T^1\}$ //store sample trajectory
  if $l \geq K_0$ { //judge the stored trajectories before learning
    $\theta_{l+1} = \theta_l$
    for $m = 1$ to M { //iteration using same data sets
      for $k = 1$ to $\min(l+1, K)$ {
        for $t = 0$ to T^k {
          $\rho_t = \pi_{\theta_{l+1},t} / b^k_{p_{\Xi_{l-k+1}},t}$
        }
        $\tilde{\phi}^k_{0:T^k-1} = [[\psi^k_{0:T^k-1}]', [\nabla_\theta \log \pi_{\theta_{l+1},0:T^k-1}]']'$ //k-th basis vector
        $\bar{\phi}^k_{1:T^k} = [[\psi^k_{1:T^k}]', 0']'$ //k-th basis ($q \times 1$ in each t) vector
        $c_0 = \rho_0 g_0^k$ //for one-step-ahead eligibility prediction
        $z_0 = c_0 \phi_0^k$
        for $t = 1$ to $T^k - 1$ {
          $c_t = \rho_t c_{t-1} + g_t^k$
          $z_t = \gamma\lambda\rho_t z_{t-1} + \tilde{\phi}_t^k c_t$
        }
        $A^k = \sum_{t=0}^{T^k-1} z_t(\tilde{\phi}^k_t - \gamma\bar{\phi}^k_{t+1})'$
        $b^k = \sum_{t=0}^{T^k-1} z_t r^k_{t+1}$
      }
      $A = \sum_{k=1}^{\min(l+1,K)} A^k + \delta I, b = \sum_{k=1}^{\min(l+1,K)} b^k$
      $[v', \eta']' = A^{-1}b$ //v: $((q - q_\theta) \times 1)$, η: natural policy gradient $(q_\theta \times 1)$
      $\theta_{l+1} = \theta_{l+1} + \alpha\eta$ //policy improvement
    }
    Reduce σ //reduce exploration factor for convergence
  }
}
Output: π_{θ^*}

In the natural policy gradient learning, the Q-function is approximated as a linear combination of basis (compatible) functions $\nabla_\theta \log \pi_\theta$, and then the linear coefficient η becomes equivalent to the natural policy gradient (lines 30 and 31) [9]. In this method, however, the parametric linear combination of the compatible functions: $\eta' \nabla_\theta \log \pi_\theta$ essentially expresses the advantage function: $Q^\pi(s,a) - V^\pi(s)$, because $\int_a \pi_\theta \nabla_\theta \log \pi_\theta da = 0$ [1] . If we employ the TD learning as a learning method for the Q-function, it is then appropriate to introduce some additional eligible basis functions $\psi(s)$ which approximates the state value function $V^\pi(s)$, so that the parametric linear combination is able to well approximate the Q-function[2]. Acoording to off-policy LSTD-Q(λ), in line 26, $\bar{\phi}_{t+1}$, used for one-step-ahead eligibility prediction, is the expected basis function vector with respect to the action at the next time step, a_{t+1}, and then its compatible functions are 0, i.e., $\bar{\phi}_t = [\psi_t', 0']'$ (line 19), and $\bar{\phi}_{t+1}$ is not multiplied by the importance weight ρ_{t+1} due to a_{t+1} (line 26).

In the off-policy TD(λ) learning [18], the critic's parameter is updated whenever a TD error is given, which makes the parameter depend highly on the latest update, then the merit of reusage of past trajectories diminishes. In addition, although the TD(λ) learning requires an appropriate setting of the learning coefficient, this is not easy especially when the number of samples is large. In our off-NAC based on the off-policy LSTD-Q(λ), on the other hand, samples are efficiently used because all of the trajectories are treated equally with the weighted importance sampling technique, and the estimation will become accurate as the number of stored trajectories increases.

In an existing study, a weighted importance sampling technique was used for estimating the summation of rewards received along control trajectories [16]. Although this study was the first application of an off-policy method to policy-based RL schemes, it did not deal with the critic's learning of the policy-gradient-based actor-critic learning [8] [7] [9] [1], where the value-function approximator accumulates the information of the target MDP efficiently based on the Bellman equation.

In the off-policy REINFORCE [17], an off-policy control of exploration and exploitation was presented. Since this technique was specially devised for REINFORCE [6], however, it suffers potentially from slow learning, and this method is not suitable for an application to the policy-gradient-based actor-critic learning either.

3.2 Hierarchical off-NAC

Although there is no asymptotical estimation bias in our off-NAC, the estimation variance could be large like in the usual on-NAC [1] [2] when the number of stored trajectories is small especially at an early learning stage. This problem can be avoided to some extent by setting a large K_0 value, but this modification is not very desirable because the change in the target policy makes ISW small and hence the data efficiency gets lower. When the dimensionality of the policy's parameter is small, on the other hand, updating the parameter to a good direction is possible though the variance may still remain. Based on these observations, we propose a new RL architecture for training a hierarchically structured policy; when the number of data is small, a policy with a low dimensionality is trained, and as the number of data increases, the frozen dimensionality of the policy is gradually released to be

[2]In the case of $\lambda < 1$, how we select the appropriate basis functions $\psi(s)$ is open. When $\lambda = 1$, only a $\psi(s) = 1$ is necessary, but it is rather slow [1] [2]

trained. This learning procedure could be interesting because of its similarity to biological developmental process [23].

Let a target policy be represented as a hierarchical binary tree (Fig.1), and we propose a hierarchical off-NAC learning in which the learning processes from the higher layer to the lower layer along the binary tree. Although the hierarchical structure of the policy is not necessarily restricted to be binary, we here present the simplest binary case as an example. The policy (Ψ^0) is given by

$$\Psi^0 \equiv \sum_{i=1}^{2^n} \theta_i^n \psi_i^n, \tag{22}$$

where ψ_i^n and θ_i^n are the i-th basis function and the corresponding linear coefficient, respectively, at the n-th layer. Equation (22) defines the policy parameterization at the n-th layer, so that the off-NAC learning can be performed at this layer independently of the other layers. We also define the output of the i-th node at the n-th layer as

$$\Psi_i^n \equiv \theta_{2i-1}^{n+1} \Psi_{2i-1}^{n+1} + \theta_{2i}^{n+1} \Psi_{2i}^{n+1}, \tag{23}$$

i.e., the weighted sum of the outputs of the two children nodes.

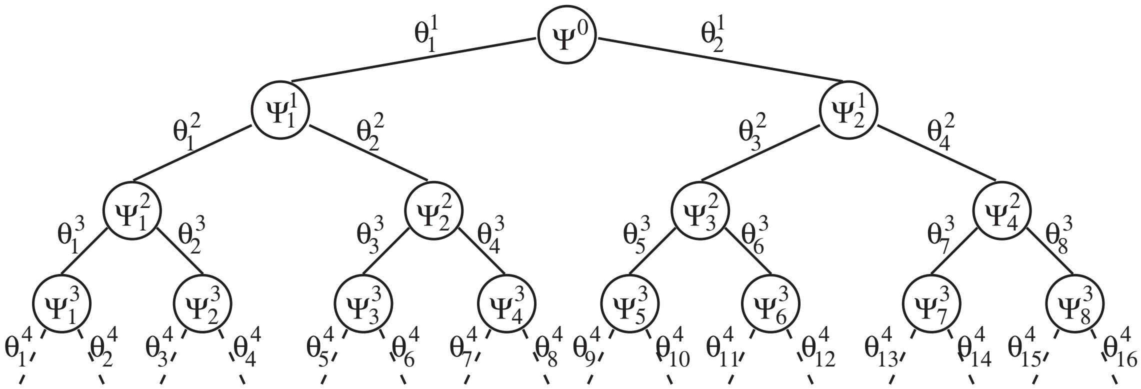


Figure 1: Binary parameterization of the Policy

From equations (22) and (23), the basis function at the n-th layer should satisfy

$$\psi_i^n = \Psi_i^n \prod_{m=1}^{n-1} \theta_{\zeta_m(i)}^{n-m}. \tag{24}$$

$\zeta_j(i) \equiv \lfloor (\zeta_{j-1}(i) + 1)/2 \rfloor$, $\zeta_0(i) \equiv i$, where $\lfloor \cdot \rfloor$ denotes the floor integer. From equation (24), a basis function at the n-th layer, ψ_i^n can be calculated by the output of the corresponding node, Ψ_i^n, which integrates the lower information according to equation (23), and $\prod_{m=1}^{n-1} \theta_{\zeta_m(i)}$, which is given by the policy parameters at the higher layers. If the outputs of leaf nodes, $\Psi_i^N (i = 1, \ldots, 2^N)$, where N denotes the height of the binary tree, is given from sample trajectories, this hierarchical setting allows us to estimate the policy parameters θ_i^n from the root node to the leaf nodes according to equation (24). Equation (22) states that there are 2^n policy parameters at the n-th layer. Therefore, it is a possible algorithm that the parameters at a higher layer are optimized when the number of data is small, and those at a lower layer are gradually optimized as the data number increases, so that the bottom layer with the complete (probably high-dimensional) parameterization of the policy is trained using a sufficient number of data.

For the l-th episode, the probability of releasing (training) the parameters at the n-th layer is, for example, given as

$$p(n|l) \propto \mathcal{N}(l; K_0 + \mu_n, \sigma_H^2), \tag{25}$$

where a gradual release scheme along the binary tree is realized by $\mu_1 \leq \mu_2 \leq \ldots \leq \mu_N$. Since the policy optimization process is iterated M times for a single learning episode in Algorithm 2 (line 13), this selection and learning of the n-th layer is carried out M times in the episode. By setting $\mu_n (n = 1, \ldots, N)$ appropriately, the learning at higher layers, which is easy to be optimized, is carried out at an early learning stage, and the learning of lower layers, which have high approximation ability but difficulty in the optimization, is carried out at a late learning stage.

4 Experiment

In this section, we apply our off-NAC to an automatic control problem of locomotion by a snake-like robot simulator. The snake-like robot simulator is composed of ten rigid links, depicted in Fig.2(b). Each link possesses passive wheels on its both sides, and its length, width and height are 0.15m, 0.1m and 0.08m, respectively. The robot was controlled by torques applied to nine joints, and has a sensor which detects the nearest wall to itself. The task by the snake-like robot is to go through a crank course, depicted in Fig.2(a), without hitting the wall, and the objective of the simulation is to obtain an actor that allows the robot to complete the task. We set the width and the total length of the crank course at 2m and 13.7m, respectively.

Although the snake-like robot has an action space of relatively high dimensionality corresponding to nine control torques which can be applied to the nine joints, the actual control space would be much smaller; the locomotion of a crawling snake is usually rhythmic. In this study, we then employ a central pattern generator (CPG) which is suitable for controlling rhythmic locomotions.

We here assume there is one CPG, and its state at time t is represented as $\Lambda(t)$. A torque applied to the i-th joint at time t, $\tau_i(t)$, is assumed to be generated as

$$\tau_i(t) = \xi_1 \cos(\Lambda(t) + i\xi_2) + \xi_3, \tag{26}$$

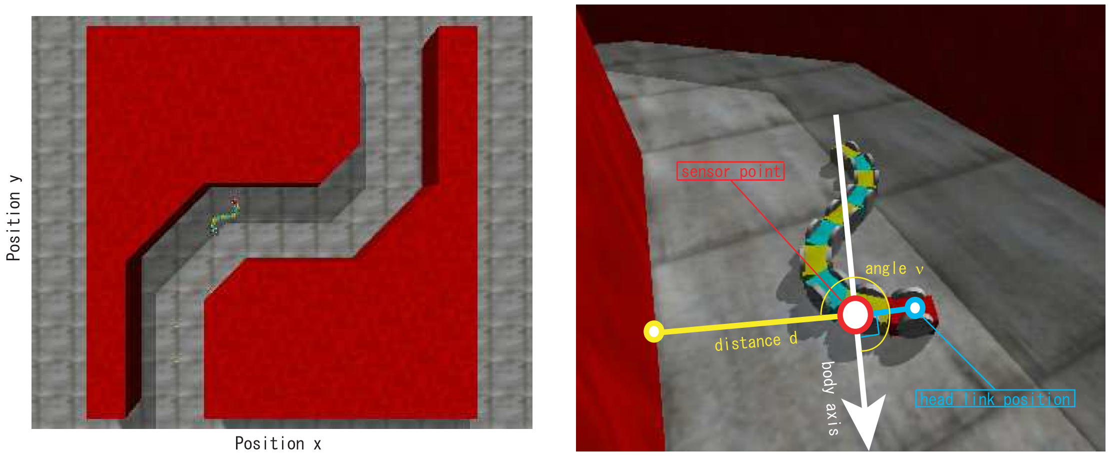
where ξ_1, ξ_2 and ξ_3 denote amplitude, phase interval and bias, respectively, of the control torque. ξ_2 was fixed at $2\pi/9$, so that the phases of torques applied from the first and to the ninth joints correspond to the one cycle (2π) of the cosine function above. Since the optimal values of ξ_1 and ξ_3 depend on the environment surrounding the snake-like robot, such as the friction, and the slope or landscape of the ground, these values can be determined within the RL. For the sake of simplicity, however, ξ_1 and ξ_3 were fixed at 0.5 and 2.8, respectively, because the task here is to avoid obstacles (walls) in the environment.

The dynamics of the CPG is given by

$$\Lambda(t+1) = \Lambda(t) + \omega(t), \tag{27}$$

where ω is the input (corresponding to an angular velocity) to the CPG, either of a constant input (ω_0) or an impulsive input ($\omega_0 + \Delta\omega$), where $\Delta\omega$ is fixed at $\pi/15$. The snake-like robot controlled by this CPG goes straight during ω is ω_0, but turns when an impulsive input is added at a certain instance; the turning angle depends on both the CPG's internal state Λ and the intensity of the impulse. An action is to select an input out of these two candidates, and the controller represents the action selection probability which is adjusted by RL.

To carry out RL with the CPG, we employ the CPG-actor-critic model [24], in which the physical system (robot) and the CPG are regarded as a single dynamical system, called the CPG-coupled system, and the control signal for the CPG-coupled system is optimized by RL.



(a) Crank course (b) Sensor setting

Figure 2: Crank course and sensor setting

4.1 Experimental settings

We here describe the settings for experiments which examine the efficiency of sample reusage and the exploration ability of our new RL method, off-NAC.

It is assumed the sensor detects the distance d between the sensor point (red circle in Fig.2(b)) and the nearest wall, and the angle ν between the proceeding direction and the nearest wall, as depicted in Fig.2(b). The sensor point is the projection of the head link's position onto the body axis which passes through the center of mass and is in the proceeding direction, of the snake-like robot. The angular sensor perceives the wall as to be within $[0, 2\pi)$ [rad] in anti-clockwise manner, and $\nu = 0$ indicates that the snake-like robot hits the wall.

According to the present CPG-actor-critic model, the actor observes a state of the CPG-coupled system, $s \equiv [\Lambda, d, \nu]'$, and outputs an action a which denotes whether the impulsive input is added or not. The action selection probability is defined by a binomial distribution:

$$a \sim Bi(1, p_b) = p_b^a (1 - p_b)^{1-a}, \tag{28}$$

where the frequency p_b is given by a sigmoidal function:

$$p_b = \frac{1}{(1 + \exp(-\Psi^0 \cos(\Lambda - 1.5) + 10))}. \tag{29}$$

Ψ^0 is defined by

$$\Psi^0 = \sum_{i=1}^{4} \theta_i (\psi_i - \psi_{i+4}), \tag{30}$$

where $\{\theta_i | i = 1, \dots, 4\}$ are policy parameters. We prepared eight basis functions, and ψ_i is the i-th basis function given as a normalized Gaussian function: $\psi_i(\bar{s}) = \Upsilon_i(\bar{s}) / \sum_{j=1}^{8} \Upsilon_k(\bar{s})$. Here, $\Upsilon_i(\bar{s})$ is a Gaussian for a state variable $\bar{s} \equiv [d, \nu]'$ and calculated by $\Upsilon_i(\bar{s}) = \exp(-(d - \mu_{d,i})^2 / 2\sigma_d^2 - (\nu - \mu_{\nu,i})^2 / 2\sigma_\nu^2)$. Means $\mu_{d,i}$ and $\mu_{\nu,i}$ were fixed at $\{0, 1, 0, 1, 0, 1, 0, 1\}$ and $\{0, 0, \frac{2}{3}\pi, \frac{2}{3}\pi, 2\pi, 2\pi, \frac{4}{3}\pi, \frac{4}{3}\pi\}$, respectively, for the eight basis functions. S.d.'s σ_d and σ_ν were also fixed at 0.3 and $2\pi/3$, respectively. In this case, the compatible function (partial derivative with respect to a policy parameter) of the i-th policy parameter θ_i is calculated as $\nabla_{\theta_i} \log \pi(s, a) = (a - p_b)(\psi_i - \psi_{i+4}) \cos(\Lambda - 1.5)$. We employ additional basis functions $f_i = \psi_i + \psi_{i+4} (i = 1, \dots, 4)$, so that the state value function was approximated as $V^\pi(s) \approx \sum v_i f_i$.

Episodes of length 3000 were considered in the learning setup, and the policy parameter θ was updated after each episode. At the onset of each learning episode, the snake-like robot was set at an initial position such that the x-axis position of the head link was randomly selected from $x \in [-0.4, 0.4]$ and the y-axis position was fixed at $y = 1.37$, and the body axis was set to be perpendicular to the nearest wall's direction, for the first episode or when the snake-like robot was stuck in the previous episode. Otherwise, the learning episode was restarted from the terminal state in the previous episode.

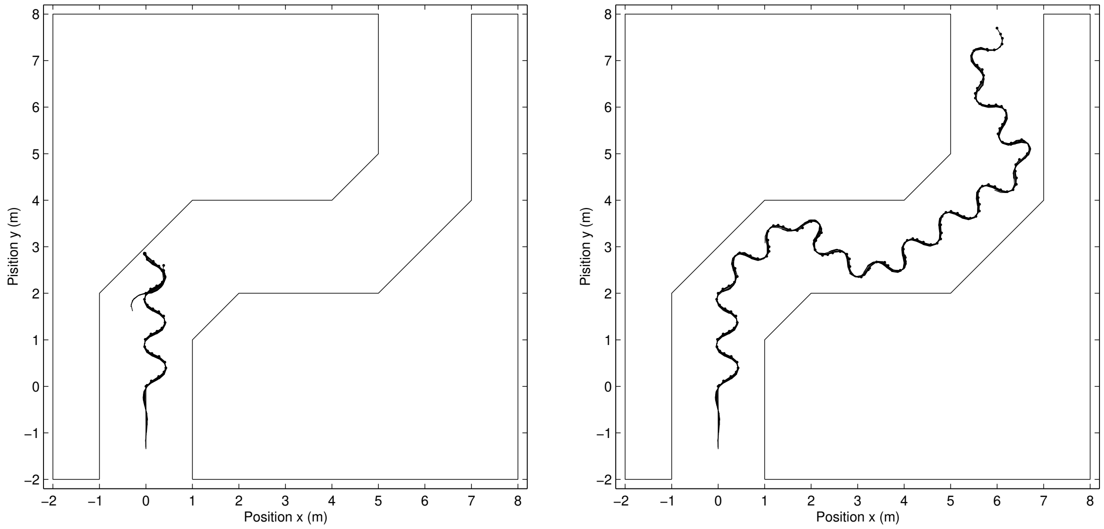
A negative reward -1 was given when the robot ran into the wall, or no reward (0) was given.

The variable g, which represents the initial distribution of learning, was set at $g_0 = 1$ and $g_t = 0 (\forall t > 0)$; namely, the simplest off-policy LSTD-Q(λ) setting.

4.2 Results by off-NAC

We here show the results by the off-NAC learning. In this experiment, initial policy parameter θ_i^b was drawn randomly from the range of $[-10, 0]^4$, and happened to be $\theta_i^b = \{-7.52, -4.75, -1.68, -5.46\}$. After learning, however, the policy parameter became $\theta_i^a = \{-33.52, -5.64, -18.45, -6.06\}$. To examine the performance of the controller before and after the learning, a control task was executed during 13,000 time steps (Fig. 3). The initial posture of the snake-like robot in the control task was set at the same posture as in a learning episode. When the robot was controlled by the policy parameter before learning,

θ_i^b, the robot soon hit the wall and got stuck (Fig. 3(a)). In contrast, the policy parameter after learning, θ_i^a, allowed the robot to pass through the crank course repeatedly (Fig. 3(b)). This result shows a good controller was acquired by our off-NAC.



(a) Before learning (b) After learning

Figure 3: Behaviors of snake-like robot

Figure 4 shows the action a (accompanied by the number of aggregated impulses), the internal state of the CPG, $\cos(\Lambda)$, the relative angle of the wall $\bar{\nu}$ (for visibility, the range $\nu \in [\pi, 2\pi)$ was moved to $\bar{\nu} \in [-\pi, 0)$. If $\bar{\nu}$ is negative (positive), the wall is on the right-hand-side (left-hand-side) of the robot), and the distance between the robot and the wall, d. When the wall was on the right-hand-side, the impulsive input was applied ($a = 1$) when $\cos(\Lambda)$ was large; this control was appropriate for walking the robot turn left. In addition, the impulsive input was added many times especially when the distance d became small.

We compare the performance of our off-NAC with that of the on-NAC [1] [2]. We prepared ten initial policy parameters randomly drawn from $[-10, 0]^4$, and they were adjusted by the two RL methods. Learning parameters, such as the learning coefficient, were best tuned by hand. To examine solely the efficiency of the reusage of past trajectories here, we did not use any exploratory behavior policies ($\sigma = 0$), i.e., trajectories were generated according to the current policy in each episode. The maximum number of stored sample sequences (K) was set at 10, and the number of initially-stored sample sequences (K_0) was set at 0. Eligibility λ was set at 0.999. We regarded a learning run as successful when the actor was improved such to allow the robot to pass through the crank course twice without any collision with the wall, after 500 learning episodes.

Table 1 shows the number of successful learning runs and the average number of episodes until the robot was first successful in passing through the crank course. With respect to both of the criteria, our off-NAC overwhelmed the on-NAC.

Figure 5 shows a typical learning run by the off-NAC (Fig. 5(1a-c)) and the best run by the on-NAC (Fig. 5(2a-c)). Panels (a), (b) and (c) show the average reward in a single

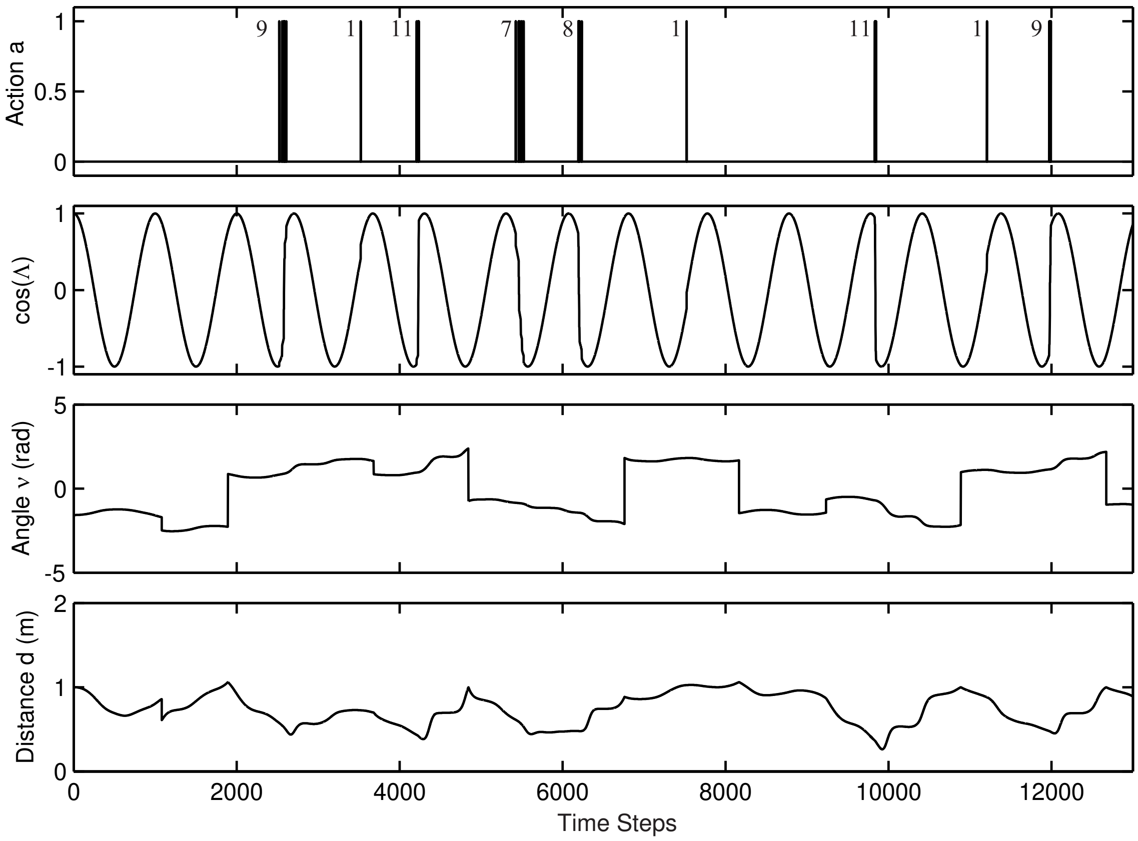


Figure 4: Control after learning

Table 1: The performance comparison of off-NAC and on-NAC

	Number of successful learning runs	Average length of episodes
off-NAC	10/10	20.6
on-NAC	7/10	145.7

episode, the policy parameter, and the number of episodes elapsed from the last time when the robot had hit the wall, respectively. A circle point indicates that the robot passed through the crank. The policy parameter converged when trained by the off-NAC, while did not converge even in the best run by the on-NAC. Because the policy gradient was estimated by focusing on a trajectory produced by the current policy in the on-NAC, the parameter became unstable and would be degraded after a single unlucky failure episode. A small learning coefficient is able to enhance the stability in the on-NAC, but the learning will be further slow.

To examine the exploration ability of our off-NAC, we compared an off-NAC$_0$ algorithm whose σ was fixed at $\sigma_0 = 0$ and an off-NAC$_2$ algorithm in which the initial value of σ was

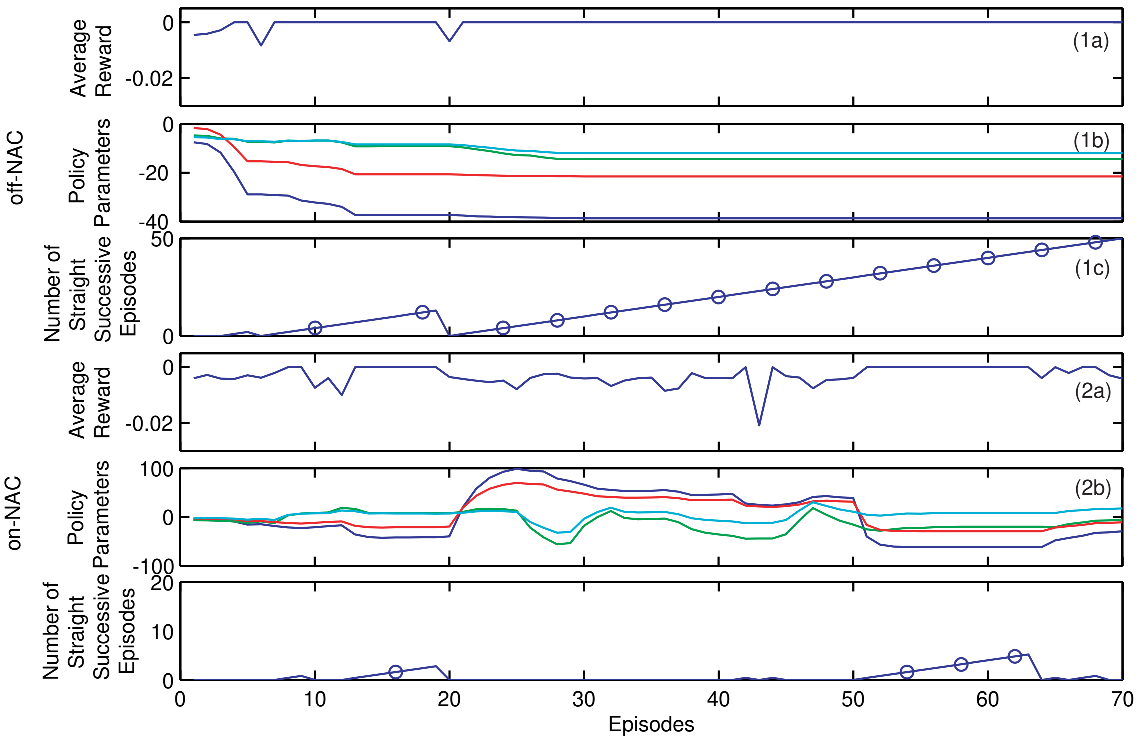


Figure 5: The comparison of learning processes of off-NAC and on-NAC

set at $\sigma_0 = 2$ but reduced by $\sigma = \sigma_0/(1 + n/100)$ as the number of episodes (n) increased. For the trajectory maintenance and the update iteration, we set $K = 10$, $K_0 = 0$, and $M = 0$. In this simulation, we prepared a difficult situation in which the initial policy parameters were located far apart from good ones, as such drawn randomly from the range of $[39, 41]^4$.

Table 2 shows the number of successful learning runs and the average number of episodes until the task was first completed. With respect to both of the criteria, off-NAC$_2$ was superior to the off-NAC$_0$, suggesting the effectiveness of our meta-control of exploration and exploitation.

Table 2: The performance comparison of off-NAC$_0$ and off-NAC$_2$

	Number of successful learning runs	Average length of episodes
off-NAC$_0$	7/10	170.0
off-NAC$_2$	9/10	99.4

Figure 6 shows typical learning curves by the two off-NAC variations. Both processes were stable, but the learning by the off-NAC$_2$ was faster than that by the off-NAC$_0$. This result shows that an appropriate meta-control of exploration and exploitation not only accelerates but also stabilizes the learning.

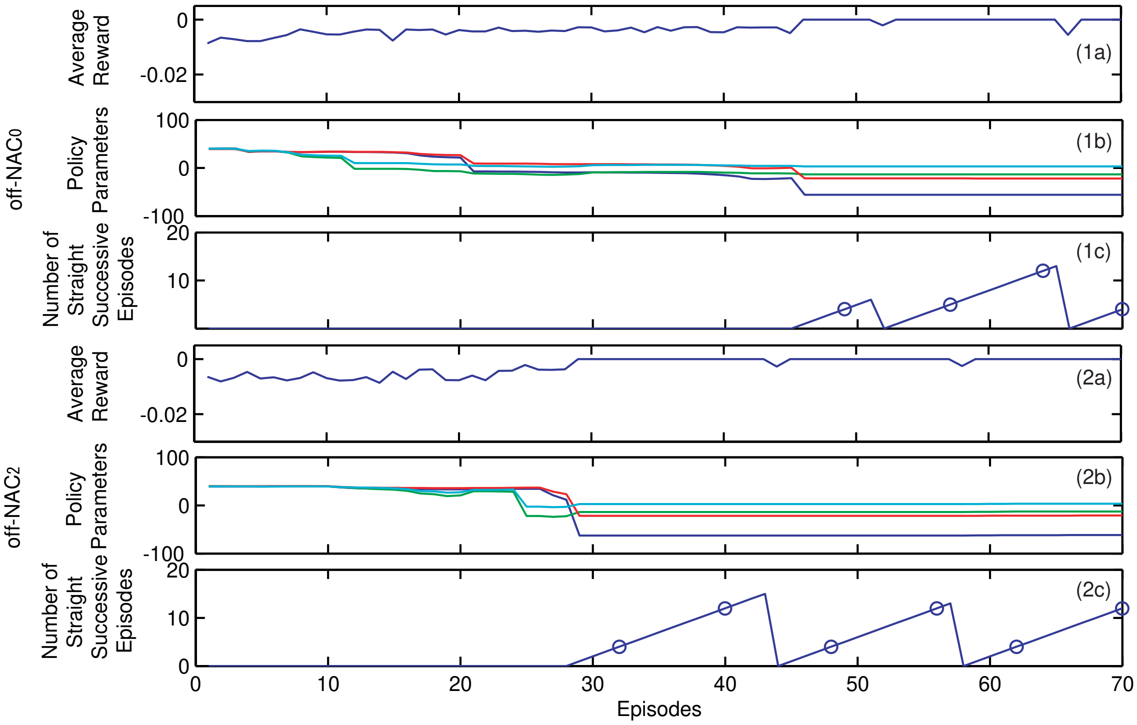


Figure 6: Comparison of learning processes of off-NAC$_0$ and off-NAC$_2$

4.3 Learning of hierarchical off-NAC

In this section, we show computer simulation results to examine the performance of the hierarchical off-NAC. In the simulations so far, some *a priori* knowledge, such as the symmetric nature of motions of the snake-like robot, was used for designing the parameterization of the policy, and then the dimensionality of the policy parameter was as small as four. In this section, we assume a hierarchically-structured parameterization of the policy, but do not design the parameterization in details. The policy was defined as a parametric linear model of basis functions which were arranged on mesh grids in the two-dimensional space of the sensory input $\bar{s} \equiv [d, \nu]'$; the dimensionality of the policy is 32 (see Table 3), and its optimization is much more difficult than that in the previous simulations.

By setting the hight of the binary tree (Fig.1) at $N = 5$, the output of the policy Ψ^0 (equation (22)) is given by

$$\Psi^0 \equiv \sum_{i=1}^{32} \Theta_i \Psi_i^5, \tag{31}$$

where $\Psi_i^5 (i = 1, \ldots, 32 = 2^N)$ are the outputs of the leaf nodes (at the fifth layer) and correspond to the basis functions above. The linear coefficients $\{\Theta_i | i = 1, \ldots, 32\}$ were calculated by equation (24) as

$$\Theta_i \equiv \prod_{m=0}^{4} \theta_{\zeta_m(i)}^{5-m}. \tag{32}$$

The i-th basis function Ψ_i^5 was input by the sensory input $\bar{s} \equiv [d, \nu]'$ and output a normalize Gaussian function: $\Psi_i^5(\bar{s}) = \Upsilon_i(\bar{s}) / \sum_{j=1}^{32} \Upsilon_k(\bar{s})$, where $\Upsilon_i(\bar{s})$ is a Gaussian: $\Upsilon_i(\bar{s}) = \exp(-(d - \mu_{d,i})^2/2\sigma_d^2 - (w - \mu_{\nu,i})^2/2\sigma_\nu^2)$. S.d.'s σ_d and σ_ν were fixed at 0.25 and $\pi/4$, respectively. Table 3 shows the means (centers) of the basis functions.

Table 3: Means of basis functions

		Distance μ_d (m)			
		1/8	3/8	5/8	7/8
Angle μ_ν (rad)	$\pi/8$	Ψ_1^5	Ψ_3^5	Ψ_9^5	Ψ_{11}^5
	$3\pi/8$	Ψ_2^5	Ψ_4^5	Ψ_{10}^5	Ψ_{12}^5
	$5\pi/8$	Ψ_5^5	Ψ_7^5	Ψ_{13}^5	Ψ_{15}^5
	$7\pi/8$	Ψ_6^5	Ψ_8^5	Ψ_{14}^5	Ψ_{16}^5
	$9\pi/8$	Ψ_{17}^5	Ψ_{19}^5	Ψ_{25}^5	Ψ_{27}^5
	$11\pi/8$	Ψ_{18}^5	Ψ_{20}^5	Ψ_{26}^5	Ψ_{28}^5
	$13\pi/8$	Ψ_{21}^5	Ψ_{23}^5	Ψ_{29}^5	Ψ_{31}^5
	$15\pi/8$	Ψ_{22}^5	Ψ_{24}^5	Ψ_{30}^5	Ψ_{32}^5

The basis functions of upper (bottom) four rows in this table, $\Psi_{1:16}^5$ ($\Psi_{17:32}^5$), may be activated when the wall exists on the left-hand-side (right-hand-side), and the output of a node at the first layer, Ψ_1^1 (Ψ_2^1), is the weighted summation of those basis functions $\Psi_{1:16}^5$ ($\Psi_{17:32}^5$) weighted by $\theta^{2:5}$ (equation (23)). A node of an upper layer is thus the abstraction of its children nodes, such to integrate the basis functions on the leaf nodes of the binary sub-tree stemming from the node. For example, the output of the left-most node on the second layer, Ψ_1^2, is given by the weighted sum of upper-left eight basis functions in Table 3, so the activation of this node indicates the wall is on the left-hand-side and located close to the robot.

For approximating the state value function, we employed the same basis functions as those listed in Table 3: $\bar{f}_i = \Psi_i^5 (i = 1, \ldots, 32)$, so that $V^\pi(s) \approx \sum v_i \bar{f}_i$. When n-th layer is selected to be trained (25), the off-policy NAC learning was conducted for the 2^n-dimensional policy which is represented by equation (22). In this case, the learning was performed as if ψ^n in equation (22) are the effective basis functions on this layer, and then the i-th compatible function is calculated as $\nabla_{\theta_i^n} \log \pi(s, a) = (a - p_b)\psi_i^n \cos(\Lambda - 1.5)$.

At the onset of each learning episode, the snake-like robot was set at an initial position at which the head link was set either at $x \in [-0.2, 0.2]$ and $y = 1.13$ or $x = 4.12$ and $y \in [2.8, 3.2]$, and the body axis was set to be perpendicular to the nearest wall's direction, for the first episode or when the robot was stuck in the previous episode. μ_n and σ_H in equation (25) determined the scheduling of releasing the frozen dimensionality and were set at $\{\mu_n\} = \{10, 30, 50, 70, 90\}$ and $\sigma_H = 20$. This setting allowed a gradual learning from the upper layer to the lower layer; when the number of samples was small especially at the early learning stage, the policy parameters at higher layers, whose number is relatively small, were optimized, and when the number of samples increased, the policy parameters at lower layers, whose number is relatively large, were optimized. In the latter case, although the training of a policy with a high dimensionality is difficult and a lot of samples are usually necessary, our off-policy NAC enables us to use the stored samples, instead of gathering large number of new samples after the change of the effective dimensionality of the policy.

Here, we compare the hierarchical off-NAC and the standard (non-hierarchical) off-NAC. To see the effectiveness of the hierarchical setting, the policy of the standard off-NAC is given by

$$\Psi^0 \equiv \sum_{i=1}^{32} \theta_i \Psi_i^5, \tag{33}$$

where Ψ_i^5 are the same basis functions as in the hierarchical one (Table 3). Ten learning runs were performed using different random seeds, by each of the two off-NAC variations. Eligibility λ was set at 0.999, and the exploitation parameter σ was set at 0 (no exploration). The maximum number of stored sample sequences (K) was set at 50, and the number of the initially-stored sample sequences (K_0) was set at 9. In each episode, the policy parameter was updated five times in an off-line manner ($M = 5$). In this experiment, we employed the quasi-active sampling method, i.e., $g_t = 1/z$, for $t = t_{\text{collision}} - 800, \ldots, t_{\text{collision}}$, where $t_{\text{collision}}$ denotes the last time step when the robot had run into the wall, and z is the normalizing term which counts the number of g_t with non-zero values. Each learning run was terminated after 100 learning episodes. The other simulation conditions are the same as those in the previous simulations.

Table 4 shows the number of successful learning runs and the average number of episodes until the robot was first successful in passing through the crank course. Among the ten runs by the standard off-NAC, no actor was successful in allowing the robot to pass through the crank course, because the number of samples was not sufficient for the optimization of the 32-dimensional policy. In eight runs, in contrast, good actors were obtained by our hierarchical off-NAC.

Table 4: The performance of hierarchical off-NAC

Number of successful trials	Average length of episodes
8/10	54.0

Figure 7 shows a typical learning curve by the hierarchical off-NAC. The vertical axes denote the average reward in a single episode (a), the number of episodes elapsed from the last failure episode (b), the 32-dimensional policy parameter $\Theta_{1:32}$ (c), and the policy

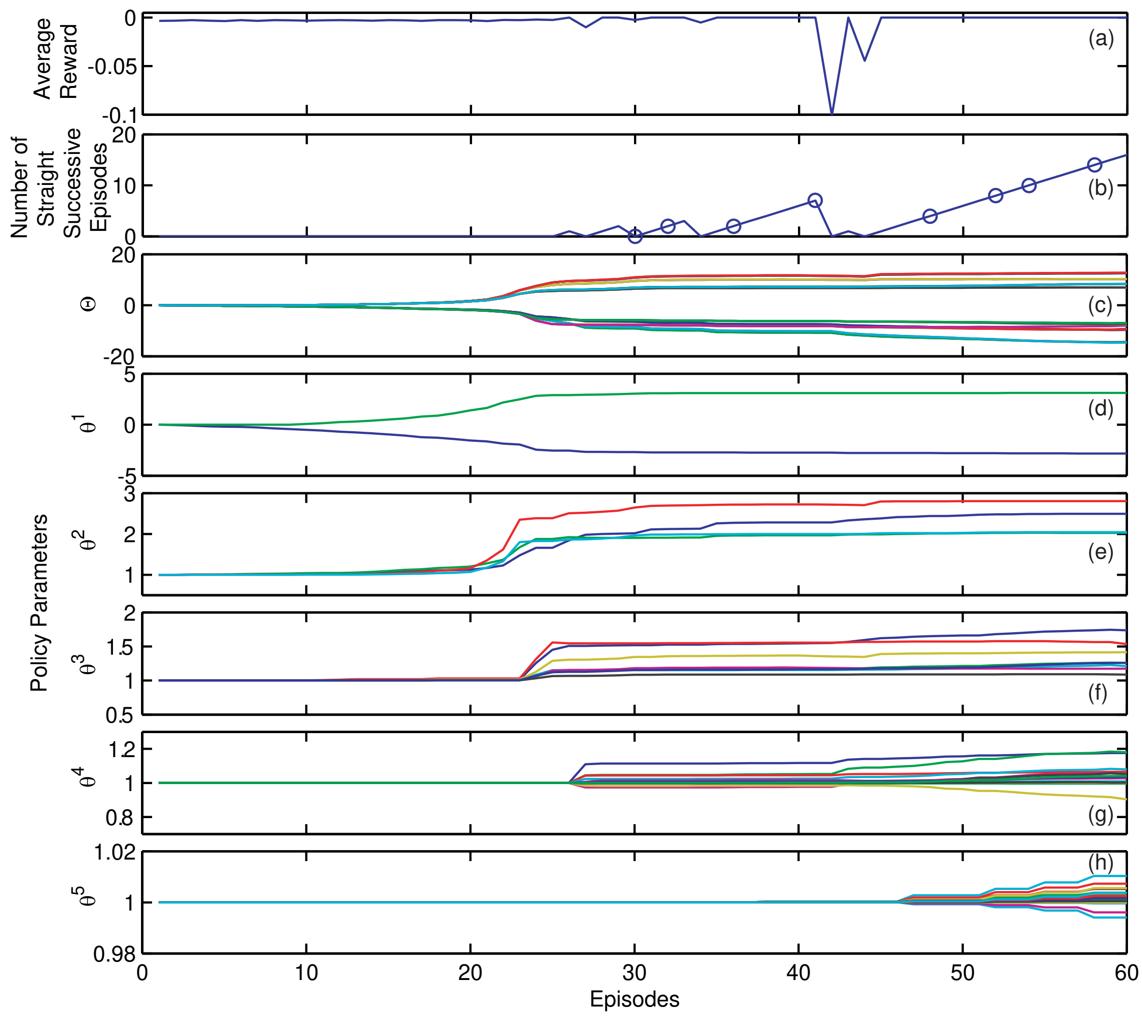


Figure 7: Learning process of hierarchical off-NAC

parameters at the five layers $\theta^{1:5}$ (d)-(h). A circle point indicates that the robot passed through the crank course.

In this learning run, although the average reward temporally dropped off at the 42-th learning episode (panel (a)) possibly by chance, the number of stored trajectories was sufficient, and the temporal degrade did not affect much the policy optimization. The policy parameter at the first layer was optimized at an early learning stage, and converged after about 25 learning episodes, but the average reward were still small and there were no successful episode (panels (a) and (b)). As the average reward and the number of successful episodes increased, the training of policy parameters in the lower layers started, which also improved the whole policy. This simulation result shows that the hierarchical off-NAC is an efficient RL scheme, when the number of policy parameters is large but a hierarchical structure can be assumed in the policy's parameterization. Although the symmetric property of motions of the snake-like robot was not incorporated directly into the policy's parameterization in this experiment, the policy parameter for right turn θ_1^1 and that for left turn θ_2^1 were adjusted with similar efficiency. Because the initial states were set arbitrary in this experiment, a bias due to the initial distribution might be introduced. The quasi-active sampling based on the collision may have worked well in the reduction of the bias.

5 Conclusion

In this study, we proposed two successful off-policy versions of the Natural Actor-Critic algorithms (NAC) [1] [2], i.e., (1) the off-NAC and (2) the hierachical off-NAC. In the standard NAC (on-NAC) learning, past trajectories under the control by past policies cannot be utilized for the training of the current policy. In our off-NAC, on the other hand, the natural policy gradient for the current policy can be estimated without introducing any asymptotic bias by using past trajectories even after the policy improvement, and hence the learning became faster and stabler than the on-NAC learning. In addition, an efficient control of exploration-exploitation can be realized by meta-controlling the generation process of behavior policies, though the meta-control does not affect directly the optimization of the policy. Moreover, we proposed a hierarchical parameterization of the policy and an off-policy NAC learning for that policy, which controls dynamically the effective dimensionality of the policy. Although a policy with a high dimensionality is difficult to train and a good and stable training process usually requires a lot of samples, our off-policy NAC enables us to reuse past trajectories even after the dimensionality change in the policy. Our hierarchical off-NAC, which has a dynamic releasing and freezing mechanism of the controller's dimensionality, exhibits not only an insight into the biological developmental process, but also a new hierarchical reinforcement learning scheme.

When the difference between the target and the behavior policies is large, however, importance sampling becomes less efficient, because long trajectory can yield very small ISW and can inhibit the approximation of target stationary distribution. In our scheme, since samples generated by target or exploratory similar behavior policy are surely employed with dismillar past ones for the gradient estimation, the approximation-failure problem can be partly avoided. In addition, the quasi-active sampling can partly restrain ISW reduction as a whole. But we think that more effective methods beyond the temporal credit assignment are necessary, such as kernel density estimation.

Since the learning processes of policy gradient methods depend on the initial policy parameter, however, it is a remaining work to develop an algorithm in which multiple policies are trained simultaneously and each controller learns from not only its own experiences but also those of others.

Appendix

A Proof of Proposition 1

From equation (14), the proof of

$$E_{b_p}\{\bar{A}_t|s_0\} = E_\pi\{A_t|s_0\},\ E_{b_p}\{\bar{b}_t|s_0\} = E_\pi\{b_t|s_0\} \tag{34}$$

is equivalent to that of equation (17). Since $H_t^\lambda, \bar{H}_t^\lambda, G_t^\lambda, \bar{G}_t^\lambda$ are convex connections of $H_t^{(n)}, \bar{H}_t^{(n)}, G_t^{(n)}, \bar{G}_t^{(n)}$, respectively, weighted by the eligibility trace (equation (11)), the proof of equation (34) is equivalent to that of equations:

$$E_{b_p}\left\{\phi_t(\bar{H}_t^{(n)} + \phi_t)' \prod_{m=0}^{t} \rho_m \middle| s_0\right\} = E_\pi\left\{\phi_t(H_t^{(n)} + \phi_t)' \middle| s_0\right\} \tag{35}$$

$$E_{b_p}\left\{\phi_t \bar{G}_t^{(n)} \prod_{m=0}^{t} \rho_m \middle| s_0\right\} = E_\pi\left\{\phi_t G_t^{(n)} \middle| s_0\right\}. \tag{36}$$

We use the notations:

$$\bar{X}_t^{(n,e,f)} \equiv \sum_{l=f}^{n-e} Z_t^{(l)} \prod_{k=1}^{l-e} \rho_{t+k}, X_t^{(n,e,f)} \equiv \sum_{l=f}^{n-e} Z_t^{(l)},$$

and define the set of states and actions from time step i to time step $j+1$ as

$$\Omega_i^j \equiv \{< s_i, a_i, \ldots, a_j, s_{j+1} > | s_i\},$$

where i is the time at which the state s_i is observed. Here, an arbitrary trajectory included in Ω_i^j is defined as $h =< s_i, a_i, \ldots, a_j, s_{j+1} >$. Then, the following relation holds.

$$\begin{aligned}
&E_{b_p}\{\bar{X}_t^{(n,e,f)} | s_t, a_t, s_{t+1}\} \\
&= \sum_{l=f}^{n-e} \int_{h \in \Omega_{t+1}^{t+l-e}} p_{p_b}(h) Z_t^{(l)} \prod_{k=1}^{l-e} \rho_{t+k} \; dh \\
&= \sum_{l=f}^{n-e} \int_{h \in \Omega_{t+1}^{t+l-e}} \left(\prod_{k=1}^{l-e} p_{s_{t+k+1}} b_{p_{t+k}}\right) Z_t^{(l)} \prod_{k=1}^{l-e} \frac{\pi_{t+k}}{b_{p_{t+k}}} \; dh \\
&= \sum_{l=f}^{n-e} \int_{h \in \Omega_{t+1}^{t+l-e}} \left(\prod_{k=1}^{l-e} p_{s_{t+k+1}} \pi_{t+k}\right) Z_t^{(l)} \; dh \\
&= E_\pi\{X_t^{(n,e,f)} | s_t, a_t, s_{t+1}\}.
\end{aligned}$$

From this relation, the following equation is satisfied for an arbitrary function Y_t:

$$\begin{aligned}
&E_{b_p}\left\{\phi_t(\bar{X}_t^{(n,e,f)} + Y_t)' \prod_{k=0}^{t} \rho_k \middle| s_0\right\} \\
&= \int_{h \in \Omega_0^t} p_{b_p}(h) \phi_t \Big(E_{b_p}\Big\{\bar{X}_t^{(n,e,f)} \Big| s_t, a_t, s_{t+1}\Big\} + Y_t\Big)' \prod_k^t \rho_k \; dh \\
&= \int_{h \in \Omega_0^t} \prod_{k=0}^{t} p_{k+1} b_{p_k} \phi_t \Big(E_\pi\Big\{X_t^{(n,e,f)} \Big| s_t, a_t, s_{t+1}\Big\} + Y_t\Big)' \prod_k^t \frac{\pi_k}{b_{p_k}} \; dh \\
&= \int_{h \in \Omega_0^t} p_\pi(h) \phi_t \Big(E_\pi\Big\{X_t^{(n,e,f)} \Big| s_t, a_t, s_{t+1}\Big\} + Y_t\Big)' \; dh \\
&= E_\pi\left\{\phi_t(X_t^{(n,e,f)} + Y_t)' \middle| s_0\right\}.
\end{aligned}$$

By applying $Z_t^{(l)} = \gamma^l \phi_{t+l}, e = 0, f = n, Y_t = \phi_t$ into the relation above, we obtain equation (35). Similarly, by applying $Z_t^{(l)} = \gamma^l r_{t+l}, e = 1, f = 1, Y_t = 0$, we obtain equation (36). Accordingly, equation (34) is proved.

B Proof of Proposition 2

Let $\{X_k\}_{k=1}^{\infty}$ be a sequence of i.i.d. random variables which satisfy $E\{|X_k|\} < \infty$. Then, $\frac{1}{K}\sum_{k=1}^{K} X_k$ converges to $E\{X_k\}$ with probability 1 according to the strong law of large number.

If $X_k = \bar{A}^k, \bar{b}^k, A^k, b^k$, $E\{|X_k|\} < \infty$ is satisfied for an episodic task. Then, $\frac{1}{K}\bar{A}^k$, $\frac{1}{K}\bar{b}^k$, $\frac{1}{K}A^k$, $\frac{1}{K}b^k$ converge to $E_{b_p}\{\bar{A}|s_0\}, E_{b_p}\{\bar{b}|s_0\}, E_\pi\{A|s_0\}, E_\pi\{b|s_0\}$, respectively, with probability 1. In addition, from Proposition 1 (equation (17)), $\bar{A}^k, \bar{b}^k$ converge to A^k, b^k, respectively, with probability 1. Moreover, from

$$w_K^\pi \equiv \left(\frac{1}{K}\sum_{k=1}^{K} A^k\right)^{-1}\left(\frac{1}{K}\sum_{k=1}^{K} b^k\right),\ \bar{w}_K^\pi \equiv \left(\frac{1}{K}\sum_{k=1}^{K} \bar{A}^k\right)^{-1}\left(\frac{1}{K}\sum_{k=1}^{K} \bar{b}^k\right),$$

$\bar{w}_K^\pi$ converges to w_K^π with probability 1.

C Transformation of matrix A and vector b

The eligibility vector $\tilde{z}_t$ of the off-policy LSTD$Q(\lambda)$ (Algorithm 1) is described by

$$\tilde{z}_t = \sum_{k=0}^{t} c_k \phi_k (\gamma\lambda)^{t-k} \prod_{j=k+1}^{t} \rho_j.$$

Here, we obtain the relation:

$$\begin{aligned}
&\sum_{t=0}^{T-1} \bar{z}_t X_t \\
&= \sum_{t=0}^{T-1} \left(\sum_{k=0}^{t} c_k \phi_k (\gamma\lambda)^{t-k} \prod_{j=k+1}^{t} \rho_j\right) X_t \\
&= \sum_{t=0}^{T-1} c_t \phi_t \sum_{k=t}^{T-1} (\gamma\lambda)^{k-t} X_k \prod_{j=t+1}^{k} \rho_j \\
&= \sum_{t=0}^{T-1} \phi_t \left(\sum_{k=t}^{T-1} (\gamma\lambda)^{k-t} X_k \prod_{j=t+1}^{k} \rho_j\right) \sum_{k=0}^{t} g_k \prod_{m=k}^{t} \rho_m \\
&\approx \sum_{t=0}^{T-1} \phi_t \left(\sum_{k=t}^{\infty} (\gamma\lambda)^{k-t} X_t \prod_{j=k+1}^{t} \rho_j\right) \sum_{k=0}^{t} g_t \prod_{m=k}^{t} \rho_m.
\end{aligned}$$

By substituting X_t by $(\phi_t - \gamma\rho_{t+1}\phi_{t+1})'$,

$$= \sum_{t=0}^{T-1} \phi_t (\bar{H}_t^\lambda + \phi_t) \sum_{k=0}^{t} g_t \prod_{m=k}^{t} \rho_m = \sum_{t=0}^{T-1} \tilde{A}_t,$$

is obtained [3]. And, by substituting X_t by r_{t+1},

$$= \sum_{t=0}^{T-1} \phi_t(\bar{G}_t^\lambda) \sum_{k=0}^{t} g_t \prod_{m=k}^{t} \rho_m = \sum_{t=0}^{T-1} \tilde{b}_t$$

is obtained. By adding the conditions, $c_k = 1, \rho_k = 1, \forall k$ and $g_0 = 1, g_j = 0, j > 0$, a similar relation for the on-policy LSTD$Q(\lambda)$ can be obtained.

D Sequential calculation of inverse matrix

In Algorithm 1, the Shannon-Morison equation:

$$(C + xy')^{-1} = C^{-1} - \frac{C^{-1}xy'C^{-1}}{1 + y'C^{-1}x}, \tag{37}$$

[25] is useful for the sequential calculation of the inverse matrix of A. The cost of this calculation is in $O(q^2)$, which is in a smaller order than that of Algorithm 1 ($O(q^3)$), so this modification has a merit when we want to calculate w^π frequently. In this case, line 10 (Algorithm 1) is modified into

$$\begin{aligned} B_i^{-1} &= (B_{i-1} + z_t(\phi_t^k - \gamma\rho_{t+1}\phi_{t+1}^k)')^{-1} \\ &= B_{i-1}^{-1} - \frac{B_{i-1}^{-1} z_t (\phi_t^k - \gamma\rho_{t+1}\phi_{t+1}^k)' B_{i-1}^{-1}}{1 + (\phi_t^k - \gamma\rho_{t+1}\phi_{t+1}^k)' B_{i-1}^{-1} z_t}, \end{aligned}$$

by using the notations: $B_i \equiv \sum_{m=0}^{t} A_m^k + \sum_{m=1}^{k-1} A^m, B_{-1} \equiv 0, i \equiv t + \sum_{j=1}^{k-1} T^j$. Since B_t^{-1} is the inverse of the summation of matrix A until time t, w_i^π is obtained as

$$w_i^\pi = B_i^{-1} \left\{ \sum_{i=0}^{t} b_i + \sum_{m=1}^{k-1} b^m \right\}.$$

References

[1] J. Peters, S. Vijayakumar and S. Schaal: "Reinforcement learning for humanoid robotics", Humanoids2003, Third IEEE-RAS International Conference on Humanoid Robots (2003).

[2] J. Peters, S. Vijayakumar and S. Schaal: "Natural actor-critic", Proceedings of the 16th European Conference on Machine Learning, pp. 280–291 (2005).

[3] R. Sutton and A. Barto: "Reinforcement Learning: An Introduction", MIT Press (1998).

[4] D. P. Bertsekas and J. N. Tsitsiklis: "Neuro-Dynamic Programming", Athena Scientific (1996).

[5] L. C. Baird: "Residual algorithms: Reinforcement learning with function approximation", Proceedings of the Twelfth International Conference on Machine Learning, pp. 30–37 (1995).

[6] R. Williams: "Simple statistical gradient following algorithms for connectionist reinforcement learning", Machine Learning, **8**, pp. 279–292 (1992).

[7] V. R. Konda and J. N. Tsitsiklis: "Actor-critic algorithms", Advances in Neural Information Processing Systems, Vol. 12, pp. 1008–1014 (2000).

[8] R. S. Sutton, D. McAllester, S. Singh and Y. Mansour: "Policy gradient methods for reinforcement learning with function approximation", Advances in Neural Information Processing Systems, Vol. 12, pp. 1057–1063 (2000).

[9] S. A. Kakade: "A natural policy gradient", Advances in Neural Information Processing Systems, Vol. 14 (2002).

[10] S. Bradtke and A. Barto: "Linear least-squares algorithms for temporal difference learning", Machine Learning, **22**, (1996).

[11] J. A. Boyan: "Technical update: Least-squares temporal difference learning", Machine Learning, **49**, (2002).

[12] M. G. Lagoudakis and R. Parr: "Least-squares policy iteration", Journal of Machine Learning Research, **4**, pp. 1107–1149 (2003).

[13] Y. Nakamura, T. Mori and S. Ishii: "Natural policy gradient reinforcement learning for a CPG control of a biped robot", International Conference on Parallel Problem Solving from Nature, pp. 972–981 (2004).

[14] S. B. Thrun: "The role of exploration in learning control with neural networks", Handbook of intelligent control: neural, fuzzy and adaptive approaches (Eds. by White, D. A. and Sofge, D. A.), Kentucky, Van Nostrand Reinhold (1992).

[15] S. Ishii, W. Yoshida and J. Yoshimoto: "Control of exploitation-exploration metaparameter in reinforcement learning", Neural Networks, **15**, pp. 665–687 (2002).

[16] C. R. Shelton: "Importance Sampling for Reinforcement Learning with Multiple Objectives", PhD thesis, Massachusetts Institute of Technology (2001).

[17] N. Meuleau, L. Peshkin and K.-E. Kim: "Exploration in gradient-based reinforcement learning", Technical report, AI Memo, MIT (2001-003).

[18] D. Precup, R. S. Sutton and S. Dasgupta: "Off-policy temporal-difference learning with function approximation", Proceedings of the Eighteenth International Conference on Machine Learning, pp. 417–424 (2001).

[19] L. C. Baird: "Advantage updating", Technical report wl-tr-93-1146, Wright Patterson AFB OH (1993).

[20] D. Precup, R. S. Sutton and S. Singh: "Eligibility traces for off-policy policy evaluation", Proceedings of the Seventeenth International Conference on Machine Learning, pp. 759–766 (2000).

[21] A. Nedic and D. P. Bertsekas: "Least-squares policy evaluation algorithms with linear function approximation", Discrete Event Systems: Theory and Applications, **13**, pp. 79–110 (2003).

[22] S. Thrun: "Efficient exploration in reinforcement learning", Technical report, Carnegie Mellon University (1992).

[23] G. Taga: "Freezing and freeing degrees of freedom in a model neuro-musculo-skeletal system for development of locomotion", Proc XVIth Int Soc Biomechanics Congress, pp. 47–47 (1997).

[24] T. Mori, Y. Nakamura, M. Sato and S. Ishii: "Reinforcement learning for a CPG-driven biped robot", Ninteenth National Conference on Artificial Intelligence, pp. 623–630 (2004).

[25] G. H. Golub and C. F. V. Loan: "Matrix Computations, 3rd edition", Johns Hopkins University Press, Baltimore, MD (1996).