Robotics and Autonomous Systems 112 (2019) 72–83

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation

Yoshihisa Tsurumine [a,*], Yunduan Cui [a], Eiji Uchibe [b], Takamitsu Matsubara [a]

[a] Graduate School of Science and Technology, Division of Information Science, Nara Institute of Science and Technology, 8916-5 Takayamacho, Ikoma, Nara, Japan

[b] Department of Brain Robot Interface, ATR Computational Neuroscience Laboratories, 2-2-2 Hikaridai, Seikacho, Soraku-gun, Kyoto, Japan

HIGHLIGHTS

- Learning of cloth manipulation by Deep Reinforcement Learning by a dual-arm robot.
- Combine smooth policy update with feature extraction in deep neural networks.
- Propose new Deep Reinforcement Learning based on Dynamic Policy Programming.
- Achieved better sample efficiency than comparisons by smooth policy update.

ARTICLE INFO

Article history:
Available online 19 November 2018

MSC:
00-01
99-00

Keywords:
Deep reinforcement learning
Robotic cloth manipulation
Dynamic policy programming

ABSTRACT

Deep Reinforcement Learning (DRL), which can learn complex policies with high-dimensional observations as inputs, e.g., images, has been successfully applied to various tasks. Therefore, it may be suitable to apply them for robots to learn and perform daily activities like washing and folding clothes, cooking, and cleaning since such tasks are difficult for non-DRL methods that often require either (1) direct access to state variables or (2) well-designed hand-engineered features extracted from sensory inputs. However, applying DRL to real robots remains very challenging because conventional DRL algorithms require a huge number of training samples for learning, which is arduous in real robots. To alleviate this dilemma, in this paper, we propose two sample efficient DRL algorithms: Deep P-Network (DPN) and Dueling Deep P-Network (DDPN). The core idea is to combine the nature of smooth policy update with the capability of automatic feature extraction in deep neural networks to enhance the sample efficiency and learning stability with fewer samples. The proposed methods were first investigated by a robot-arm reaching task in the simulation that compared previous DRL methods and applied to two real robotic cloth manipulation tasks: (1) flipping a handkerchief and (2) folding a t-shirt with a limited number of samples. All the results suggest that our method outperformed the previous DRL methods.

© 2018 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the capability of learning optimal policies by interacting with an unknown environment, Reinforcement Learning (RL) [1] has been applied to a broad range of platforms in robot control, such as autonomous helicopter/vehicle [2,3], a robot dog [4], and humanoid robots [5–7]. Most of the RL algorithms in the above studies require either (1) direct access to state variables or (2) well-designed hand-engineered features extracted from sensory inputs. However, they become difficult in general when considering more complex and practical tasks/situations. For example, in household robots, such as humans' daily activities as washing and folding clothes, cooking and cleaning are desirable to be learned and performed by RL, but it is not easy to achieve either (1) or (2) (e.g., [8]).

The recent advance of Deep Neural Networks (DNNs) [9] enables automatic extraction of high-level features to outperform traditional hand-engineered features extracted from high-dimensional observations as input like raw images [10–12] and audio signals [13,14]. Deep Reinforcement Learning (DRL), e.g., Deep Q-Network (DQN) [15] and Trust Region Policy Optimization (TRPO) [16], have been proposed by exploiting such DNN capabilities for automatic feature extraction in RL. By automatically abstracting good high-level features from raw images, DQN can learn a complex policy with human-level performances on various Atari video games. On the other hand, the application of DQN-like algorithms to real robot control problems remains limited due to insufficient samples. To learn suitable features, DRL generally

* Corresponding author.
E-mail addresses: tsurumine.yoshihisa.tm6@is.naist.jp (Y. Tsurumine), cuiyunduan@gmail.com (Y. Cui), uchibe@atr.jp (E. Uchibe), takam-m@is.naist.jp (T. Matsubara).

https://doi.org/10.1016/j.robot.2018.11.004
0921-8890/© 2018 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

requires a huge number of training samples; unfortunately, generating these samples in a real robot system is arduous because of the high sampling cost. Such a dilemma between the difficulty of feature extraction and high sampling cost is one of the most crucial issues for making the RL approach more practical in real robot control.

In this paper, we propose two sample efficient DRL algorithms: Deep P-Network (DPN) and Dueling Deep P-Network (DDPN). The core idea shared by these algorithms is to combine the nature of smooth policy update in value function-based reinforcement learning with the automatic feature extraction from high-dimensional observations in deep neural networks to enhance the sample efficiency and the learning stability with fewer samples. The smoothness of policy update is promoted by limiting the relative entropy or the Kullback–Leibler divergence between the current and new policies in the learning process. Even though several RL algorithms with such smooth policy update have been proposed [17,18], we focus on Dynamic Policy Programming (DPP) [19] for the following reasons: (1) its asymptotic convergence nature to the optimal policy (for discrete state–action cases); (2) a discrete action space that can easily use high-level actions; and (3) success in high-dimensional robot control tasks with direct access to state variables [20].

We first present DPN as a novel deep reinforcement learning based on DPP. It automatically abstracts the features from raw images by exploiting the nature of smooth policy update by introducing the Kullback–Leibler divergence between current and new policies as a regularization term into the reward function for better sample efficiency. Then an extension of DPN with a particularly suitable network structure of DNN, Dueling DPN, is proposed for better generalization capability inspired by the dueling network structure for DQN [21]. DPN and Dueling DPN are first applied to a n DOF simulated manipulator reaching task to evaluate their learning performances and compare the effect of different network structures and parameter settings with previous DRL methods. Then Dueling DPN was applied to real robotic cloth manipulation tasks to control a dual-arm humanoid robot NEXTAGE (Fig. 1a) to learn (1) the flipping of a handkerchief (Fig. 1b) and (2) folding a t-shirt (Fig. 1c) with a limited number of samples. We chose robotic cloth manipulation because it requires both a complicated and a high-dimensional state definition and a huge number of training samples to recognize and model the flexible cloth or to learn a suitable manipulation policy. Although several studies have been conducted in similar tasks [22,23], their approaches were based on learning from demonstration scheme, where a huge number of manually generated samples and a relatively long training period are required without self-improvement in the learning loop. To the best of our knowledge, this is the first successful application of deep reinforcement learning for robotic cloth manipulation with a small number of demonstration data for initializing the networks and raw images as observations. Our methods employ a discrete action space for grasping and dropping points on cloth following a previous work [22] so that the RL agent learns high-level and general policies that are independent on a robot platform.

Our preliminary work was published as a conference paper [24]. This article extends our preliminary work as follows:

1. provides an initialization scheme of DPNs with (non-expert) demonstration data for accelerating learning;
2. validates the effectiveness of our methods by thorough simulations;
3. investigates the effect of the degree of promotion of the smoothness in policy update into reward functions;
4. further analyzes the average Bellman error in value function approximations to demonstrate the superiority of the proposed algorithm;
5. conducts real robot experiments on a t-shirt folding task with complex action space and reward functions.

The remainder of this paper is organized as follows. Section 2 introduces previous research on deep reinforcement learning and robotic cloth manipulation are introduced. The preparations are introduced in Section 3. The details of the proposed method are explained in Section 4. Sections 5 and 6 present the experimental results in simulations and real robot experiments on cloth manipulation tasks, respectively. Discussion and conclusion are described in Sections 7 and 8, respectively.

2. Related work

2.1. Deep reinforcement learning

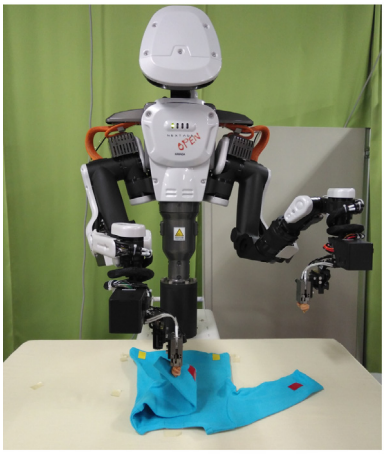
DRL methods, which are very popular solutions to deal with high-dimensional observations in RL, apply DNNs to automatically extract features from high-dimensional observations. As the basics of a value function based DRL that learns with a global state–action value function and discrete action space, DQN [15] is the first successful integration of deep learning with Q-learning. As further improvements, double deep Q-networks [25] and dueling architecture DQN [21] were proposed to increase robustness against the overestimating value function and easier convergence, respectively. The policy search based DRL, e.g., [16], focuses on utilizing DNN to locally learn a policy that maximizes the total reward during the task. DRL can learn complex policies with human-level performances in both simulation tasks and video games through trial and error. On the other hand, DRL applications to real robot control problems remain challenging due to the requirement of a huge number of training samples generated by operating robots to be interactive with the environment. One solution is to simultaneously obtain many samples from multiple sampling robots [26] or simulations [27]. Another solution is to achieve better sample efficiency during learning, e.g., by setting suitable initial policies [28].

2.2. RL with smooth policy update

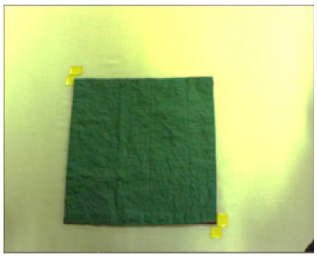
To improve the sample efficiency and learning stability with fewer samples in RL, smooth policy update is exploited to limit the information that is lost during learning [29]. The main idea is to introduce the Kullback–Leibler divergence to limit the differences between the current and new policies into the reward function. The related approaches include both value function-based, e.g., [19], and policy search, e.g., relative entropy policy search [17] and guided policy search [30]. In the robot control domain, the smooth policy update was applied to learn hierarchical policies [31] and achieve sample efficiency and learning stability with kernel trick in robot hand control with a 32-dimensional state space [20]. The current combination of smooth policy update and DRL [32] focuses on learning end-to-end motor policies represented by linear Gaussian controllers in continuous action space. On the other hand, combining the value function based DRL with smooth policy update has not been intensively studied.

2.3. Robotic cloth manipulation

Studies in robotic cloth manipulation can be divided into two directions: task-oriented and knowledge-based approaches [22]. The former focuses on cloth recognition based on manually selected features, depth sensor and cloth model, e.g., the hem [33], the corners and wrinkles [34,35], the geometric cloth polygon [36], the 2.5D depth sensor point cloud [37], a 3D range camera [38], and models and simulations [39–41] for subsequent manipulation. One complete pipeline of autonomously folding clothes based on vision features is described in [42]. The latter learns the relationship



(a) NEXTAGE: dual arm humanoid robot



(b) Handkerchief

(c) T-shirt

Fig. 1. Real robot setting: Our targets are two robotic cloth manipulation tasks with a dual-arm humanoid robot NEXTAGE (a) (1) flipping of a handkerchief (b) and (2) folding a t-shirt (c) with a limited number of samples.

between the robot manipulation and the clothing shape. Matsubara et al. [43] proposed an RL approach to learn motor skills to handle a t-shirt based on the topological relationship between the robot configuration and the non-rigid material. Another friction model based RL approach was proposed to search robot motion trajectories to place a scarf around the mannequin's neck in [44]. Lee et al. [45] presented force-based demonstration learning for deformable object manipulation. The convolutional neural network is applied to classify different types of clothes and control a robot to handle the clothes' pose by selecting grasping points in [46]. Other recent works [22,23] employed a deep convolutional autoencoder to learn high-level features from cloth images and implicitly generated a cloth model with manipulation. Corona et al. [47] propose an algorithm using DNN detect grasping points that bring the garment to a known pose. These works demonstrated the potential of DNN in robotic cloth manipulation, even though many samples and a relatively long training time were required.

3. Preparation

3.1. Reinforcement learning

RL [1,48] solves the Markov decision process (MDP) defined by a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ is a finite set of states. $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$ is a finite set of actions. $\mathcal{T}^a_{ss'}$ is the probability of transitioning from state s to state s' under action a. The corresponding reward is defined as $r^a_{ss'}$ with reward function $\mathcal{R}$. $\gamma \in (0, 1)$ is the discount parameter. Policy $\pi(a|s)$ represents the probability of action a being taken under state s. The value function is defined as the expected discounted total reward in state s:

$$V(s) = \mathbb{E}_{\pi,\mathcal{T}}\left[\sum_{t=0}^{\infty} \gamma^t r_{s_t} \middle| s_0 = s\right], \tag{1}$$

where $r_{s_t} = \sum_{\substack{a\in\mathcal{A}\\ s'\in\mathcal{S}}} \pi(a|s_t)\mathcal{T}^a_{s_t s'} r^a_{s_t s'}$ is the expected reward from state s_t.

The objective of RL is to find optimal policy π^* that maximizes the value function to satisfy the following Bellman equation:

$$V^*(s) = \max_{\pi} \sum_{\substack{a\in\mathcal{A}\\ s'\in\mathcal{S}}} \pi(a|s)\mathcal{T}^a_{ss'}\left(r^a_{ss'} + \gamma V^*(s')\right), \tag{2}$$

or a Q function for state–action pairs (s, a):

$$Q^*(s, a) = \max_{\pi} \sum_{s'\in\mathcal{S}} \mathcal{T}^a_{ss'}\left(r^a_{ss'} + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s')Q^*(s', a')\right). \tag{3}$$

Value function based RL algorithms, e.g., Q-learning [49], SARSA [50], and LSPI [51], approximate the value/Q function using the Temporal Difference (TD) error. For example, the TD update rule in Q-learning follows $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r^{a_t}_{s_t s_{t+1}} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$, where α is the learning rate.

3.2. Deep Q-network

As a combination of Q-learning and DNN, DQN [15] successfully approximates the Q function by DNN. Since the direct approximation of a dynamically changing Q function by DNN is difficult, DQN stabilizes the learning by several tricks, like target network, error clip, and experience replay. When Q function approximated by DNN parameter $\boldsymbol{\theta}$ is represented by $\hat{Q}(s, a; \boldsymbol{\theta})$, a target network is defined as $\hat{Q}(s, a; \boldsymbol{\theta}^-)$. $\boldsymbol{\theta}^-$ is updated every C steps following $\boldsymbol{\theta}^- = \boldsymbol{\theta}$, and $\boldsymbol{\theta}$ is updated every step with sample $(s_j, a_j, r^{a_j}_{s_j s_{j+1}}, s_{j+1})$ from a global memory that stores all the generated samples by performing a gradient descent with the TD error:

$$J(\boldsymbol{\theta}, \boldsymbol{\theta}^-) \triangleq \sum_{(s_j, a_j, r^{a_j}_{s_j s_{j+1}}, s_{j+1})\in\mathcal{D}} \left(r^{a_j}_{s_j s_{j+1}} + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \boldsymbol{\theta}^-) - \hat{Q}(s_j, a_j; \boldsymbol{\theta})\right)^2, \tag{4}$$

where $\mathcal{D}$ denotes the experience replay buffer. The gradient descent step on $\boldsymbol{\theta}$ needs to be sufficiently small to make the learning slow and reduce the sample efficiency to avoid excessively changing the target function in the function approximation with DNN. One serious concern of DQN is that the smoothness in the policy update is not explicitly considered during learning. Such a lack of smoothness can drastically deteriorate the learning performance when the new policy is radically different from the previous one. In the subsequent section, we give a short summary of Dynamic Policy Programming [52,19], which is a value function based RL algorithm that employs a smooth policy update.

3.3. Dynamic policy programming

To exploit the nature of smooth policy update, DPP [52,19] considers the Kullback–Leibler divergence between current policy π and baseline policy $\bar{\pi}$ into the reward function to minimize the difference between the current and baseline policy while maximizing the expected reward:

$$D_{\mathrm{KL}} = \sum_{a\in\mathcal{A}} \pi(a|s) \log \frac{\pi(a|s)}{\bar{\pi}(a|s)}. \tag{5}$$

Thus, the Bellman optimality equation Eq. (2) is modified as:

$$V^*_{\bar{\pi}}(s) = \max_{\pi} \sum_{\substack{a\in\mathcal{A} \\ s'\in\mathcal{S}}} \pi(a|s) \left[\mathcal{T}^a_{ss'} \left(r^a_{ss'} + \gamma V^*_{\bar{\pi}}(s') \right) - \frac{1}{\eta} \log\left(\frac{\pi(a|s)}{\bar{\pi}(a|s)} \right) \right]. \tag{6}$$

The effect of the Kullback–Leibler divergence is controlled by inverse temperature η. Following [52,53], we let η be a positive constant. Optimal value function $V^*_{\bar{\pi}}(s)$ for all $s \in \mathcal{S}$ and Optimal policy $\bar{\pi}^*(a|s)$ for all (s, a) satisfy double-loop fixed-point iterations as follows:

$$V^{t+1}_{\bar{\pi}}(s) = \frac{1}{\eta} \log \sum_{a\in\mathcal{A}} \bar{\pi}^t(a|s) \exp\left[\eta \sum_{s'\in\mathcal{S}} \mathcal{T}^a_{ss'} \left(r^a_{ss'} + \gamma V^t_{\bar{\pi}}(s') \right) \right] \tag{7}$$

$$\bar{\pi}^{t+1}(a|s) = \frac{\bar{\pi}^t(a|s) \exp\left[\eta \sum_{s'\in\mathcal{S}} \mathcal{T}^a_{ss'} \left(r^a_{ss'} + \gamma V^t_{\bar{\pi}}(s') \right) \right]}{\exp\left(\eta V^{t+1}_{\bar{\pi}}(s) \right)}. \tag{8}$$

Action preferences function [1] at the $(t + 1)$-iteration for all state–action pairs (s, a) is defined following [19] to obtain the optimal policy that maximizes the above value function:

$$P_{t+1}(s, a) = \frac{1}{\eta} \log \bar{\pi}^t(a|s) + \sum_{s'\in\mathcal{S}} \mathcal{T}^a_{ss'} \left(r^a_{ss'} + \gamma V^t_{\bar{\pi}}(s') \right). \tag{9}$$

Combining Eq. (9) with Eqs. (7) and (8), a simple form is obtained:

$$V^t_{\bar{\pi}}(s) = \frac{1}{\eta} \log \sum_{a\in\mathcal{A}} \exp\left(\eta P_t(s, a) \right) \tag{10}$$

$$\bar{\pi}^t(a|s) = \frac{\exp\left(\eta P_t(s, a) \right)}{\sum_{a'\in\mathcal{A}} \exp\left(\eta P_t(s, a') \right)}. \tag{11}$$

The update rule of action preference function $P_{t+1}(s, a) = \mathcal{O}P_t(s, a)$ is derived by plugging Eqs. (10) and (11) into Eq. (9):

$$\mathcal{O}P_t(s, a) = P_t(s, a) - \mathcal{L}_\eta P_t(s) + \sum_{s'\in\mathcal{S}} \mathcal{T}^a_{ss'} \left(r^a_{ss'} + \gamma \mathcal{L}_\eta P_t(s') \right), \tag{12}$$

where $\mathcal{L}_\eta P(s) \triangleq \frac{1}{\eta} \log \sum_{a\in\mathcal{A}} \exp(\eta P(s, a)) = V_{\bar{\pi}}(s)$. The difference between $P_{t+1}(s, a)$ and $\mathcal{O}P_t(s, a)$ is used to calculate the error signal to train the action preference function.

The original DPP is only applicable to problems with a modest number of discrete states and prior knowledge about the underlying model. Sampling-based Approximate Dynamic Policy Programming (SADPP) [52] extends it to model-free learning with large-scale (continuous) states. For N training samples, $[s_n, a_n]_{n=1:N}$, SADPP approximates $P(s, a)$ by Linear Function Approximation (LFA): $\hat{P}(s_n, a_n; \boldsymbol{\theta}) = \boldsymbol{\phi}(\boldsymbol{x}_n)^{\mathrm{T}}\boldsymbol{\theta}$, where $\boldsymbol{\phi}(\boldsymbol{x}_n)$ denotes the output vector of the basis functions and $\boldsymbol{\theta}$ is the corresponding weight vector. The weight vector is updated by minimizing empirical loss function $J(\boldsymbol{\theta}; \hat{\boldsymbol{P}}) \triangleq \|\boldsymbol{\Phi}\boldsymbol{\theta} - \mathcal{O}\hat{\boldsymbol{P}}\|^2_2$, where $\mathcal{O}\hat{\boldsymbol{P}}$ is an $N \times 1$ matrix with elements $\mathcal{O}\hat{P}(s, a; \boldsymbol{\theta})$ following Eq. (12), where $\mathcal{L}_\eta P(s)$ is translated into a Boltzmann softmax operator for more analytically tractable recursion.

Although such an extension to DPP can be applied to toy problems as mountain-car control [52], its scalability is still limited due to the exponentially growing size of the basis functions with increasing input dimensionality and corresponding intractability. More scalable function approximators, such as non-parametric regression, have been employed and successfully applied for real robot control tasks [54,20]. However, their applications to such very high-dimensional and redundant state like sensor data and raw image data remain infeasible.

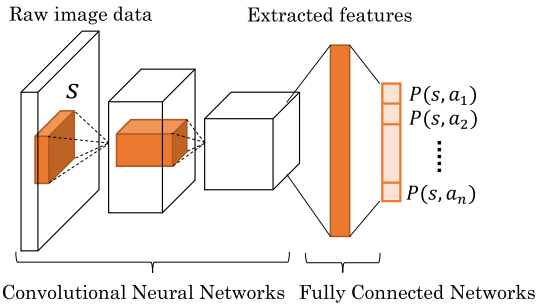


Fig. 2. Network architectures of Deep P-Network.

4. Proposed method

In this section, we first present a novel deep reinforcement learning algorithm, Deep P-Network (DPN), that exploits the advantages of both DRL for high-dimensional state space and DPP for smooth policy update. Next we consider a more suitable neural network architecture for DPN as inspired by the Dueling DQN [21] that has a new neural network architecture with two parts to automatically produce separate estimates of value function $V(s)$ and advantage function $A(s, a)$ that fulfills $A(s, a) = Q(s, a) - V(s)$ without any extra supervision. Finally, we present how we can initialize both DPN and Dueling DPN with demonstration data for accelerating learning.

4.1. Deep P-network

In this subsection, we propose DPN, which approximates the action preferences function $P(s, a; \boldsymbol{\theta})$ in Eq. (9) by DNN. Its network structure is defined in Fig. 2. Initial input state s is a raw RGB/grayscale image that usually has very high dimensionality. A Convolutional Neural Network (CNN) abstracts the raw image to a lower-dimensional high-level feature set. These features are in turn processed by a Fully Connected Network (FCN) and the final layer has m nodes, where m is the number of actions in $\mathcal{A}$ and the ith node's output is approximated value $\hat{P}(s, a_i; \boldsymbol{\theta})$.

The training algorithm for DPN is given by Algorithm 1. In DPN, I is the number of iterations of DPN, and each iteration has M episodes with $M \times T$ samples. Local memory $\mathcal{D}$ is maintained to store the current E iteration samples for experience feedback. The updating of networks is operated in every episode. The current parameters as $\boldsymbol{\theta}^-$ are saved to build target network $\hat{P}(s, a; \boldsymbol{\theta}^-)$. The update is divided into N sub-problems. In each one, the agent repeatedly collects mini-batches of samples $(s_j, a_j, r^{a_j}_{s_j s_{j+1}}, s_{j+1})$ from $\mathcal{D}$ and calculates teaching signal y_j following Eq. (12):

$$y_j(\boldsymbol{\theta}^-) = \hat{P}(s_j, a_j; \boldsymbol{\theta}^-) - \mathcal{L}_\eta \hat{P}(s_j; \boldsymbol{\theta}^-) + r^{a_j}_{s_j s_{j+1}} + \gamma \mathcal{L}_\eta \hat{P}(s_{j+1}; \boldsymbol{\theta}^-). \tag{13}$$

The network parameters are updated by applying gradient descent algorithms to minimize the loss function:

$$J(\boldsymbol{\theta}, \boldsymbol{\theta}^-) \triangleq (y_j(\boldsymbol{\theta}^-) - \hat{P}(s_j, a_j; \boldsymbol{\theta}))^2. \tag{14}$$

Algorithm 1: Deep P-Network

Initialize local memory $\mathcal{D}$ and its size E
Initialize network weights $\boldsymbol{\theta}$
Initialize target network weights $\boldsymbol{\theta}^- = \boldsymbol{\theta}$
Function `UpdatePNetwork(`$\boldsymbol{\theta}, \boldsymbol{\theta}^-, \mathcal{D}, N$`)`**:**
  Initialize target network update number N
  Initialize epoch number U
  Initialize number of mini-batches $B = E/$(minibatch size)
  for $n = 1, 2, \ldots, N$ **do**
    for $i = 1, 2, \ldots, U$ **do**
      Shuffle local memory $\mathcal{D}$ index
      for $j = 1, 2, \ldots, B$ **do**
        Sample minibatch of transition $(s_j, a_j, r^{a_j}_{s_j s_{j+1}}, s_{j+1})$ in $\mathcal{D}$
        Calculate the teaching signal:
        $y_j = \hat{P}(s_j, a_j; \boldsymbol{\theta}^-) - \mathcal{L}_\eta \hat{P}(s_j; \boldsymbol{\theta}^-) + r^{a_j}_{s_j s_{j+1}} + \gamma \mathcal{L}_\eta \hat{P}(s_{j+1}; \boldsymbol{\theta}^-)$
        Get *loss* and update $\boldsymbol{\theta}$ by performing a gradient descent step on $(y_j - \hat{P}(s_j, a_j; \boldsymbol{\theta}))^2$
    Update target network $\boldsymbol{\theta}^- = \boldsymbol{\theta}$
Initialize DPP parameters I, M, T
for $i = 1, 2, \ldots, I$ **do**
  for *episode* $= 1, 2, \ldots, M$ **do**
    for $t = 1, 2, \ldots, T$ **do**
      Take action a_t with softmax policy based on $\bar{\pi}^t(a_t|s_t)$ following $\hat{P}(s_t, a_t; \boldsymbol{\theta})$ and Eq. (11)
      Receive new state s_{t+1} and reward $r^{a_t}_{s_t s_{t+1}}$
      Store transition $(s_t, a_t, r^{a_t}_{s_t s_{t+1}}, s_{t+1})$ in $\mathcal{D}$
  `UpdatePNetwork(`$\boldsymbol{\theta}, \boldsymbol{\theta}^-, \mathcal{D}, N$`)`
  if $i > (E-1)$ **then**
    Update $\mathcal{D}$ to store the current $(E-1)$ iterations' samples

The loss of the gradient descent is added to the average loss. When the average loss is less than a threshold, the current sub-update is terminated and the target networks are updated after processing N mini-batch by $\boldsymbol{\theta}^- = \boldsymbol{\theta}$. In summary, the following are the main differences between DQN and DPN:

1. Local memory $\mathcal{D}$ with the current E iteration samples is applied to store fewer samples and focuses more on new samples. As the value function is updated using only the data sampled from the new policy of updating, the improvement of the policy is fast.
2. Update $\boldsymbol{\theta}$ every episode with T steps rather than every step. The DPN updates in the same process as the DPP updating the value function.
3. Parameter $\boldsymbol{\theta}$ of DNN is updated with the number of epochs U using memory $\mathcal{D}$. Switch the target network and update $\boldsymbol{\theta}$ N times. By switching the target network while updating the value function, update the iteration of the value function without new samples.

4.2. Dueling network architecture for DPN

In this subsection, Dueling DPN (DDPN) is proposed as a natural extension of DPN toward a dueling network [21].

Plugging $V_{\bar{\pi}}(s) = \mathcal{L}_\eta P(s) \triangleq \frac{1}{\eta} \log \sum_{a \in \mathcal{A}} \exp(\eta P(s, a))$ into Eq. (12), we obtain:

$$P_{t+1}(s, a) = P_t(s, a) - V^t_{\bar{\pi}}(s) + \sum_{s' \in \mathcal{S}} \mathcal{T}^a_{ss'} \left(r^a_{ss'} + \gamma V^t_{\bar{\pi}}(s) \right). \tag{15}$$

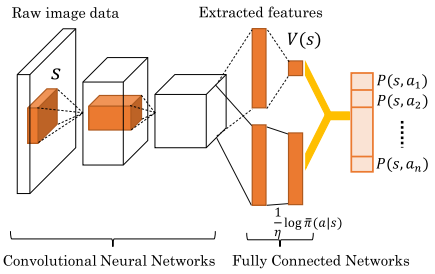


Fig. 3. Network architecture of Dueling Deep P-Network.

Combining it with Eq. (9), the action preference function can be represented as:

$$P_t(s, a) = \frac{1}{\eta} \log \bar{\pi}^t(a|s) + V^t_{\bar{\pi}}(s), \tag{16}$$

You can directly derive Eq. (16) from Eqs. (10) and (11) because the denominator of Eq. (11) is given by $\exp(\eta V^t_{\bar{\pi}}(s))$. which can be naturally divided into two parts: value function $V^t_{\bar{\pi}}(s)$ and advantage function $A(s, a) \triangleq \frac{1}{\eta} \log \bar{\pi}^t(a|s)$. Fig. 3 shows the architecture of the Dueling DPN that consists of two streams of $V(s)$ and $A(s, a)$ while sharing one convolutional feature abstraction module. One regularization term is added to Eq. (16) to fulfill $\sum_{a' \in \mathcal{A}} \exp\left(\log \bar{\pi}_t(a|s)\right) = 1$:

$$P_t(s, a) = \frac{1}{\eta} \log \bar{\pi}_t(a|s) + V^t_{\bar{\pi}}(s) - \frac{1}{\|\mathcal{A}\|} \times \left\{ \left(\sum_{a' \in \mathcal{A}} \exp\left(\log \bar{\pi}_t(a|s)\right) \right) - 1 \right\}. \tag{17}$$

4.3. Prior policy initialization of DPN/Dueling DPN

Based on previous works [55,28], DRL can successfully learn tasks with a small number of samples by initializing its policy from demonstrations. DPN and Dueling DPN are expected to fully exploit the successfully initialized policies based on prior knowledge since they employ a smooth policy update in the DRL framework. Even though the given initial policies may not be perfect, they should be beneficial for accelerating reinforcement learning.

We show an initialization procedure for both DPN and Dueling DPN using a small number of demonstration samples (generated by a human operator), summarized in Algorithm 2. By following Algorithm 2, we store these samples in local memory $\mathcal{D}$ and update the P-network before learning the results in the networks including prior knowledge. Parameter K controls the effect of the demonstration data on learning. The nature of the smooth policy update enables the DPN/Dueling DPN to smoothly update the resulting policies from the initialized ones.

5. Simulation

5.1. n DOF manipulator reaching task

In this section, we investigated the learning performance of DPN and Dueling DPN in a simulated n DOF manipulator reaching task ($n = 5, 15, 25, 50$) by comparing DQN and Dueling DQN. The state is the entire grayscale 84×84 px image where the n DOF manipulator is drawn, as shown in Fig. 4. The length of each limb between the adjoining joints is set to $\frac{1}{n}$. The environment and the DPN parameter settings are respectively shown in Table 1a and b. For the network architecture, the input layer has $84 \times 84 \times 1$ nodes for each pixel of the state image. The setting of the middle layer

Algorithm 2: Initialization and learning of DPN policy

Initialize local memory $\mathcal{D}$ and its size E
Initialize network weights $\boldsymbol{\theta}$
Initialize target network weights $\boldsymbol{\theta}^- = \boldsymbol{\theta}$
Load L points initial sample and store in local memory $\mathcal{D}$
Initialize number of initial updates K
`UpdatePNetwork(`$\boldsymbol{\theta}, \boldsymbol{\theta}^-, \mathcal{D}, K$`)` # Initialization DPN policy
Initialize DPP parameters I, M, T
for $i = 1, 2, ..., I$ **do**
  for *episode* $= 1, 2, ..., M$ **do**
    for $t = 1, 2, ..., T$ **do**
      Take action a_t with softmax policy based on $\bar{\pi}^t(a_t|s_t)$ following $\hat{P}(s_t, a_t; \boldsymbol{\theta})$ and Eq. (11)
      Receive new state s_{t+1} and reward $r^{a_t}_{s_t s_{t+1}}$
      Store transition $(s_t, a_t, r^{a_t}_{s_t s_{t+1}}, s_{t+1})$ in $\mathcal{D}$
  `UpdatePNetwork(`$\boldsymbol{\theta}, \boldsymbol{\theta}^-, \mathcal{D}, N$`)`
  if $i > (E-1)$ **then**
    Update $\mathcal{D}$ to store the current $(E-1)$ iteration samples

Table 1
Settings and learning parameters of n DOF manipulator reaching task.

(a) Parameter setting of n DOF manipulator reaching task

MDP setting	Description
State	The entirety of a grayscale 84×84 px image.
Action	Discrete actions $[-0.0875, -0.0175, 0, 0.0175, 0.0875]$ (rad) to increment the joint with the respective angle for each DOF. We define an action at each time step as one move per joint to reduce the number of actions to $(N \times 5)$.
Reward	Reward function is set as $r = -\left(\lvert X_{\text{target}} - X\rvert + \lvert Y_{\text{target}} - Y\rvert\right)$ where X, Y is the current position of the manipulator's end-effector, and $X_{\text{target}} = 0.6830$, $Y_{\text{target}} = 0$ is the target position.
Initial state	The first joint is set to position [0, 0]. All angles are initialized to 0 rad at the start of the simulation.

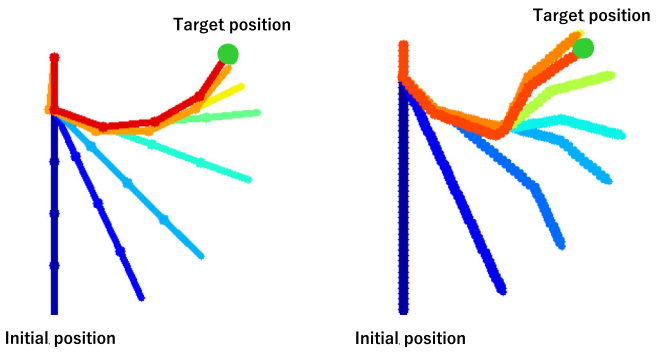
(b) Parameter setting of DPN algorithm

Parameter	Meaning	Value
η	Parameters controlling the effect of smooth policy update	1
M	Number of episodes for one iteration	5
T	Number of steps for one episode	30
E	Size of memory $\mathcal{D}$ to store sample	450
N	Number of target network updates in one iteration	2
U	Epoch number of DNN updates	80
L	Number of samples for policy initialization	300
K	Number of P-network updates at policy initialization	20

and the optimizer follows previous works [15,21]. The policies of DPN and Dueling DPN are calculated by Eq. (11) while DQN uses a ε-greedy policy. All the results are derived in five repetitions. Our hardware platform is a PC with an Intel Core i7-5960 CPU, a Nvidia GTX 1080 GPU, and 64 GB memory. The software platform was built by Tensorflow [56] and Keras [57].

5.2. Learning results

The learning results with different numbers of DOFs are shown in Fig. 5. The left side shows the results without the initialization procedure with demonstration samples. Both DPN and Dueling DPN clearly performed as well as expected in the simulations: they stably improved the performance with only around 2000 samples, but DQN and Dueling DQN could not. By separately learning the action preferences and value functions, Dueling DPN outperformed DPN. The right side of Fig. 5 shows the result with prior policy initialization (marked by "init", where the number of demonstration samples is defined as L in Table 1b, and in this task, we used 300 samples generated by a non-optimal policy). Compared with other algorithms, Dueling DPN supported the value of initialization more and quickly outperformed the performance of the demonstration samples (purple dashed line). All these results show the sample efficiency of DPN/Dueling DPN. They used fewer samples to achieve higher average reward values than the DQN algorithms under the same DNN setting, and their superiority rose with the DOF increase.



(a) 5-DOF manipulater (b) 50-DOF manipulater

Fig. 4. Successful behaviors for n (5 and 50) DOF manipulator reaching tasks. Different colored robots represent manipulator states in different steps.

5.3. Effect of parameter η in DPN and Dueling DPN

We investigated the effect of DPN's parameter η in a 5-DOF manipulator reaching task. With an increase of η, the Kullback–Leibler divergence term in Eq. (6) limits the policy update less. Since the operator $\mathcal{L}_\eta$ is the log-sum-exp function, it is considered a soft-max operator and it converge to the max operator as $\eta \to \infty$. Therefore, the choice of η determines the smoothness of the operator. In addition, DPP converges to the optimal policy for any η, but it changes the rate of convergence significantly. Both DPN and Dueling DPN were tested in five repetitions where $\eta = [0.01, 0.1, 0.5, 1.5, 3.0, 10, 20]$. Fig. 6a shows the results; in a suitable range, i.e., $[0.01, 0.3]$, a larger η resulted in faster learning due to less smoothness in the policy update. On the other hand, an extremely large η caused divergent learning due to the numerical instabilities using the exponential function in the action preferences function [20]. Dueling DPN also has better stability with various η than DPN, maybe because its architecture divides the action preferences function into two parts. This may contribute to Dueling DPN's better learning capability shown in Section 5.2.

5.4. Bellman error in DPN

Next we investigated the effect of smooth policy update in DPN's function approximation. We define the Bellman error as:

$$BE(\hat{V}^{t+1}(s), r^a_{ss'}, \hat{V}^t(s)) = \|\hat{V}^{t+1}(s) - (r^a_{ss'} + \gamma \hat{V}^t(s))\|. \tag{18}$$

The Bellman error measures the approximation error of the value function during learning. State value function $\hat{V}^t$ of DPN can be calculated following Eq. (10), and the calculation in DQN follows $\hat{V}^t(s) = \mathbb{E}_\pi\left[Q^t(s,a)\right]$. Fig. 7 shows the average Bellman error in the first 15 iterations of ten learnings in 5-DOF manipulation reaching tasks. As a result, we achieved more accurate value function approximation in DPN and Dueling DPN than in DQN and Dueling DQN due to limiting the overly large updates that result in stable and efficient learning.

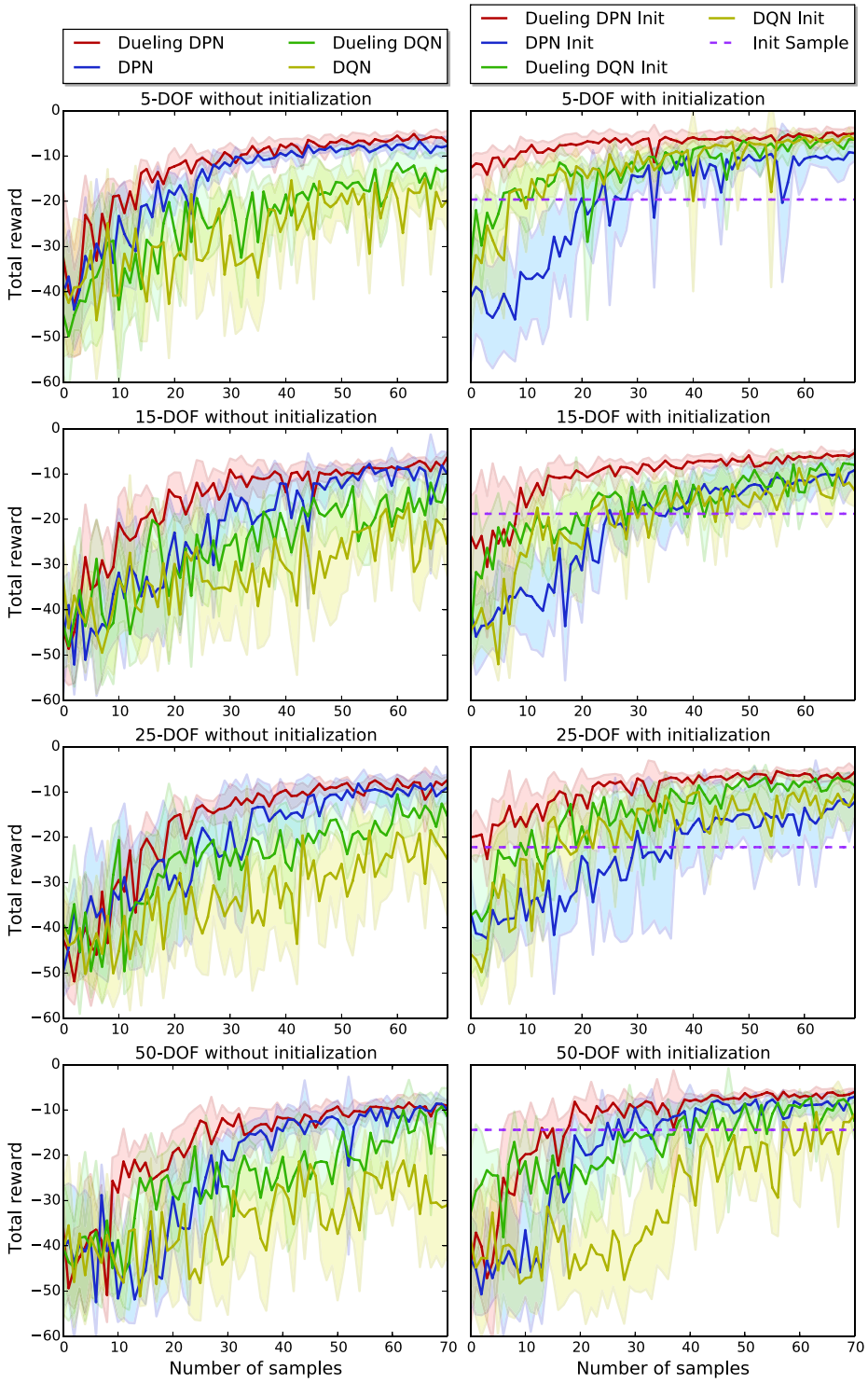


Fig. 5. Learning curve of n DOF manipulator reaching task.

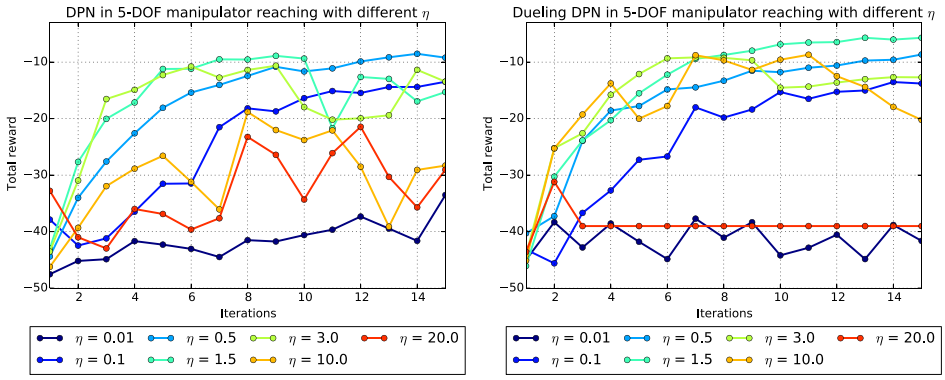
6. Real robot experiment

In this section, we applied the Dueling DPN to the NEXTAGE robot, a 15-DOF humanoid robot with sufficient precision for manufacturing, to learn two robotic cloth manipulation tasks. Following [22], we focus on learning a policy with high-level discrete actions, i.e., the grasp and release points on a cloth to solve two tasks: (1) flipping over a handkerchief and (2) folding a t-shirt.

6.1. Flipping a handkerchief

6.1.1. Setting

The environment and the DPN settings of this task are respectively shown in Table 2a and b. The network architecture, the optimizer, and the hardware/software mainly follow the setting in Section 5.1, and the input layer is fixed to $84 \times 84 \times 3$ nodes for the RGB state image. The software is built on the Robot Operating System (ROS) [58].



(a) DPN (b) Dueling DPN

Fig. 6. Average learning results of DPN/Dueling DPN with different values of η in the 5-DOF manipulator reaching task over ten experiments.

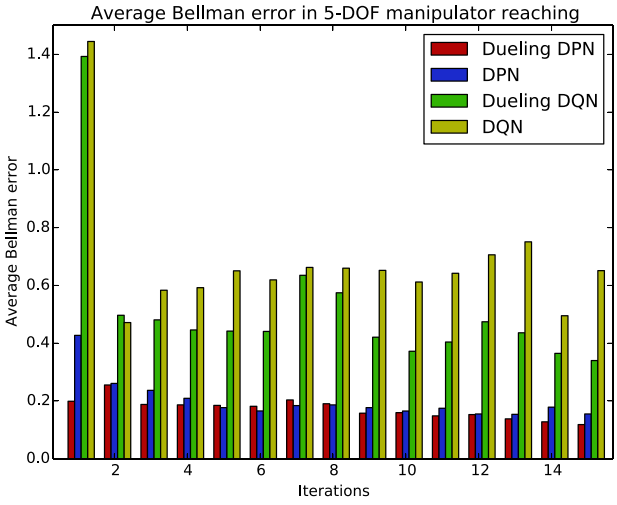
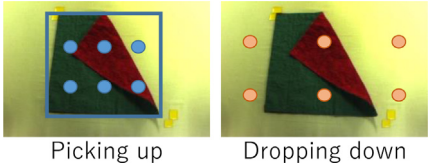
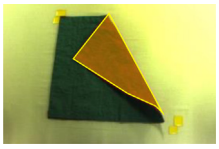


Fig. 7. Average Bellman error in each iteration.

Table 2
Settings and learning parameters of flipping handkerchief task.

(a) Parameter setting of flipping handkerchief task

MDP setting	Description
State	The input state is a 84 × 84 px RGB image from NEXTAGE's integrated camera.
Action	6 × 6 = 36 gripper actions are defined as picking up the handkerchief from 2 × 3 points over its current area and dropping it down to 2 × 3 points over the table.
	
Reward	The reward is defined as the ratio of the red area over the whole image in the current state.
	
Initial state	The handkerchief is initially placed green side up by a human.

(b) Parameter setting of DPN algorithm

Parameter	Meaning	Value
η	Parameters controlling the effect of smooth policy update	1
M	Number of episodes for one iteration	5
T	Number of steps for one episode	30
E	Size of memory $\mathcal{D}$ to store samples	450
N	Number of target network updates in one iteration	2
U	Epoch number of DNN updates	40
L	Number of samples for policy initialization	300
K	Number of P-network updates at policy initialization	20

6.1.2. Results

The learning results of three experiments are shown in Fig. 9. Each experiment took about four hours, including 40 min for manually initializing the handkerchief (≈ 30 seconds per episode) to generate 2400 samples (= 16 iterations). The samples used for the Dueling DPN initialization were generated by a human operator who selected the actions for several states. Note that these samples are collected from non-expert demonstrations and they are insufficient to learn a good policy. The performance of the policy learned from these samples using supervised learning is shown as "Supervised" in Fig. 9. It is better than the random action but cannot be as good as the proposed methods. From the results, without prior policy initialization, Dueling DPN converged faster and achieved higher reward than Dueling DQN. With only 300 demonstration samples for initialization, Dueling DPN learned better policies by exploring with 2400 additional samples. Both Dueling DPN with/without prior policy initialization outperformed the corresponding Dueling DQN algorithms. Fig. 8 shows one example of a handkerchief flipping process learned by Dueling DPN, which turned about 80% of the handkerchief over in about 15 steps.

6.2. Folding a t-shirt

6.2.1. Setting

The next task was to fold a t-shirt. The details of the environment and the algorithm settings are respectively summarized in Table 3a and b. The network architecture, the optimizer, and the hardware/software settings are the same as in the handkerchief flipping task. It is a more challenging task than flipping handkerchief due to (1) a larger action space to deal with various clothing operations, (2) a more complex reward function that shows the stepwise folding achievement degree as in Algorithm 3, (3) fewer steps in one rollout, (4) fewer samples (80) generated by a human operator for prior policy initialization.

6.2.2. Results

The averaged learning results based on three experiments are shown in Fig. 11. Dueling DPN successfully learned the task with only 80 demonstration samples for policy initialization and 112 additional samples generated during reinforcement learning. The

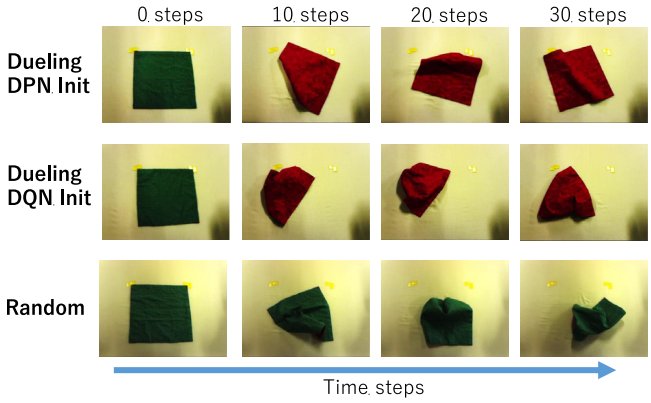


Fig. 8. Handkerchief folding trajectory generated from policy learned from 2400 samples.

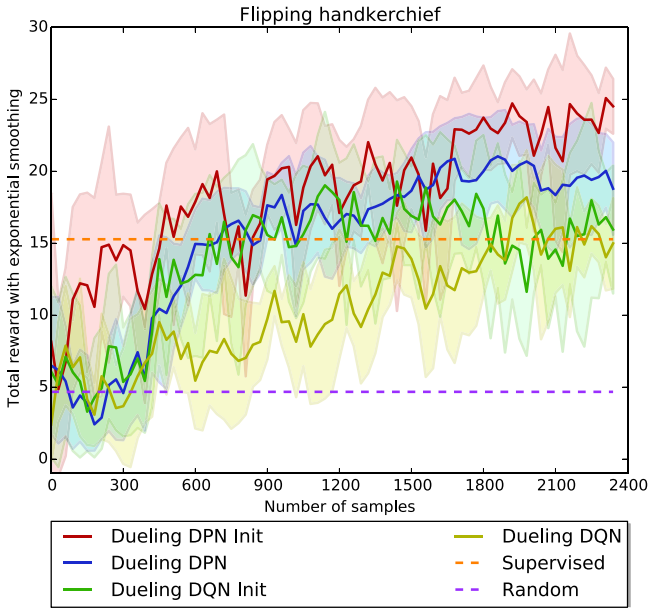


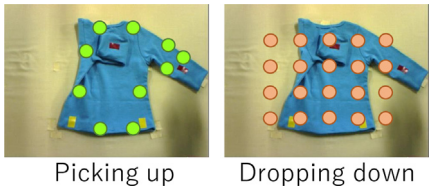
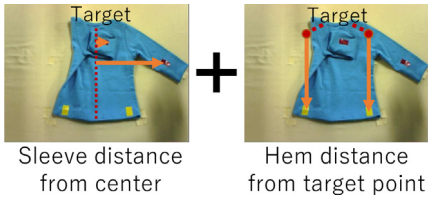
Fig. 9. Learning curve of flipping a handkerchief.

samples for initialization are collected from non-expert demonstrations. The line "Supervised" in Fig. 11 shows that the policy learned by these samples using supervised learning could not lead to a good policy. According to Fig. 11, only Dueling DPN was able to improve its performance based on the given non-expert demonstration while other methods gradually improved their performance but never learned sufficient policies for folding the t-shirt. These results suggest that only our proposed method, Dueling DPN, has the capability to learn tasks with a large action space and a complex reward function even with very limited samples. Fig. 10 shows one t-shirt folding procedure learned by Dueling DPN and DQN with prior policy initialization. Dueling DPN init successfully folded it by appropriately selecting three actions per step, but the corresponding Dueling DQN could only achieve the first step.

Several examples of high-level features learned by Dueling DPN are visualized by Grad-CAM [59] in Fig. 12. These heat maps where the red/blue colors indicate high/low attention of the agent indicate that our proposed method successfully learned useful and meaningful features. The t-shirt's sleeves drew the agent's attention following the order of operations in the first two steps. Then the hem's corner was concerned more to finish the folding task.

Table 3
Settings and learning parameters of folding t-shirt task.

(a) Parameter setting of folding t-shirt task

MDP setting	Description
State	The input state is a 84 × 84 px RGB image from the NEXTAGE's integrated camera.
Action	10 × 20 = 200 gripper actions are defined as picking up the t-shirt from 10 points over its current area and dropping it down to 5 × 4 points over the table.
	
Reward	The reward function is designed to trigger an action to fold the hem after folding the sleeve. The processing is shown in Algorithm 3.
	
Initial state	T-shirt is initialized to a state in which it is spread by a human.

(b) Parameter setting of DPN algorithm

Parameter	Meaning	Value
η	Parameters controlling the effect of smooth policy update	1
M	Number of episodes for one iteration	5
T	Number of steps for one episode	8
E	Size of memory $\mathcal{D}$ to store samples	120
N	Number of target network updates in one iteration	2
U	Epoch number of DNN updates	30
L	Number of samples for policy initialization	80
K	Number of P-network updates at policy initialization	5

Algorithm 3: Reward function of t-shirt folding task

Initialize *InitHemR* = [0.675, 0.8], *InitHemL* = [0.325, 0.8]
Initialize *TargetHemR* = [0.675, 0.208],
TargetHemL = [0.325, 0.208]
Function `HemReward`(*SleevePoint, CenterHem*)**:**
  Initialize reward = 0
  reward = -Sum(|*SleevePoint - CenterHem*|)
  return reward
Function `SleeveReward`(*HemPoint, InitHem, TargetHem*)**:**
  Initialize reward = 0
  Initialize *Distance* = |*InitHem* − *TargetHem*|
  reward = Sum(*Distance* − |*HemPoint* − *TargetHem*|)
  return reward
Function `ShirtReward()`**:**
  Initialize reward = 0
  Update color marker
  Get *HemPointR*, *HemPointL*, *SleevePointR*, *SleevePointL*
  if *Detect hem marker* **then**
    CenterHem = (*HemPointR* + *HemPointL*)/2
    reward = `SleeveReward`(*SleevePointR, CenterHem*) +
    `SleeveReward`(*SleevePointL, CenterHem*)
  else
    reward = 1
  if *Detect sleeve marker* **then**
    reward = reward +
    `HemReward`(*HemPointR, InitHemR, TargetHemR*) +
    `HemReward`(*HemPointL, InitHemL, TargetHemL*)
  return reward

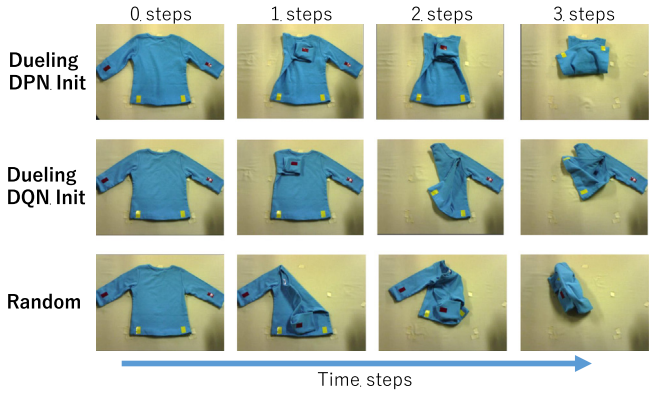


Fig. 10. T-shirt folding trajectory generated from policy learned from 2400 samples.

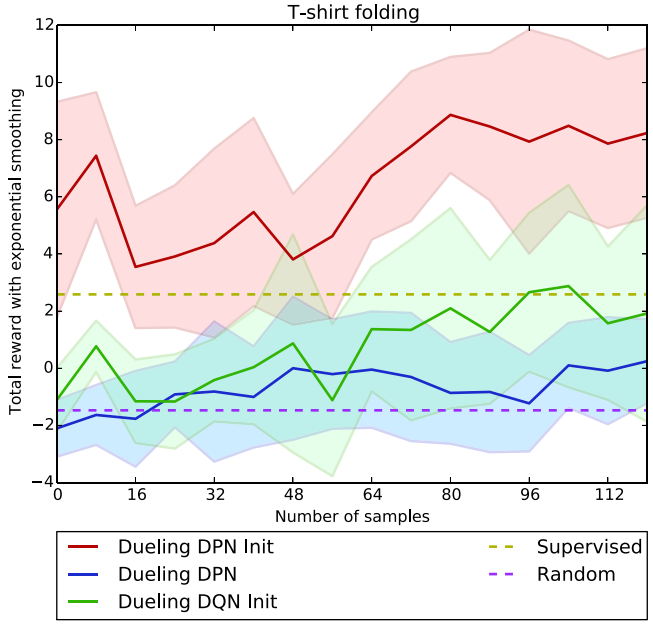


Fig. 11. Learning curve of t-shirt folding task.

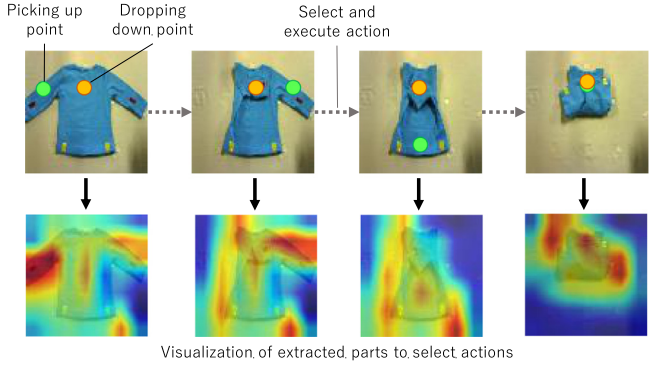


Fig. 12. Visualization of extracted parts in images for action selection using Grad-CAM. Heat map shows parts extracted when actions are selected . (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

7. Discussion

As shown in the experimental results, our proposed methods can learn complex policies with high-dimensional observations such as raw images as input. By promoting a smooth policy update, our methods achieved more sample efficient and stable learning than previous methods with fewer samples.

One future work is to design compact and efficient action space for more complicated deformable object manipulation tasks. In our current work, the action space was set by all the combinations of picking and dropping points allocated on the plane in a grid manner which is not compact or sufficient for more complex tasks. Several previous works could be applied to improve the actions' dexterity like defining meaningful predefined patterns [60], and exploiting synergies [61]. Furthermore, cooperating motion trajectories between two or more robot arms should be considered as actions for challenging tasks like wrapping clothes in [62]. It is also interesting to design an action space to efficiently manipulate the clothes like human beings based on several related works: [34] detected the wrinkle condition and operate robot to extend necessary wrinkle for better folding performance; [35] gripped the cloth edge to reduce the wrinkles caused during operation; [36] directly folded the hem and sleeve of clothes based on a clothing model.

Since the main objective of our experiments is to investigate the sample efficiency of our DRL methods and their effectiveness for robotic cloth manipulation tasks, we alleviated the difficulties in reward function design by using a two-toned handkerchief and color markers that allow us to intuitively design color-based reward functions. Although such a reward function design is not related to the features extracted from states by DRL and therefore has no conflict to the merit of DRL [63]. Designing suitable reward functions for more complex cloth manipulation tasks is another future direction for our work. By applying human feedback [64], it is possible to learn the optimum strategy without designing and implementing the reward function ad hoc. Furthermore, several approaches [65,66] have been proposed to learn the reward function from limited human knowledge. If the expert demonstration is available, it is possible to apply the inverse reinforcement learning [67] to learn a good reward function.

In our robot experiments, initialization required the most time and labor: returning a cloth to its initial state in every iteration. Recent work suggests a potentially helpful approach to alleviate this issue by simultaneously learning a rest policy as a usual policy [68]. Extending our methods by combining them with previous work [68] is also interesting future work.

8. Conclusion

The contribution of this paper is twofold. We proposed two new deep reinforcement learning algorithms, Deep P-Network (DPN) and Dueling Deep P-Network (DDPN). The core idea shared by them is to combine the nature of smooth policy update in value function based reinforcement learning (Dynamic Policy Programming) with the capability of automatic feature extraction in deep neural networks to enhance the sample efficiency and the stability of the learning process with fewer samples. To investigate the performance of our proposed methods, we compared them with previous DRL methods in a simulated n DOF manipulator reaching task. Furthermore, we applied them to two robotic cloth manipulation tasks with a dual arm robot, NEXTAGE: (1) flipping of a handkerchief and (2) folding a t-shirt with a limited number of samples. We confirmed in all the experiments that our method achieved more sample efficiency and stabilized learning than the previous methods.

Acknowledgment

We gratefully acknowledge the support from the New Energy and Industrial Technology Development Organization (NEDO), Japan for this research.

References

[1] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT press Cambridge, 1998.

[2] A. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, E. Liang, Autonomous inverted helicopter flight via reinforcement learning, in: International Symposium on Experimental Robotics (ISER), 2006, pp. 363–372.

[3] T. Hester, M. Quinlan, P. Stone, RTMBA: A real-time model-based reinforcement learning architecture for robot control, in: IEEE International Conference on Robotics and Automation (ICRA), 2012, pp. 85–90.

[4] E. Theodorou, J. Buchli, S. Schaal, Reinforcement learning of motor skills in high dimensions: A path integral approach, in: IEEE International Conference on Robotics and Automation (ICRA), 2010, pp. 2397–2403.

[5] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, G. Cheng, Learning CPG-based biped locomotion with a policy gradient method: Application to a humanoid robot, Int. J. Robot. Res. 27 (2) (2008) 213–228.

[6] S. Bitzer, M. Howard, S. Vijayakumar, Using dimensionality reduction to exploit constraints in reinforcement learning, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2010, pp. 3219–3225.

[7] H. Durrant-Whyte, N. Roy, P. Abbeel, Learning to control a low-cost manipulator using data-efficient reinforcement learning, in: Robotics: Science and Systems (RSS), 2012, pp. 57–64.

[8] K. Yamazaki, R. Ueda, S. Nozawa, M. Kojima, K. Okada, K. Matsumoto, M. Ishikawa, I. Shimoyama, M. Inaba, Home-assistant robot for an aging society, Proc. IEEE 100 (8) (2012) 2429–2441.

[9] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

[10] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems (NIPS), 2012, pp. 1097–1105.

[11] V. Mnih, Machine Learning for Aerial Image Labeling, (Ph.D. thesis), University of Toronto, 2013.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9.

[13] G.E. Dahl, D. Yu, L. Deng, A. Acero, Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition, IEEE Trans. Audio Speech Lang. Process. 20 (1) (2012) 30–42.

[14] A. Graves, A. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013, pp. 6645–6649.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.

[16] J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in: International Conference on Machine Learning (ICML), Vol. 37, 2015, pp. 1889–1897.

[17] J. Peters, K. Mülling, Y. Altun, Relative entropy policy search, in: Association of the Advancement of Artificial Intelligence (AAAI), 2010, pp. 1607–1612.

[18] S. Levine, N. Wagoner, P. Abbeel, Learning contact-rich manipulation skills with guided policy search, in: IEEE Conference on Robotics and Automation (ICRA), 2015.

[19] M.G. Azar, V. Gómez, H.J. Kappen, Dynamic policy programming, J. Mach. Learn. Res. 13 (1) (2012) 3207–3245.

[20] Y. Cui, T. Matsubara, K. Sugimoto, Kernel dynamic policy programming: applicable reinforcement learning to robot systems with high dimensional states, Neural Netw. 94 (2017) 13–23.

[21] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, N. De Freitas, Dueling network architectures for deep reinforcement learning, in: International Conference on Machine Learning (ICML), 2016, pp. 1995–2003.

[22] D. Tanaka, S. Arnold, K. Yamazaki, EMD Net: An encode-manipulate-decode network for cloth manipulation, IEEE Robot. Autom. Lett. 3 (3) (2018) 1771–1778.

[23] P.C. Yang, K. Sasaki, K. Suzuki, K. Kase, S. Sugano, T. Ogata, Repeatable folding task by humanoid robot worker using deep learning, IEEE Robot. Autom. Lett. 2 (2) (2017) 397–403.

[24] Y. Tsurumine, Y. Cui, E. Uchibe, T. Matsubara, Deep dynamic policy programming for robot control with raw images, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 1545–1550.

[25] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning., in: Association of the Advancement of Artificial Intelligence (AAAI), 2016, pp. 2094–2100.

[26] S. Gu, E. Holly, T. Lillicrap, S. Levine, Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates, in: IEEE International Conference on Robotics and Automation (ICRA), 2017.

[27] A.A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, R. Hadsell, Sim-to-real robot learning from pixels with progressive nets, in: Conference on Robot Learning (CoRL), PMLR, 2017, pp. 262–270.

[28] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, M.A. Riedmiller, Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards, Computing Research Repository (CoRR) abs/1707.08817.

[29] H. Van Hoof, G. Neumann, J. Peters, Non-parametric policy search with limited information loss, J. Mach. Learn. Res. 18 (1) (2017) 2472–2517.

[30] S. Levine, V. Koltun, Guided policy search, in: International Conference on Machine Learning (ICML), 2013, pp. 1–9.

[31] C. Daniel, G. Neumann, J. Peters, Hierarchical relative entropy policy search, in: International Conference on Artificial Intelligence and Statistics (AISTATS), 2012, pp. 273–281.

[32] S. Levine, C. Finn, T. Darrell, P. Abbeel, End-to-end training of deep visuomotor policies, J. Mach. Learn. Res. 17 (1) (2016) 1334–1373.

[33] K. Yamazaki, Grasping point selection on an item of crumpled clothing based on relational shape description, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2014, pp. 3123–3128.

[34] L. Sun, G. Aragon-Camarasa, S. Rogers, J.P. Siebert, Accurate garment surface analysis using an active stereo robot head with application to dual-arm flattening, in: IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 185–192.

[35] H. Yuba, S. Arnold, K. Yamazaki, Unfolding of a rectangular cloth from unarranged starting shapes by a Dual-Armed robot with a mechanism for managing recognition error and uncertainty, Adv. Robot. 31 (10) (2017) 544–556.

[36] S. Miller, J. Van-Den-Berg, M. Fritz, T. Darrell, K. Goldberg, P. Abbeel, A geometric approach to robotic laundry folding, Int. J. Robot. Res. 31 (2) (2012) 249–267.

[37] A. Ramisa, G. Alenyá, F. Moreno-Noguer, C. Torras, Finddd: A fast 3d descriptor to characterize textiles for robot manipulation, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013, pp. 824–830.

[38] A. Doumanoglou, A. Kargakos, T.-K. Kim, S. Malassiotis, Autonomous active recognition and unfolding of clothes using random decision forests and probabilistic planning, in: IEEE International Conference on Robotics and Automation (ICRA), 2014, pp. 987–993.

[39] J. Stria, D. Prusa, V. Hlavac, L. Wagner, V. Petrik, P. Krsek, V. Smutny, Garment perception and its folding using a dual-arm robot, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2014, pp. 61–67.

[40] Y. Li, Y. Yue, D. Xu, E. Grinspun, P.K. Allen, Folding deformable objects using predictive simulation and trajectory optimization, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 6000–6006.

[41] N. Koganti, T. Tamei, K. Ikeda, T. Shibata, Bayesian nonparametric learning of cloth models for real-time state estimation, IEEE Trans. Robot. 33 (4) (2017) 916–931.

[42] A. Doumanoglou, J. Stria, G. Peleka, I. Mariolis, V. Petrik, A. Kargakos, L. Wagner, V. Hlaváč, T.K. Kim, S. Malassiotis, Folding clothes autonomously: A complete pipeline, IEEE Trans. Robot. 32 (6) (2016) 1461–1478.

[43] T. Matsubara, D. Shinohara, M. Kidode, Reinforcement learning of a motor skill for wearing a T-shirt using topology coordinates, Adv. Robot. 27 (7) (2013) 513–524.

[44] A. Colomé, A. Planells, C. Torras, A friction-model-based framework for reinforcement learning of robotic tasks in non-rigid environments, in: IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 5649–5654.

[45] A.X. Lee, H. Lu, A. Gupta, S. Levine, P. Abbeel, Learning force-based manipulation of deformable objects from multiple demonstrations, in: IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 177–184.

[46] A. Gabas, E. Corona, G. Alenyá, C. Torras, Robot-aided cloth classification using depth information and CNNs, in: International Conference on Articulated Motion and Deformable Objects (ICAMDO), 2016, pp. 16–23.

[47] E. Corona, G. Alenyà, A. Gabas, C. Torras, Active garment recognition and target grasping point detection using deep learning, Pattern Recogn. 74 (4) (2018) 629–641.

[48] J. Kober, J.A. Bagnell, J. Peters, Reinforcement learning in robotics: A survey, Int. J. Robot. Res. 32 (11) (2013) 1238–1274.

[49] C.J. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (3–4) (1992) 279–292.

[50] R.S. Sutton, Generalization in reinforcement learning: Successful examples using sparse coarse coding, Adv. Neural Inf. Process. Syst. (NIPS) (1996) 1038–1044.

[51] M.G. Lagoudakis, R. Parr, Least-squares policy iteration, J. Mach. Learn. Res. 4 (44) (2003) 1107–1149.

[52] M.G. Azar, V. Gómez, B. Kappen, Dynamic policy programming with function approximation, in: International Conference on Artificial Intelligence and Statistics (AISTATS), 2011, pp. 119–127.

[53] E. Todorov, Linearly-solvable Markov decision problems, in: Advances in Neural Information Processing Systems (NIPS), 2006, pp. 1369–1376.

[54] Y. Cui, T. Matsubara, K. Sugimoto, Pneumatic artificial muscle-driven robot control using local update reinforcement learning, Adv. Robot. (2017) 1–16.

[55] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J.Z. Leibo, A. Gruslys, Learning from demonstrations for real world reinforcement learning, Computing Research Repository (CoRR) abs/1704.03732.
[56] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, et al., Tensorflow: Large-scale machine learning on heterogeneous distributed systems, arXiv preprint arXiv:1603.04467.
[57] F. Chollet, et al., Keras, 2017. https://github.com/keras-team/keras.
[58] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A.Y. Ng, ROS: an open-source Robot Operating System, in: ICRA Workshop on Open Source Software, 2009, p. 5.
[59] R.R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra, Grad-cam: Visual explanations from deep networks via gradient-based localization, in: IEEE International Conference on Computer Vision (ICCV), 2018, pp. 618–626.
[60] T. Jung, D. Polani, Kernelizing lspe(λ), in: 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007, pp. 338–345.
[61] C. Alessandro, I. Delis, F. Nori, S. Panzeri, B. Berret, Muscle synergies in neuroscience and robotics: from input-space to task-space perspectives, Front. Comput. Neurosci. 7 (2013) 43.
[62] N. Hayashi, T. Suehiro, S. Kudoh, Planning method for a wrapping-with-fabric task using regrasping, in: IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 1285–1290.
[63] K. Arulkumaran, M.P. Deisenroth, M. Brundage, A.A. Bharath, Deep reinforcement learning: A brief survey, IEEE Signal Process. Mag. 34 (6) (2017) 26–38.
[64] P.M. Pilarski, M.R. Dawson, T. Degris, F. Fahimi, J.P. Carey, R.S. Sutton, Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning, in: IEEE International Conference on Rehabilitation Robotics (ICORR), 2011, pp. 1–7.
[65] P.F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, D. Amodei, Deep reinforcement learning from human preferences, in: Advances in Neural Information Processing Systems (NIPS), 2017, pp. 4299–4307.
[66] C. Daniel, M. Viering, J. Metz, O. Kroemer, J. Peters, Active reward learning, in: Robotics: Science and Systems (RSS), 2014.
[67] E. Uchibe, Model-free deep inverse reinforcement learning by logistic regression, Neural Process. Lett. (2017) 1–15.
[68] B. Eysenbach, S. Gu, J. Ibarz, S. Levine, Leave no trace: Learning to reset for safe and autonomous reinforcement learning, in: International Conference on Learning Representations (ICLR), 2018.

Yoshihisa Tsurumine received his B.E. in Advanced Course of Production Systems Engineering from National Institute of Technology, Ube College, Yamaguci, Japan, in 2016 and his M.E. in information science from the Nara Institute of Science and Technology, Nara, Japan, in 2018. His research interests are robot control using machine learning.

Yunduan Cui was born in China in 1990. He is currently undertaking a research assistant professor at Nara Institute of Science and Technology, Japan. He received Ph.D. in information science from Nara Institute of Science and Technology, Japan in September 2017, M.E in computer science from Doshisha University, Japan in September 2014, and the B.E. in Electronic Engineering from Xidian University, China in 2012. His research interests are machine learning and control theory of robotics, especially reinforcement learning in robot control.

Eiji Uchibe received his B.S. in 1994, M.S. in 1996, and Ph.D. in 1999, at Osaka University. He worked as a research associate of the Japan Society for the Promotion of Science, in the Research for the Future Program titled Cooperative Distributed Vision for Dynamic Three-Dimensional Scene Understanding. Then he joined ATR as a researcher in 2001. Since 2004 he has been a group leader of Adaptive Systems Group at the Neural Computation Unit, Okinawa Institute of Science and Technology Graduate University. He joined Department of Brain Robot Interface,ATR Computational Neuroscience Laboratories in 2015 as Principal Researcher. His research interests are in learning robots, reinforcement learning, evolutionary computation, and computational neuroscience, and their applications.

Takamitsu Matsubara received his B.E. in electrical and electronic systems engineering from Osaka Prefecture University, Osaka, Japan, in 2003, an M.E. in information science from the Nara Institute of Science and Technology, Nara, Japan, in 2005, and a Ph.D. in information science from the Nara Institute of Science and Technology, Nara, Japan, in 2007. From 2005 to 2007, he was a research fellow (DC1) of the Japan Society for the Promotion of Science. From 2013 to 2014, he is a visiting researcher of the Donders Institute for Brain Cognition and Behavior, Radboud University Nijmegen, Nijmegen, The Netherlands. He is currently an associate professor at the Nara Institute of Science and Technology and a visiting researcher at the ATR Computational Neuroscience Laboratories, Kyoto, Japan. His research interests are machine learning and control theory for robotics.