A recurrent reinforcement learning approach applicable to highly uncertain environments

Yang Li¹, Shijie Guo¹, Lishuang Zhu¹, Toshiharu Mukai² and Zhongxue Gan³

Abstract
Reinforcement learning has been a promising approach in control and robotics since data-driven learning leads to non-necessity of engineering knowledge. However, it usually requires many interactions with environments to train a controller. This is a practical limitation in some real environments, for example, robots where interactions with environments are restricted and time inefficient. Thus, learning is generally conducted with a simulation environment, and after the learning, migration is performed to apply the learned policy to the real environment, but the differences between the simulation environment and the real environment, for example, friction coefficients at joints, changing loads, may cause undesired results on the migration. To solve this problem, most learning approaches concentrate on retraining, system or parameter identification, as well as adaptive policy training. In this article, we propose an approach where an adaptive policy is learned by extracting more information from the data. An environmental encoder, which indirectly reflects the parameters of an environment, is trained by explicitly incorporating model uncertainties into long-term planning and policy learning. This approach can identify the environment differences when migrating the learned policy to a real environment, thus increase the adaptability of the policy. Moreover, its applicability to autonomous learning in control tasks is also verified.

Keywords
Reinforcement learning, migration, adaptive policy, environmental encoder

Date received: 23 October 2019; accepted: 16 February 2020

Introduction
Recently, reinforcement learning (RL) has shown its immense potential for processing complex and large-scale tasks.¹⁻³ In particular, it becomes a useful approach to realize optimal control in robotics since data-driven learning leads to non-necessity of engineering knowledge,⁴ which is usually difficult to obtain.⁵⁻⁷ However, learning is prohibitively slow, that is, the required number of interactions with the environment is impractically large. Even in problems with low-dimensional state spaces or fairly benign dynamics, thousands of trials are usually required in

¹ State Key Laboratory for Reliability and Intelligence of Electrical Equipment, Hebei Key Laboratory of Robot Sensing and Human-Robot Interaction, Hebei University of Technology, Tianjin, China
² Faculty of Information Science, Meijyo University, Nagoya, Japan
³ Institute of AI and Robotics, Fudan University, Shanghai, China

Corresponding author:
Shijie Guo, State Key Laboratory for Reliability and Intelligence of Electrical Equipment, Hebei Key Laboratory of Robot Sensing and Human-Robot Interaction, Hebei University of Technology, No. 8, No. 1 Road, Dingzigu, Hongqiao District, Tianjin 300130, China.
Email: guoshijie@hebut.edu.cn

Creative Commons CC BY: This article is distributed under the terms of the Creative Commons Attribution 4.0 License (https://creativecommons.org/licenses/by/4.0/) which permits any use, reproduction and distribution of the work without further permission provided the original work is attributed as specified on the SAGE and Open Access pages (https://us.sagepub.com/en-us/nam/open-access-at-sage).
learning. This data inefficiency makes it impractical to apply RL to real robotics and prohibits RL approaches in more challenging scenarios. Thus, learning is generally conducted with a simulation model, and after that, a migration process is required from the simulation environment to the real environment. However, the errors (commonly referred to as reality gap (RG)) between the simulation environment and the real environment make it challenging to apply the learned policy to the real environment. In general, adding additional measurement sensors can increase the adaptability of a learned policy, but this is both cost inefficient and time inefficient, and furthermore, the difference between the simulation environment and the real environment cannot be clearly understood. Therefore, it is crucial to train an adaptive policy that can be applied to an environment with high uncertainty.

Suppression of RG caused by model difference and/or uncertainty in policy migration has become a hot research topic. Motor primitives have been introduced to accelerate learning speed and reduce task complexity as well as the number of trials, but retraining needs to be conducted after migration. Trials focusing on system identification have also been reported, which provide a framework for solving the problem. System identification can help to generalize the knowledge of the system to unobserved states, thus reducing the number of trials for policy optimization. However, the learned policy still relies on the number of trials and the quality of the data.

On the other hand, uncertainties are treated as noise, which can be handled by a robust control policy. Lee et al. used a Bayesian network to estimate the error among environments and developed a robust policy. However, since uncertainty could not be fully considered, this approach was effective only when the dynamic model was a good approximation of the real environment. Unfortunately, this condition usually cannot be satisfied in a complex dynamic system like an actual robot. To solve the problem, Yu et al. built an online system identification model that was able to consider all the uncertain factors, but the accuracy of Q-value would be decreased since the window of motion history was narrowed, and furthermore, it took model parameters as a training target.

In this article, we propose a recurrent RL approach, which is based on the deep deterministic policy gradient (DDPG) architecture. It can achieve an adaptive policy by combining an environmental encoder (EE) with a universal policy. As recurrent neural network (RNN) can integrate the information across time frames, the EE is built by RNN from motion history (time series of state–action pairs). The proposed approach is called as recurrent DDPG (RDDPG). The critic network and EE are trained to estimate Q-values in any possible situations of an uncertain environment. The latent variables between the EE and the critic network are defined as meta-parameters, which are used to identify the parameters in continuous state–action domains. Thus, this approach can give an accurate estimation of the Q-values and consequently achieve an adaptive policy.

This article is organized as follows: the related work is described in the second section, and the key ideas of the proposed approach, that is, learning framework, policy improvement, and unsupervised learning of the EE, are given in the third section. The fourth section describes the simulation experiment and discusses the effectiveness of the proposed approach.

**Related work**

Deep Q-network is a well-known deep RL method proposed by DeepMind. It achieved massive success in higher-dimensional problems with discrete action spaces, such as the Atari game. However, in many tasks of interest, especially physical control tasks, the action space is continuous. To solve this problem, DDPG was proposed. It is an algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function and uses the Q-function to learn the policy. DDPG gained great success in observable problems, such as the cart-pole swing-up task and the reaching task. It can learn value functions in stable and robust way because the network is trained off-policy with samples from a replay buffer to minimize correlations between samples. The network is trained with a target Q-network to give consistent targets.

However, the task studied in this article can be classified as a partially observable Markov decision process (POMDP) due to the existence of environmental uncertainty. The agent cannot directly observe the parameters of the environment because they fall outside of the history window. RNN can solve this type of problem since it learns across time series data.

In general, POMDP is a sequential decision-making model where the underlying states of an environment are partially available or the observation received by the agent is an incomplete state. It can be described as a 6-tuple \((S, A, P, R, O, \mathcal{Q})\), where \(S\), \(A\), \(P\), and \(R\) are states, actions, transitions, and rewards, respectively, whereas \(O\) and \(O\) are the observations and conditional probability, respectively. The agent receives observation \(o \in \mathcal{O}\) instead of the complete system state \(s \in S\). The observation is generated from the underlying system state according to the probability distribution \(o \sim \mathcal{O}(S)\). RL has no explicit mechanisms for deciphering the underlying state of POMDP and it is effective only when the observation reflects the state of the environment. In our case, the Q-value and action of the learned policy cannot be accurately generated from the observation of POMDP since \(o = [s, p]\), where \(p \in P\) is environment parameters. Thus, \(Q(o, a) \neq Q(s, a)\) and \(a(o) \neq a(s)\). To solve this problem,
we narrowed the gaps between the two pairs, that is, $Q(o, a)/Q(s, a)$ and $a(o)/a(s)$.

The parameters of an environment can be expressed by the time series of state–action pairs. Therefore, it is possible to extract information from the time series by RNN to refine the observation. Thus, we introduced recurrence to RL to build meta-parameters so that the actual Q-values could be estimated from a value network. In applying RNN, we used long short-term memory (LSTM), which is designed to supervise time series learning for long-term dependencies, for solving the problem where errors propagate back in time. The problems of vanishing/exploding gradients in LSTM can be prevented by using constant error carousels (CECs).21

Figure 1 shows the architecture of LSTM. It adds or deletes information with gates, which can selectively allow information to pass through a sigmoid layer $σ$. LSTM uses three gate structures, that is, forget gates, input gates, and output gates. Forget gates yield a vector $f$, according to the output $y_{t-1}$ in the previous moment and the input $x_t$ in the current moment. Input gates determine CECs $i_t$ according to middle information $i_t$ and $c_t$. Output gates determine output $y_t$ according to $c_t$ and $o_t$. The error between the output prediction $y_t(x_i|w_j)$ and the target $y_t^*(x_i)$ is minimized by updating the weight $w_j$, and this updating is conducted at each time step as

$$w_j \leftarrow \arg\min_{w_j} \sum_t (y_t(x_i|w_j) - y_t^*(x_i))^2$$

Since the physical parameters of the environment are changed randomly, the Q-value changes in each episode of learning even though the policy is unchanged. This change causes the uncertainty of the value network. However, an accurate value network is the premise of an optimal policy. To solve this problem, we use meta-parameters, which are generated by LSTM to reflect the change of the Q-value. Taking meta-parameters as additional input in the learning, the value network can be specific in each episode.

**RDDPG architecture**

In RL, an agent receives a state $s_t$ and takes an action $a_t$ based on the state $s_t$ at time $t$ and then the environment encounters a new state $s_{t+1}$ and a reward $r_t$. Since this article focuses on model-free learning, the agent transits $(s_t, a_t)$ to $s_{t+1}$ and gets a reward $r_t$ from $(s_{t+1}, a_{t+1})$ regarding the task. The deterministic policy $π$, which is parameterized by $ω_π$, takes state $s_t$ as an input and generate action $a_t$ as the output. The value network, which is parameterized by $ω_v$, takes state $s_t$ and action $a_t$ as inputs and yields discounted future reward as the Q-value

$$Q(s_t, a_t) = \sum_{t=1}^{T} \gamma^{t-1} r_t = E[r_t + \gamma Q(s_{t+1}, a_{t+1})]$$  

(2)

where $\gamma \in [0, 1]$ is a discounting factor. The objective of the value network is to predict the expected discounted future reward. That of the policy network is to maximize the Q-value, which is assumed to be the return estimated by the value network.

The objective of RL is to maximize the Q-value shown in equation (2). The key problem here is the lack of parameter identification when the environment is changing. This makes the learned policy weak in adaptability. To address this question, we introduce recurrence to DDPG for parameters identification and call the improved approach as RDDPG. In this approach, the policy determines the action depending on time series data rather than current state.

DDPG is an actor-critic algorithm,24 which can learn policies in continuous action spaces, the optimization procedure in RDDPG is to update the policy network and the value network alternatively. The process is described in Figure 2, where LSTM, as an EE, yields meta-parameters as an additional input of the value network and the policy network. The value network is trained to estimate the Q-value by minimizing the temporal difference error. The policy network, that is, a nonstationary, meta-conditioned, deterministic policy,25 then yields a specific action by maximizing the Q-value. The EE is not parameterized by a certain task objective. Instead, it is optimized by a gradient back-propagation of a value network. Hence, the value network leads to a relatively accurate estimation of the Q-value, and the policy network takes an accurate action even though uncertainties exist.

Compared with a typical training scenario, in which a teacher and a student are deterministic single-task participants, an EE is a processor of time series shared across different environments. It provides meta-parameters to a single value network and a single policy network to deal with different environments. Explicitly, the EE, which is parameterized by $w_p$, takes the transition $s_t = [s_{t-1}, a_{t-1}, r_{t-1}, s_t]$ as input, which contains a state–action pair, and yields meta-parameters as

$$mp_t = M_p(s_t|w_p)$$  

(3)
The update rule is

\[
\begin{align*}
w_a &\leftarrow \arg \max_{w_a} Q(s_t, \pi(s_t, m_p), w_a) | w_c \\
w_c, w_p &\leftarrow \arg \min_{w_c, w_p} \left( Q(s_t, a_t, M_p(s_t, w_p)) + r_t - \gamma Q(s_{t+1}, \pi(s_{t+1}, m_p), w_a) \right)^2
\end{align*}
\]

in which \(Q\) is parameterized by \(\omega_a\). In the optimization of \(\omega_a\), the gradient of \(\omega_p\) was ignored since \(m_p\) was taken as a constant.

**Recurrent update**

DDPG updates the parameters of a network from samples of replay buffer \(R\), a finite-sized cache consisting of a fixed number of transitions \(s_t = (s_{t-1}, a_{t-1}, r_{t-1}, s_{t+1})\) where \(0 \leq t < J \times T\), in which \(J\) is a constant number and \(T\) is the maximum number of steps in an episode. LSTM generally uses a sequential update method to perform updates. \(21\) This method has the advantage of carrying LSTM’s hidden state forward from the beginning of the episode. However, it is against the random sampling policy of RL26 since it conducts sampling sequentially episode by episode rather than at the whole set of replay buffer. To overcome this problem, we tried to update at randomly selected segments of the episodes in the replay buffer. In the meantime, we need to determine the hidden state of EE at the beginning point of each randomly selected segment.

Here, we tested two types of updates. One is “zero-state update,” which initializes the hidden state parameters of EE to zero at beginning points, and the other is “save-state update,” which saves the hidden state parameters of each step in the replay buffer. Therefore, we used “zero-state update” in this research since it could decrease the space complexity of RDDPG. It does not need to save the hidden states of every step.

Buffer \(K\) stores \(st\) at every step of each episode. During the training, it randomly builds a set consisting of \(N\) subsets, \(L = [ST_0, ST_1, \ldots, ST_{N-1}]\), and each subset consists of \(K\) time sequence transitions sampled randomly from the buffer, \(ST_n = [st_{b(n)+0}, st_{b(n)+1}, \ldots, st_{b(n)+K-1}]\), where \(b(n)\) represents the beginning point of \(ST_n\), which is a randomly chosen number satisfying the following condition

\[
0 < b(n) < (J \times T) \quad \left( b(n) - \left\lfloor \frac{b(n)}{K} \right\rfloor \times K \right) < (T - K - 1)
\]

where \(J\) is a constant number, and \(T\) is the maximum number of steps in an episode.

Since \(st\) is a complete transition, we can build the set of the current states \(S_t = [s_{b(n)+1}, s_{b(n)+2}, \ldots, s_{b(n)+K}]\), the set of the next states \(S_n = [s_{b(n)+2}, s_{b(n)+3}, \ldots, s_{b(n)+K+1}]\), the set of the actions \(A_n = [a_{b(n)+1}, a_{b(n)+2}, \ldots, a_{b(n)+K}]\), as

---

**Figure 2.** The architecture of RDDPG. RDDPG: recurrent deep deterministic policy gradient.
well as the set of the rewards $R_n$. The main network in Figure 2 estimates the current Q-value. For instance, if $N=1$, the EE takes $s_{t-1}^{0}(t)+k$ and hidden state of the last step as inputs. It yields meta-parameters $m_{t-1}^{0}$ and hidden state that are used as inputs of the EE for the next step of training. Meanwhile, the meta-parameters $m_{t-1}^{0}$, $s_{t}^{0}(t)+k$, and the action $a_{t}^{0}(t)$ are inputted to the value network to yield the Q-value of the current step. The target network in Figure 2 estimates the Q-value of the next step in the same approach as the main network estimating the Q-value of the current step, but the action set $A_n$ is generated with the policy network.

On the other hand, to perform an update, we need to calculate the gradients of the EE at each step. LSTM usually calculates the gradients by minimizing the error between the predicted output and the target output. But we do not use this process since EE does not take the parameters of the environment as targets in training. In this study, the gradients were calculated with equation (5), where the weight $\omega_p$ of the EE and the weight $\omega_c$ of the value network were updated by minimizing the temporal difference error. The weight $\omega_p$ of the policy network was updated by maximizing the Q-value.

The algorithm

Errors are inevitable due to the difference between the actual Q-value $[Q(s, \pi(s), p)]$ and the calculated Q-value $[Q(s, \pi(s))]$, where $p$ represents the parameter set of the environment which changes randomly in each episode. The change of parameters makes it difficult for the value network to converge to the actual Q-value and to optimize the policy network. In this study, since learning was performed with a simulation model by randomly changing the model parameters, we supposed that the learning was conducted with many randomly distributed models.

There is no feedback in DDPG for distinguishing the models. The policy obtained with DDPG may be optimal to an “averaged” model among all the models, but it cannot provide an optimal policy to the tasks with different environments because the Q-value is not related to model changes. While, in RDDPG, owing to the existence of the EE, the learned policy is associated with each specific model because meta-parameters can reflect model parameters as feedbacks. The detail of RDDPG is shown in Algorithm 1. EE generates meta-parameters from the time series of state–action pairs. As a result, a policy can be obtained by incorporating $m_{t}$ as an additional input into the value network and the policy network. Using $m_{t}$ can help to narrow the gap between $Q(s, \pi(s), p_t)$ and $Q(s, \pi'(s, p_t), p_t)$.
This means RDDPG can treat the problem of POMDP. Here, it should be noted that meta-parameters $mp$ keep model parameters, but they are neither model parameters nor measures that help to find model parameters.

### Experiment

When we apply a policy that is learned on a simulation model to a real environment, the situation is the same as that we apply a policy learned on one environment to another environment. To confirm the effectiveness of RDDPG, three types of tasks were constructed, each one contained an environment with a few uncertainties. We conducted the experiments using a low-dimensional state description with joint angles and positions. The characteristic parameters (the weight and length of each link, the damping of each joint, etc.) were changed randomly within a certain band in each episode. Figure 3(a) is a two-degrees of freedom (2-DOF) cart-pole model. Figure 3(b) and (c) are 2-DOF manipulators with and without loads, respectively. The task shown in Figure 3(a) was to control the pole to keep a vertical position. In each training episode, the mass and length of the pole changed randomly. On the other hand, the tasks shown in Figure 3(b) and (c) were to control the manipulators to reach certain positions in their working spaces.

| Parameter (unit) | Abbreviation | Cart-pole | Manipulator | Load manipulator |
|------------------|--------------|-----------|-------------|------------------|
| Length of link 1 (m) | $L_1$ | 4.0 | 2.0 | 2.0 | 2.0 |
| Mass of link 1 (kg) | $M_1$ | 16.0 | 30.0 | 1.0 | 2.5 | 1.0 | 2.5 |
| Length of link 2 (m) | $L_2$ | --- | --- | 2.0 | 2.5 | 2.0 | 2.5 |
| Mass of link 2 (kg) | $M_2$ | --- | --- | 1.0 | 2.5 | 1.0 | 2.5 |
| Damping of joint 1 (Ns/m) | $D_1$ | --- | --- | 5.0 | 10.0 | 5.0 | 10.0 |
| Damping of joint 2 (Ns/m) | $D_2$ | --- | --- | 3.0 | 6.0 | 3.0 | 6.0 |
| Mass of load (kg) | $M_3$ | --- | --- | --- | --- | 0.0 | 12.0 |

Figure 3. Model robots as environments for comparing RDDPG and DDPG. (a) Cart-pole. (b) Manipulator. (c) Load manipulator. (a) Cart-pole balance task with variable masses of the pole and the cart. (b) and (c) Positioning tasks of manipulators with and without load. The lengths and masses of links 1 and 2, the damping at joints 1 and 2, and the load are variables. RDDPG: recurrent deep deterministic policy gradient; DDPG: deep deterministic policy gradient.

Figure 4. Comparison of total rewards during the learning of RDDPG and DDPG for the three tasks. The parameters of environments are varied randomly at the beginning of each episode. (a) Cart-pole. (b) Manipulator. (c) Load manipulator. RDDPG: recurrent deep deterministic policy gradient; DDPG: deep deterministic policy gradient.

$Q(s, \pi(s, mp), mp)$. This means RDDPG can treat the problem of POMDP. Here, it should be noted that meta-parameters $mp$ keep model parameters, but they are neither model parameters nor measures that help to find model parameters.
As shown in Figure 3(a) and (b), the environments could be considered having uncertain bounded parameters, while the task shown in Figure 3(c) was a task with external load. The ranges of the parameters are given in Table 1. Figure 4 is a comparison of the performances of DDPG and RDDPG for different tasks. The total reward was defined as

\[ J = \int \left[ e^T(t)\Lambda e(t) + a^T(t)\Sigma a(t) \right] dt \]  

where \( e(t) \) and \( a(t) \) represent the positioning error and action value at the step \( t \), respectively, whereas \( \Lambda \) and \( \Sigma \) are positive definite matrices.

It can be seen from Figure 4 that RDDPG had better total reward than DDPG during learning. In the cart-pole task, the two algorithms gave almost the same results, as shown in Figure 4(a), but RDDPG was more stable. This result indicates that meta-parameters can efficiently reflect the
parameters of an environment. To further confirm the effectiveness of RDDPG, we compared the control performance of the two algorithms for the model robot, as shown in Figure 3(c). Figure 5(a) and (b) is the action (driving torques) at each step, and Figure 5(c) gives positioning errors of the end of the manipulator at each step. The positioning errors and residual oscillations with the policy learned by RDDPG were relatively small in comparison with DDPG, and the oscillations of joint torques were also limited. The two algorithms showed different features in the adaption to an environment. Although optimum control and feedback control invoke different philosophies, the results obtained in the experiment demonstrate that RDDPG could provide a robust performance as a feedback controller which could reduce steady-state errors, whereas DDPG behaved as an open-loop controller.

The physical parameters of the model robots, as shown in Figure 3, were arbitrarily chosen. The same learned policy could provide almost the same manipulation performance even though the parameters of the environment were changed within an “error band.” Figure 6 compares the performance of the learned policies for the cart-pole model and the 2-DOF manipulator model by RDDPG and DDPG. The former provided higher total reward than the latter. To further investigate the capability of RDDPG, the learned policy was applied to the model robot, as shown in Figure 3(c), with changing external load. The total rewards are shown in Figure 7. RDDPG gave a better performance than DDPG at any loads, demonstrating an excellent capability in dealing with an uncertain environment.

**Conclusions**

To make a learned policy on a simulation model to adapt to a real environment with limited uncertainties, an RL approach, which is called RDDPG, was proposed. It features the use of the EE and extensive training by changing environment parameters randomly in a limited range. Simulation experiments were conducted and the results demonstrated that the learned policy could adapt to a dynamic model with high uncertainties, and indeed, it identified the parameters of the model in real time. Simulation experiments on three model robots showed that RDDPG could reduce positioning errors and residual oscillations of both positions and joint torques considerably as compared to traditional DDPG. It is different from DDPG in adapting to an uncertain environment. It relies on the EE that exploits time series data. The EE enables it to deal with the uncertainties of the characteristics of an environment. It is believed that the high adaptability of RDDPG comes from the precomputing of the possible models and the identification of parameters with an EE, which is used to identify the parameters. This process can avoid excessive reliance on the accuracy of the parameters.

In the forthcoming study, we will apply the proposed approach to an actual robot to confirm its effectiveness to real problems.

**Declaration of conflicting interests**

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

**Funding**

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the National Key Research and Development Plan (2016YFE0128700), the Natural Science Foundation of Hebei Province (E2017202270), the Key Research and Development Plan of Hebei Province (18211816D), and the National Key Research and Development Plan (2017YFB1301002).

**ORCID iD**

Yang Li https://orcid.org/0000-0002-0840-5796

**References**

1. Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. *Nature* 2015; 518(7540): 529–533.
2. Silver D, Huang A, Maddison CJ, et al. Mastering the game of Go with deep neural networks and tree search. *Nature* 2016; 529(7587): 484–489.
3. Jaderberg M, Mnih V, Czarnecki WM, et al. Reinforcement learning with unsupervised auxiliary tasks, https://arxiv.org/abs/1611.05397 (accessed 16 November 2016).
4. Lake BM, Ullman TD, Tenenbaum JB, et al. Building machines that learn and think like people. *Behav Brain Sci* 2017; 40: 1.
5. Niu J, Wang H, Shi H, et al. Study on structural modeling and kinematics analysis of a novel wheel-legged rescue robot. *Int J Adv Rob Syst* 2018; 15: 1.
6. Bozek P, Pokorný P, Svetlík J, et al. The calculations of Jordan curves trajectory of the robot movement. *Int J Adv Rob Syst* 2016; 13(5): 1729881416663665.
7. Božek P, Ivandić Ž, Lozhkin A, et al. Solutions to the characteristic equation for industrial robot’s elliptic trajectories. *Teh Vjesn* 2016; 23(4): 1017–1023.
8. Koos S, Mouret JB, and Doncieux S. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In: Proceedings of the 12th annual conference on genetic and evolutionary computation, Portland, USA, 7–11 July 2010, pp. 119–126. New York: ACM.

9. Pivarciová E, Božek P, Turygin Y, et al. Analysis of control and correction options of mobile robot trajectory by an inertial navigation system. Int J Adv Rob Syst 2018; 15(1): 1729881418755165.

10. Mcgovern A and Barto AG. Automatic discovery of subgoals in reinforcement learning using diverse density. In: Eighteenth international conference on machine learning, San Francisco, USA, 28 June–1 July 2001, pp. 361–368. New York: ACM.

11. Kober J and Peters JR. Policy search for motor primitives in robotics. In: Advances in neural information processing systems, Vancouver, Canada, 6–8 December 2009, pp. 849–856. Cambridge: MIT.

12. Yoshimoto J, Ishii S, and Sato M. System identification based on online variational Bayes method and its application to reinforcement learning. In: Artificial neural networks and neural information processing—ICANN/ICONIP 2003, joint international conference ICANN/ICONIP 2003, Istanbul, Turkey, 26–29 June 2003, pp. 123–131. Berlin: Springer.

13. Abbeel P and Ng AY. Exploration and apprenticeship learning in reinforcement learning. In: International conference on machine learning, Bonn, Germany, 7–11 August 2005, pp. 1–8. New York: ACM.

14. Bongard JC and Lipson H. Nonlinear system identification using coevolution of models and tests. IEEE Trans Evol Comput 2005; 9(4): 361–384.

15. Ting JA, D’Souza A, and Schaal S. Bayesian robot system identification with input and output noise. Neural Netw 2011; 24(1): 99–108.

16. Lee G, Hou B, Mandalika A, et al. Bayesian policy optimization for model uncertainty, https://arxiv.org/abs/1810.01014 (accessed 8 May 2019).

17. Abbeel P, Quigley M, and Ng AY. Using inaccurate models in reinforcement learning. In: Proceedings of the 23rd international conference on Machine learning, Pennsylvania, USA, 25–29 June 2006, pp. 1–8. New York: ACM.

18. Yu W, Tan J, Liu CK, et al. Preparing for the unknown: learning a universal policy with online system identification, https://arxiv.org/abs/1702.02453 (accessed 15 May 2017).

19. Lillicrap TP, Hunt JJ, Pritzel A, et al. Continuous control with deep reinforcement learning, https://arxiv.org/abs/1509.02971 (accessed 5 July 2019).

20. Schmidhuber J. Reinforcement learning in Markovian and non-Markovian environments. Adv Neural Inf Proc Syst 1991; 3: 500–506.

21. Hochreiter S and Schmidhuber J. Long short-term memory. Neural Comput 1997; 9(8): 1735–1780.

22. Silva BD, Konidaris G, and Barto A. Learning parameterized skills. In: International conference on machine learning, Edinburgh, Scotland, 27 June–3 July 2012. New York: ACM.

23. Kupcsik AG, Deisenroth MP, Peters J, et al. Data-efficient generalization of robot skills with contextual policy search. In: Twenty-seventh AAAI conference on artificial intelligence, Washington, USA, 14–18 July 2013, pp. 1401–1407. Palo Alto: AAAI.

24. Grondman I, Busoniu L, Lopes GAD, et al. A survey of actor-critic reinforcement learning: standard and natural policy gradients. IEEE Trans Syst Man Cybern Part C Appl Rev 2012; 42(6): 1291–1307.

25. Silver D, Lever G, Heess N, et al. Deterministic policy gradient algorithms. In: International conference on machine learning, Beijing, China, 21–26 June 2014. New York: ACM.

26. Mnih V, Kavukcuoglu K, Silver D, et al. Playing Atari with deep reinforcement learning, https://arxiv.org/abs/1312.5602v1 (accessed 19 Dec 2013).

27. Gers FA, Schmidhuber J, and Cummins F. Learning to forget: continual prediction with LSTM. Neural Comput 2000; 12(10): 2451–2471.

28. Hausknecht M and Stone P. Deep recurrent Q-learning for partially observable MDPs, https://arxiv.org/abs/1507.06527 (accessed 11 Jan 2017).

29. Lewis FL, Vrabie D, and Vamvoudakis KG. Reinforcement learning and feedback control: using natural decision methods to design optimal adaptive controllers. IEEE Control Syst Mag 2012; 32(6): 76–105.