frontiers in Neurorobotics

ORIGINAL RESEARCH
published: 27 September 2018
doi: 10.3389/fnbot.2018.00061

Cooperative and Competitive Reinforcement and Imitation Learning for a Mixture of Heterogeneous Learning Modules

*Eiji Uchibe**

Advanced Telecommunications Research Computational Neuroscience Laboratories, Department of Brain Robot Interface, Kyoto, Japan

This paper proposes Cooperative and competitive Reinforcement And Imitation Learning (CRAIL) for selecting an appropriate policy from a set of multiple heterogeneous modules and training all of them in parallel. Each learning module has its own network architecture and improves the policy based on an off-policy reinforcement learning algorithm and behavior cloning from samples collected by a behavior policy that is constructed by a combination of all the policies. Since the mixing weights are determined by the performance of the module, a better policy is automatically selected based on the learning progress. Experimental results on a benchmark control task show that CRAIL successfully achieves fast learning by allowing modules with complicated network structures to exploit task-relevant samples for training.

Keywords: reinforcement learning, imitation learning, modular architecture, parallel learning, entropy-regularization, multiple importance sampling

OPEN ACCESS

Edited by:
Andrew Barto,
University of Massachusetts Amherst, United States

Reviewed by:
Gerhard Neumann,
Technische Universität Darmstadt, Germany
Luca Citi,
University of Essex, United Kingdom

***Correspondence:**
Eiji Uchibe
uchibe@atr.jp

Received: *31 May 2018*
Accepted: *06 September 2018*
Published: *27 September 2018*

Citation:
Uchibe E (2018) Cooperative and Competitive Reinforcement and Imitation Learning for a Mixture of Heterogeneous Learning Modules. Front. Neurorobot. 12:61. doi: 10.3389/fnbot.2018.00061

1. INTRODUCTION

Reinforcement Learning (RL) (Sutton and Barto, 1998; Kober et al., 2013) is an attractive learning framework with a wide range of possible application areas. A learning agent attempts to find a policy that maximizes its total amount of reward received during interaction with its environment. Recently, such nonlinear function approximators as artificial neural networks are being used to approximate a policy with the help of deep learning. Deep Reinforcement Learning (DRL), which integrates both deep learning and reinforcement learning, has achieved several remarkable successes in decision-making tasks, such as playing video games (Mnih et al., 2015) and the board game Go (Silver et al., 2016, 2017).

However, DRL's performance critically depends on its architectures, learning algorithms, and meta-parameters (Henderson et al., 2018). On one hand, a shallow Neural Network (NN) with fewer connection weights usually learns faster, but its performance may be limited. A deep and/or wide NN with many network weights can represent any complex policy, but it usually needs a huge amount of experiences to find an appropriate one. Since the motivation to use NNs is to represent complicated nonlinear mapping from state to action, it is reasonable to select a deep and wide NN as a function approximator. However, training data must be gathered by the learning agent for reinforcement learning as opposed to the standard settings of the classification problems of deep learning. Since a complicated NN policy whose many weights are initialized randomly does not collect useful experiences to seek its goal, it is not promising to collect

good experiences by itself, especially at the beginning of the learning. Therefore, we have to find an appropriate network architecture based on the task's complexity. Although an evolutionary method was applied to the problem of a neural architecture search (Whiteson and Stone, 2006) for tiny problems, experimenters usually manually prepare a learning module with an appropriate network architecture depending on the situation. Furthermore, it is crucial to select an appropriate RL algorithm based on the given task. For instance, two major types of algorithms exist: value-based reinforcement learning and policy search methods, including policy gradient reinforcement learning. Such value-based reinforcement learning as Q-learning (Watkins and Dayan, 1992) and SARSA (Rummery and Niranjan, 1994) learns faster than vanilla policy search methods such as REINFORCE (Williams, 1992) because value-based reinforcement learning exploits the Bellman equation under the Markovian assumption. The policy search methods are robust and find a better stochastic policy even if the state representation is deficient (Kalyanakrishnan and Stone, 2011).

In practice, experimenters test different combinations to select the best one since their appropriate combination is unknown in advance. Moreover, since the sequential testing of these factors is very time-consuming, to eliminate the need for such human hand-tuning, we proposed Cooperative and competitive Learning with Importance Sampling (CLIS) (Uchibe and Doya, 2004, 2005). Here, the agent possesses multiple heterogeneous learning modules and selects an appropriate module based on the task and its experience. We consider a mechanism by which an agent can best utilize its behavioral experiences to train multiple learning modules with different network architecture and learning algorithms. By exploiting task-relevant experiences gathered by suboptimal but fast-learning modules, a complicated module learns faster than when it was trained alone. Unfortunately, CLIS is unstable in learning for several reasons. One is the naive use of importance sampling to compensate for the mismatch in the target and behavior policies. The other is that the original CLIS adopts classical RL algorithms and linear function approximators. In addition, the application of CLIS to robot control is quite limited because it is implicitly assumed that the action is discrete.

To overcome the problems raised by the study of CLIS, this paper proposes Cooperative and competitive Reinforcement And Imitation Learning (CRAIL), which extends CLIS to stabilize learning processes and improve sampling efficiency. Similar to CLIS, CRAIL maintains a set of multiple heterogeneous policies, including hand-coded controllers, and collects samples by a behavior policy constructed by the mixture distribution of the policies. Because the mixing weights are computed by the performance of the module, a better policy is automatically selected based on the learning progress. Then all the modules are trained simultaneously by two objective functions. CRAIL introduces the following two components to CLIS: (1) multiple importance sampling, and (2) policy learning using a combination of temporal difference and behavior cloning loss. Using multiple importance sampling stabilizes the learning process of the policy search methods because the correction factor, which is called the importance-sampling ratio, is upper-bounded. One critical contribution of CRAIL is its introduction of behavior cloning loss as well as temporal difference learning. Based on the learning processes of multiple modules, CRAIL dynamically updates the behavior policy that will be used as the best expert policy. Unlike learning from demonstrations, we can explicitly compute the behavior cloning loss based on a behavior policy, which significantly improves the policy updates. Furthermore, we use modern reinforcement learning algorithms such as entropy-regularized RL because of several advantages described later.

We compare CRAIL with CLIS on four benchmark control tasks supported by the OpenAI gym (Brockman et al., 2016). Experimental results indicate that by exploiting task-relevant episodes generated by suboptimal, but fast-learning modules a complex learning module trained with CRAIL actually learns faster than when it is trained alone. Due to adding the behavior cloning loss, CRAIL learns much faster than CLIS on all the benchmark tasks. In addition, CRAIL effectively transfers samples collected by the fixed hand-coded controller to train policies implemented by neural networks.

2. RELATED WORK

Several reinforcement learning methods with multiple modules have been proposed. Compositional Q-learning (Singh, 1992) selects a learning module with the least TD-error, and Selected Expert Reinforcement Learner (Ring and Schaul, 2011) extends the value function to select a module with better performance. Doya et al. (2002) proposed Multiple Model-based Reinforcement Learning (MMRL), in which each module is comprised of a state prediction model and the module with the least prediction error is selected and trained. These approaches are interpreted as the concept of "Mixture of Experts." In these approaches, the structure of each module is the same and uses the same learning algorithm, while CRAIL enables the use of heterogeneous learning modules that can be trained concurrently. One interpretation is that the modules are spatially distributed in their methods because they change the module based on the current environmental state. On the other hand, CRAIL temporarily distributes the modules because it switches them due to the learning progress.

Some researchers integrated an RL algorithm with hand-coded policies to improve the learning progress in its initial stage. Smart and Kaelbling (2002) proposed an architecture comprised of a supplied control policy and Q-learning. In the first learning phase, a robot was controlled with the supplied control policy developed by a designer. The second learning phase begins to control the robot effectively when the value function is approximated sufficiently. Xie et al. (2018) proposed a similar approach to incorporate *a prior* knowledge, in which Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016) and a PID controller are used as off-policy learning and a hand-coded policy, respectively. However, a limitation of their approach is that it uses only one learning module. CRAIL is a more general architecture for incorporating multiple prior knowledge. In addition, it can automatically select an appropriate

module depending on the learning progress. Sutton et al. (1999) described the advantages of off-policy learning and proposed a novel framework to accelerate learning by representing policies at multiple levels of temporal abstraction. Although their method assumed a semi-Markov decision problem and AVRL, CLIS can use different learning algorithms.

Our framework can be interpreted as learning from demonstrations. Many previous studies can be found in this field, and some recent studies such as (Gao et al., 2018; Hester et al., 2018; Nair et al., 2018) integrated reinforcement learning with learning from demonstrations by augmenting the objective function. Our framework resembles those methods from the viewpoint of the design of the objective function. The role of the demonstrator is different because our framework's demonstrator is selected from multiple heterogeneous policies based on the learning progress; previous studies assumed that it is stationary and used it to generate a training dataset. Since CRAIL explicitly represents the behavior policy, actions can be easily sampled from it to evaluate the behavior cloning loss.

The most closely related study is Mix & Match (Czarnecki et al., 2018), in which multiple heterogeneous modules are trained in parallel. Mix & Match's basic idea resembles CRAIL, but it does not consider multiple reinforcement learning algorithms; CRAIL adopts three learning algorithms for every module. In addition, Mix & Match uses a mixture of policies and optimizes the mixing weights by a kind of evolutionary computation. Since Mix & Match needs multiple simulators, it is sample-inefficient. The mixing weights are automatically determined in the case of CRAIL.

3. METHODS

3.1. CRAIL's Architecture

We investigate the standard Markov Decision Process (MDP) framework, which is not known by an agent in the model-free RL setting (Sutton and Barto, 1998). An MDP is formulated as follows: (1) $\mathcal{X}$ is the state space and $\boldsymbol{x}_t \in \mathcal{X}$ denotes the state of the environment at time t; (2) $\mathcal{U}$ is the action space and $\boldsymbol{u}_t \in \mathcal{U}$ is the action executed by the agent at time t; (3) $p_e(\boldsymbol{x}' \mid \boldsymbol{x}, \boldsymbol{u})$ is the state transition probability for $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$ and $\boldsymbol{u} \in \mathcal{U}$; (4) $p_0(\boldsymbol{x})$ is the initial state probability; and (5) $r(\boldsymbol{x}, \boldsymbol{u})$ is a reward function. CRAIL has M learning modules as shown in **Figure 1**, and each of which has state value function $V_i(\boldsymbol{x}; \boldsymbol{\psi}_i)$, state-action value function $Q_i(\boldsymbol{x}, \boldsymbol{u}; \boldsymbol{\theta}_i)$, and policy $\pi_i(\boldsymbol{u} \mid \boldsymbol{x}; \boldsymbol{\phi}_i)$, where $\boldsymbol{\psi}_i, \boldsymbol{\theta}_i$, and $\boldsymbol{\phi}_i$ are the parameters, respectively. V_i and Q_i are defined as a discounted sum of the rewards given by

$$V_i(\boldsymbol{x}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{x}_t, \boldsymbol{u}_t) \,\middle|\, \boldsymbol{x}_0 = \boldsymbol{x}\right],$$

$$Q_i(\boldsymbol{x}, \boldsymbol{u}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{x}_t, \boldsymbol{u}_t) \,\middle|\, \boldsymbol{x}_0 = \boldsymbol{x}, \boldsymbol{u}_0 = \boldsymbol{u}\right],$$

where $\gamma \in [0, 1)$ is a discount factor that determines the relative weighting of immediate versus later rewards. For simplicity, all the modules share the same sensory-motor system.

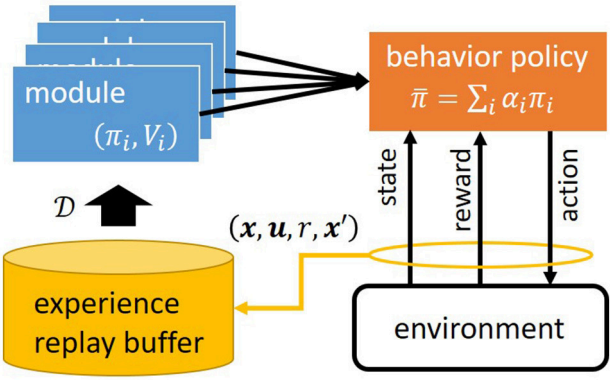

FIGURE 1 | Architecture of Cooperative and Competitive Reinforcement And Imitation Learning (CRAIL).

Algorithm 1 Stepwise CRAIL

Initialize all parameters of the learning modules.
Initialize empty replay buffer $\mathcal{D}$.
repeat
  $\boldsymbol{x}_0 \sim p_0(\cdot)$ ▷ Draw an initial state.
  for $t = 0, \ldots, T-1$ **do**
    $\boldsymbol{u}_t \sim \bar{\pi}(\cdot \mid \boldsymbol{x}_t), \boldsymbol{x}_{t+1}, r_t \sim p_e(\cdot, \cdot \mid \boldsymbol{x}_t, \boldsymbol{u}_t)$.
    Add batch data $\{\boldsymbol{x}_t, \boldsymbol{u}_t, r_t, \boldsymbol{x}_{t+1}\}$ to replay buffer $\mathcal{D}$.
    for $i = 1, \ldots, M$ **do** ▷ Update all the modules.
      update the parameters by **Algorithms 4** or **5**.
    end for
  end for
until convergence

At each time step t, the agent selects an action based on the following behavior policy:

$$\bar{\pi}(\boldsymbol{u}_t \mid \boldsymbol{x}_t) = \sum_{i=1}^{M} \alpha(i \mid \boldsymbol{x}_t) \pi_i(\boldsymbol{u}_t \mid \boldsymbol{x}_t; \boldsymbol{\phi}_i). \tag{1}$$

Because the state value function evaluates the policy's performance, we use it to determine the mixing weight:

$$\alpha(i \mid \boldsymbol{x}_t) = \frac{\exp(\beta V_i(\boldsymbol{x}_t; \boldsymbol{\psi}_i))}{\sum_{j=1}^{M} \exp(\beta V_i(\boldsymbol{x}_t, \boldsymbol{\psi}_j))}, \tag{2}$$

where β is an inverse temperature. A low β value causes (most of) the equiprobable selection of all the modules, while its high value causes the selection of a module with the highest value when the probability comes closest to one. Inverse temperature β plays an important role at the early stage of learning concerning whether to select optimistic modules that may have large initial values. **Algorithm 1** illustrates an overview of the learning process of stepwise CRAIL. The agent maintains experience replay buffer $\mathcal{D}$ to store state transition $(\boldsymbol{x}, \boldsymbol{u}, r, \boldsymbol{x}')$ by behavior policy $\bar{\pi}$.

As a special case for episodic tasks, we focus on episodic CRAIL, which is basically identical to the original CLIS, as shown

Algorithm 2 Episodic CRAIL/CLIS

Initialize all parameters of the learning modules.
Initialize empty replay buffer $\mathcal{D}$.
repeat
  for $k = 1, \ldots, K$ **do** ▷ Collect K episodes
    $\boldsymbol{x}_0 \sim p_0(\cdot)$ ▷ Draw an initial state.
    $i \sim \alpha(\cdot \mid \boldsymbol{x}_0)$ ▷ Select a module.
    for $t = 0, \ldots, T-1$ **do**
      $\boldsymbol{u}_t \sim \pi_i(\cdot \mid \boldsymbol{x}_t), \boldsymbol{x}_{t+1}, r_t \sim p_e(\cdot, \cdot \mid \boldsymbol{x}_t, \boldsymbol{u}_t)$.
    end for
    Add batch data $\{i, \boldsymbol{x}_{0:T}, \boldsymbol{u}_{0:T-1}, r_{0:T-1}\}$ to replay buffer $\mathcal{D}$.
    for $i = 1, \ldots, M$ **do** ▷ Update all the modules.
      update the parameters by **Algorithms 3**, **4**, or **5**.
    end for
  end for
until convergence

in **Algorithm 2**. At the beginning of every episode, a module is chosen by Equation (2) to generate a sequence of states, actions, and rewards denoted by

$$h \triangleq [\boldsymbol{x}_1, \boldsymbol{u}_1, r_1, \ldots, \boldsymbol{x}_T, \boldsymbol{u}_T, r_T],$$

where T denotes the number of steps called the horizon length. This modification is useful from the viewpoint of numerical stability when a hand-coded deterministic policy is used as domain knowledge. For example, a Central Pattern Generator (CPG) is widely used to generate rhythmic motions like walking without rhythmic sensory inputs (Ijspeert, 2008), but it cannot be represented by policy $\pi_i(\boldsymbol{u} \mid \boldsymbol{x})$ because CPG has internal states that are not observable by other modules. In this case, the module has to cope with partially observable MDP tasks if the experiences generated by the CPG-based controller are used for training.

3.2. Learning Algorithm in Each Module

Similar to CLIS, all the modules learn an optimal policy in parallel on the samples from $\mathcal{D}$ collected by the behavior policy. The learning algorithms used by CRAIL should be able to learn from the experiences gathered by other modules, and therefore, we adopt the following three methods as an off-policy RL algorithm: REINFORCE (Williams, 1992), Soft Actor-Critic (Soft AC) (Haarnoja et al., 2018), and Deterministic Policy Gradient (DPG) (Lillicrap et al., 2016). We modify these algorithms by incorporating behavior loss to update the policy to improve their learning efficiency.

3.2.1. REINFORCE With Importance Sampling

Policy search methods that do not rely on the Bellman optimality equation such as REINFORCE (Williams, 1992) have been reevaluated because of their simplicity and robust performance with non-Markovian tasks (Meuleau et al., 1999). REINFORCE is essentially an on-policy method (Sutton and Barto, 1998) because it estimates the gradient at a particular point in the policy space by acting precisely in the manner of its corresponding policy during learning trials. To use samples collected by the behavior policy, we introduce importance sampling to the REINFORCE algorithm (Meuleau et al., 2001) as an off-policy learning algorithm. Note that REINFORCE is applicable for the episodic CRAIL because it requires a set of sequences as a dataset.

REINFORCE evaluates sequence h by

$$J_i^{\pi}(\boldsymbol{\phi}_i, h) = R(h) = \sum_{t=1}^{T} \gamma^{t-1} r_t,$$

where $R(h)$ is called the return, which is defined as the discounted sum of rewards along h. To update $\boldsymbol{\phi}_i$, REINFORCE adopts the stochastic gradient ascent method with the gradient given by

$$\frac{\partial J_i^{\pi}(\boldsymbol{\phi}_i, h)}{\partial \boldsymbol{\phi}_i} = (R(h) - b)\rho_i(h) \sum_{t=1}^{T} \frac{\partial \ln \pi_i(\boldsymbol{u}_t \mid \boldsymbol{x}_t)}{\partial \boldsymbol{\phi}_i}, \tag{3}$$

where b is a baseline parameter for variance reduction and $\rho_i(h)$ is the importance-sampling weight ratio to account for the change in the distribution, defined by

$$\rho_i(h) = \prod_{t=1}^{T} \rho_i(\boldsymbol{x}_t, \boldsymbol{u}_t) = \prod_{t=1}^{T} \frac{\pi_i(\boldsymbol{u}_t \mid \boldsymbol{x}_t)}{\bar{\pi}(\boldsymbol{u}_t \mid \boldsymbol{x}_t)}, \tag{4}$$

under the Markovian assumption. Unlike CLIS, CRAIL uses multiple importance sampling in which the denominator in (4) is the mixture distribution (1) and therefore ρ_i is upper-bounded. Note that Equation (3) is slightly different from the standard expression because the expected value with respect to all possible sequences should be considered to exploit the baseline and importance sampling. We will take expectations later to clarify how the gradient of our method is different from the original one.

Although the gradient estimator (3) is sample-efficient, it is close to zero when π_i is far from $\bar{\pi}$. This situation is often observed at the early stage of learning. To overcome this problem, we introduce the following additional objective function given by the KL divergence between the learning and behavior policies:

$$J_i^{\mathrm{BC}}(\boldsymbol{\phi}_i, \boldsymbol{x}_t) = D_{\mathrm{KL}}(\bar{\pi}(\cdot \mid \boldsymbol{x}_t) \parallel \pi_i(\cdot \mid \boldsymbol{x}_t)). \tag{5}$$

Minimizing (5) is behavior cloning, which is also known as supervised imitation learning. However, our method is more computationally efficient because we can draw samples from $\bar{\pi}$ without interacting through the environment. Consequently, the gradient to train the policy parameter is given by

$$\frac{\partial J_i^{\pi}(\boldsymbol{\phi}_i)}{\partial \boldsymbol{\phi}_i} = \mathbb{E}_{h \sim \mathcal{D}}\left[\frac{\partial J_i^{\pi,\mathrm{RL}}(\boldsymbol{\phi}_i, \cdot)}{\partial \boldsymbol{\phi}_i}\right] - \eta \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}}\left[\frac{\partial J_i^{\pi,\mathrm{BC}}(\boldsymbol{\phi}_i, \cdot)}{\partial \boldsymbol{\phi}_i}\right], \tag{6}$$

where η is a positive meta-parameter. When $\eta = 0$, Equation (6) is identical to the original gradient estimator of REINFORCE with importance sampling.

Finally, state value function $V_i(\boldsymbol{x}, \boldsymbol{\psi}_i)$ is also trained with the Monte Carlo method because it is used to construct the behavior

Algorithm 3 REINFORCE with importance sampling and Imitation Learning

Require: dataset $\mathcal{D}$

1: Sample a random minibatch of sequences h from $\mathcal{D}$.
2: Evaluate gradient $\partial J_i^{\pi,\mathrm{RL}}/\partial \boldsymbol{\phi}_i$.
3: Sample a random minibatch of states $\boldsymbol{x}$ from $\mathcal{D}$ and $\boldsymbol{u}$ from $\bar{\pi}$, respectively.
4: Evaluate gradient $\partial J_i^{\pi,\mathrm{BC}}/\partial \boldsymbol{\phi}_i$.
5: Update $\boldsymbol{\phi}_i$ by the stochastic gradient ascent method with Equation (6).
6: Update $\boldsymbol{\psi}_i$ by minimizing Equation (7).

policy. When the number of sequences in $\mathcal{D}$ is denoted by K, the loss function to optimize the state value function is given by

$$J_i^V(\boldsymbol{\psi}_i) = \frac{1}{2}\sum_{k=1}^{K}\sum_{t=1}^{T}\left(V_i(\boldsymbol{x}_t^k) - Y_t^k\right)^2, \tag{7}$$

where Y_t^k is the target value defined as

$$Y_t^k = \prod_{t'=t}^{T}\rho(\boldsymbol{x}_t^k, \boldsymbol{u}_t^k)\sum_{t'=t}^{T}\gamma^{t'-t}r_t^k.$$

The update rule of the modified REINFORCE with importance sampling is given in **Algorithm** (**3**).

3.2.2. Soft Actor-Critic and Imitation Learning

The original CLIS adopted SARSA (Rummery and Niranjan, 1994) with importance sampling (Precup et al., 2001) as an off-policy value-based reinforcement learning algorithm. An advantage is that the technique called eligibility traces (Sutton and Barto, 1998) can be used to accelerate the speed of learning, and it was experimentally shown that deep SARSA can achieve a comparable performance to DQN even though it does not exploit the method of experience replay and target network (Elfwing et al., 2018). However, SARSA implicitly assumes that action is discrete because the stochastic policy must be derived from the state-action value function. Since we are interested in robot control, action must be continuous. Therefore, we adopt Soft Actor-Critic (Haarnoja et al., 2018) as an off-policy algorithm using the value function. Soft Actor-Critic augments the reward function to replace the max-operator with a differentiable one. The reward function is assumed to be given by the following form:

$$\tilde{r}(\boldsymbol{x}, \boldsymbol{u}) = r(\boldsymbol{x}, \boldsymbol{u}) + \frac{1}{\alpha}\mathcal{H}(\pi_i(\cdot \mid \boldsymbol{x})), \tag{8}$$

where α is a positive meta-parameter and and $\mathcal{H}(\pi(\cdot \mid \boldsymbol{x}))$ is the (differential) entropy of policy π_i. Assuming reward function (8), an optimal state value function satisfies the following Bellman optimality equation:

$$V_i(\boldsymbol{x}) = \max_{\pi_i}\mathbb{E}_{\pi_i}\left[r(\boldsymbol{x}, \boldsymbol{u}) - \frac{1}{\alpha}\ln\pi_i(\boldsymbol{u} \mid \boldsymbol{x}) + \gamma\mathbb{E}_{P_T}\left[V_i(\boldsymbol{x}')\right]\right]. \tag{9}$$

The right hand side of Equation (9) is a constrained optimization problem given by

$$\max_{\pi_i}\int \mathrm{d}\boldsymbol{u}\pi_i(\boldsymbol{u} \mid \boldsymbol{x})\left[r(\boldsymbol{x}, \boldsymbol{u}) - \frac{1}{\alpha}\ln\pi_i(\boldsymbol{u} \mid \boldsymbol{x}) + \gamma\mathbb{E}_{P_T}\left[V_i(\boldsymbol{x}')\right]\right],$$

subject to $\int \mathrm{d}\boldsymbol{u}\pi_i(\boldsymbol{u} \mid \boldsymbol{x}) = 1$. In this case, we can analytically maximize the right hand side of Equation (9) by a method with Lagrange multipliers. Consequently, the optimal state value function can be represented by

$$V_i(\boldsymbol{x}) = \frac{1}{\alpha}\ln\int \mathrm{d}\boldsymbol{u}\left[\exp(\alpha Q_i(\boldsymbol{x}, \boldsymbol{u}))\right], \tag{10}$$

and the corresponding optimal policy can be derived:

$$\pi_i(\boldsymbol{u} \mid \boldsymbol{x}) = \frac{\exp\left(\alpha Q_i(\boldsymbol{x}, \boldsymbol{u})\right)}{\exp(\alpha V_i(\boldsymbol{x}))}, \tag{11}$$

where state-action value function $Q(\boldsymbol{x}, \boldsymbol{u})$ is defined by

$$Q_i(\boldsymbol{x}, \boldsymbol{u}) = r(\boldsymbol{x}, \boldsymbol{u}) + \gamma\mathbb{E}_{P_T}\left[V_i(\boldsymbol{x}')\right]. \tag{12}$$

Note that the right hand side of Equation (10) uses the log-sum-exp operator if the action is discrete, and it is characterized as the "soft" max operator.

The learning algorithm of the Soft Actor-Critic is derived from Equations (10)–(12). Since Equation (12) corresponds to the Bellman optimality equation regarding the state-action value function, it can be used to train parameter $\boldsymbol{\theta}_i$ by minimizing the soft Bellman residual for all possible $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$ in buffer $\mathcal{D}$:

$$J_i^Q(\boldsymbol{\theta}_i, \boldsymbol{x}, \boldsymbol{u}, r, \boldsymbol{x}') = \frac{1}{2}\left\{Q_i(\boldsymbol{x}, \boldsymbol{u}) - \left(r + \gamma V_i(\boldsymbol{x}'; \bar{\boldsymbol{\psi}}_i)\right)\right\}^2,$$

where $V_i(\boldsymbol{x}, \bar{\boldsymbol{\psi}}_i)$ and $\bar{\boldsymbol{\psi}}_i$ respectively denote the target state value network and an exponentially moving average of the parameter vector, which stabilizes the learning used in DQN (Mnih et al., 2015). Consequently, the loss function for training $\boldsymbol{\theta}_i$ is given by

$$J_i^Q(\boldsymbol{\theta}_i) = \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{u}, r, \boldsymbol{x}')\sim\mathcal{D}}\left[J_i^Q(\boldsymbol{\theta}_i, \cdot, \cdot, \cdot, \cdot)\right], \tag{13}$$

where $(\boldsymbol{x}, \boldsymbol{u}, r, \boldsymbol{x}') \sim \mathcal{D}$ means that the transition data are drawn from $\mathcal{D}$.

When the action is discrete, the optimal policy and the state value function can be easily computed from the state-action value function. However, it is intractable in the case of continuous action because Equation (10) needs to evaluate the integral in action space. Therefore, Haarnoja et al. (2018) recommended that the state value function and policy also be separately approximated. Based on the relation (11), the approximation error of the state value function at state $\boldsymbol{x}$ is given by

$$J_i^V(\boldsymbol{\psi}_i, \boldsymbol{x}) = \frac{1}{2}\left\{V_i(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{u}\sim\pi_i}\left[Q_i(\boldsymbol{x}, \cdot) - \frac{1}{\alpha}\ln\pi(\cdot \mid \boldsymbol{x})\right]\right\}^2,$$

where the expectation is numerically computed through a Monte Carlo simulation. The loss function for training $\boldsymbol{\psi}_i$ is given by

$$J_i^V(\boldsymbol{\psi}_i) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[J_i^V(\boldsymbol{\psi}_i, \cdot)\right]. \tag{14}$$

Algorithm 4 Soft Actor-Critic and Imitation Learning

Require: dataset $\mathcal{D}$, inverse temperature η, decay rate τ.

1: Sample a random minibatch of transitions $(\boldsymbol{x}, \boldsymbol{u}, r, \boldsymbol{x}')$ from $\mathcal{D}$.
2: Evaluate gradient $\partial J_i^Q / \partial \boldsymbol{\theta}_i$ and update $\boldsymbol{\theta}_i$ by stochastic gradient descent.
3: Sample a random minibatch of states $\boldsymbol{x}$ from $\mathcal{D}$ and $\boldsymbol{u}$ from π_i, respectively.
4: Evaluate gradient $\partial J_i^V / \partial \boldsymbol{\psi}_i$ and update $\boldsymbol{\psi}_i$ by the stochastic gradient descent.
5: Sample a random minibatch of states $\boldsymbol{x}$ from $\mathcal{D}$ and $\boldsymbol{u}$ from $\bar{\pi}$, respectively.
6: Evaluate gradient $\partial J_i^\pi / \partial \boldsymbol{\phi}_i$ and update $\boldsymbol{\phi}_i$ by the stochastic gradient descent.
7: Update the parameter of the target network by $\bar{\boldsymbol{\psi}}_i \leftarrow \tau \bar{\boldsymbol{\psi}}_i + (1-\tau)\boldsymbol{\psi}_i$.

In the same way, policy parameter $\boldsymbol{\theta}_i$ is trained by minimizing the Kullback-Leibler (KL) divergence between the left and right hand sides of Equation (11):

$$J_i^{\pi,\mathrm{RL}}(\boldsymbol{\phi}_i, \boldsymbol{x}) = D_{\mathrm{KL}}\left(\pi_i(\cdot \mid \boldsymbol{x}) \parallel \frac{\exp(\alpha Q_i(\boldsymbol{x}, \cdot))}{\exp(\alpha V_i(\boldsymbol{x}))}\right), \tag{15}$$

where we need samples drawn from π_i to evaluate the KL divergence. In addition to the KL divergence, we introduce the behavior cloning loss defined as the KL divergence between the learning and behavior policies:

$$J_i^{\pi,\mathrm{BC}}(\boldsymbol{\phi}_i, \boldsymbol{x}) = D_{\mathrm{KL}}(\bar{\pi}(\cdot \mid \boldsymbol{x}) \parallel \pi_i(\cdot \mid \boldsymbol{x})). \tag{16}$$

Consequently, the loss function for training $\boldsymbol{\phi}_i$ is given by

$$J_i^\pi(\boldsymbol{\phi}_i) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[J_i^{\pi,\mathrm{RL}}(\boldsymbol{\phi}_i, \cdot) + \eta J_i^{\pi,\mathrm{BC}}(\boldsymbol{\phi}_i, \cdot)\right], \tag{17}$$

where η is a positive meta-parameter. When $\eta = 0$, Equation (17) is identical to the original update rule of Soft Actor-Critic. Note that Information projection (I-projection) is used in Equation (15), and Moment projection (M-projection) is used in Equation (16) (Kober et al., 2013). Although in principle we can select any projection, we believe that Equation (16) is appropriate for the behavior cloning loss because it is averaged over several modes of the policy. In addition, Equation (15) is appropriate because it concentrates on a single mode. π_i is usually implemented by a Gaussian policy with a single mode, but $\exp(\alpha Q_i(\boldsymbol{x}, \cdot))/\exp(\alpha V_i(\boldsymbol{x}))$ may have multiple modes. The update rule of the modified Soft Actor-Critic is given by **Algorithm 4**.

3.2.3. Deterministic Policy Gradient and Imitation Learning

Deterministic Policy Gradient (DPG) (Silver et al., 2014) and its deep version (Lillicrap et al., 2016) are a well-known off-policy reinforcement learning algorithm that can handle continuous actions. Unlike Soft Actor-Critic, DPG does not approximate the state value function. The policy network can also be simplified significantly because it does not need to approximate a continuous probability density function.

Algorithm 5 Deterministic Policy Gradient and Imitation Learning

Require: dataset $\mathcal{D}$, inverse temperature η, decay rate τ.

1: Sample a random minibatch of transitions $(\boldsymbol{x}, \boldsymbol{u}, r, \boldsymbol{x}')$ from $\mathcal{D}$.
2: Evaluate gradient $\partial J_i^Q / \partial \boldsymbol{\theta}_i$ and update $\boldsymbol{\theta}_i$ by the stochastic gradient descent.
3: Sample a random minibatch of states $\boldsymbol{x}$ from $\mathcal{D}$.
4: Evaluate gradient $\partial J_i^\pi / \partial \boldsymbol{\phi}_i$ and update $\boldsymbol{\phi}_i$ by the stochastic gradient descent.
5: Update the parameter of the target network by $\bar{\boldsymbol{\theta}}_i \leftarrow \tau \bar{\boldsymbol{\theta}}_i + (1-\tau)\boldsymbol{\theta}_i$.

The loss function to train Q_i in DPG resembles that in Soft Actor-Critic and is given by Equation (13) whose $J_i^Q(\boldsymbol{\theta}_i, \boldsymbol{x}, \boldsymbol{u}, r, \boldsymbol{x}')$ is replaced with the following equation:

$$J_i^Q(\boldsymbol{\theta}_i, \boldsymbol{x}, \boldsymbol{u}, r, \boldsymbol{x}') = \frac{1}{2}\left\{Q_i(\boldsymbol{x}, \boldsymbol{u}) - (r + \gamma Q_i(\boldsymbol{x}', \pi_i(\boldsymbol{x}'); \bar{\boldsymbol{\theta}}_i))\right\}^2,$$

where $\bar{\boldsymbol{\theta}}_i$ denotes an exponentially moving average of the parameter vector of the target state-action value network and π_i is a deterministic policy that maps $\boldsymbol{x}$ to $\boldsymbol{u}$. DPG evaluates the policy gradient at state $\boldsymbol{x}$ by

$$\frac{\partial J_i^{\pi,\mathrm{RL}}(\boldsymbol{\phi}_i, \boldsymbol{x})}{\partial \boldsymbol{\phi}_i} = \left.\frac{\partial Q_i(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{u}}\right|_{\boldsymbol{u}=\pi_i(\boldsymbol{x})} \frac{\partial \pi_i(\boldsymbol{x})}{\partial \boldsymbol{\phi}_i}.$$

As a result, the policy gradient with behavior cloning loss is computed by

$$\frac{\partial J_i^\pi}{\partial \boldsymbol{\phi}_i} = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\frac{\partial J_i^{\pi,\mathrm{RL}}(\boldsymbol{\phi}_i, \cdot)}{\partial \boldsymbol{\phi}_i} - \eta \frac{\partial J_i^{\pi,\mathrm{BC}}(\boldsymbol{\phi}_i, \cdot)}{\partial \boldsymbol{\phi}_i}\right],$$

where $J_i^{\pi,\mathrm{BC}}$ is the same function used by the modified Soft Actor-Critic explained in section 3.2.3. The state value function is simply computed by

$$V_i(\boldsymbol{x}) = Q_i(\boldsymbol{x}, \pi_i(\boldsymbol{x})).$$

The update rule of the modified DPG is given by **Algorithm 5**. One limitation of DPG is that it does not have an explicit exploration mechanism because policy π_i represents a deterministic function. Therefore, DPG usually introduces a behavior policy that is implemented by an Ornstein-Uhlenbech process (Lillicrap et al., 2016). On the other hand, CRAIL's behavior policy is dynamically constructed by mixing all of the component policies. When DPG is selected as a learning algorithm of CRAIL, at least one learning module with a stochastic policy should be added to promote exploration and discourage premature convergence.

4. EXPERIMENTS

4.1. Comparison of CRAIL and CLIS

To investigate how CRAIL improves the learning speed, we conducted several computer simulations with four MuJoCo-simulated (Todorov et al., 2012) benchmark tasks: Hopper-v2, Half-Cheetah-v2, Walker2d-v2, and Ant-v2, all of which were provided by the OpenAI gym (Brockman et al., 2016) (**Figure 2**). Hopper-v2 is a planar monopod, and Walker2d-v2 and HalfCheetah-v2 are planar biped robots. Ant-v2 is a quadruped robot that can move around a three-dimensional environment. The observation and action spaces are shown in **Table 1**, where the observation vector is used as a state vector. The goal is to move forward as quickly as possible, and the reward function is given by $r(\boldsymbol{x}, \boldsymbol{u}) = v_x - c\|\boldsymbol{u}\|_2^2$, where v_x is the forward velocity and c is a robot-dependent constant. See the supplementary materials of Duan et al. (2016) for the task specifications.

We prepared two function approximators, Neural Network (NN) and normalized Radial Basis Function (RBF), and **Table 2** shows their network architectures. For example, the module using the RBF networks represents V_i by 64 normalized radial basis functions by

$$V_i(\boldsymbol{x}; \boldsymbol{\psi}_i) = \sum_{j=1}^{N_i} \psi_{i,j} b_{i,j}(\boldsymbol{x}),$$

where N_i and $\psi_{i,j}$ respectively denote the number of basis functions and the j-th element of $\boldsymbol{\psi}_i$ and $b_{i,j}(\boldsymbol{x})$ is the basis function defined by

$$b_{i,j}(\boldsymbol{x}) = \frac{a_{i,j}(\boldsymbol{x})}{\sum_{j'=1}^{N_i} a_{i,j'}(\boldsymbol{x})}, \quad a_{i,j} = \exp\left(-\|\boldsymbol{s}_{i,j}^{\top}(\boldsymbol{x} - \boldsymbol{c}_{i,j})\|_2^2\right),$$

where $a_{i,j}$ is a Gaussian activation function with parameters $\boldsymbol{s}_{i,j}$ and $\boldsymbol{c}_{i,j}$. Since $\boldsymbol{s}_{i,j}$ and $\boldsymbol{c}_{i,j}$ were determined by a heuristic

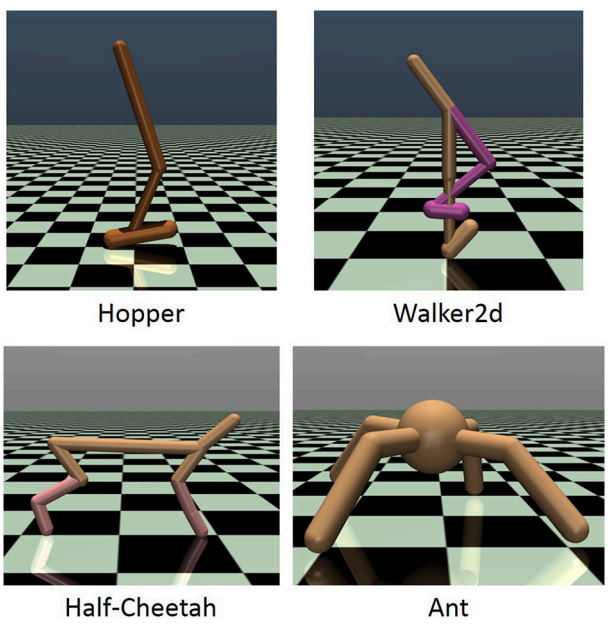

FIGURE 2 | MuJoCo-simulated environments: Hopper-v2, Walker2D-v2, Half-Cheetah-v2, and Ant-v2.

TABLE 1 | Environments used in experiments and their state and action spaces.

Environment	Observation space	Action space
Ant-v2	$\mathbb{R}^{111}$	$[-1.0, 1.0]^8$
HalfCheetah-v2	$\mathbb{R}^{17}$	$[-1.0, 1.0]^6$
Hopper-v2	$\mathbb{R}^{11}$	$[-1.0, 1.0]^3$
Walker2d-v2	$\mathbb{R}^{17}$	$[-1.0, 1.0]^6$

TABLE 2 | Network architectures of approximator in the first and the second experiments: For example, RBF module approximates Q_i by 64 basis functions, and NN module approximates two-layer feed-forward neural network consisting of (400, 300) hidden units.

Approximator	V	Q	π
RBF	(64)	(64)	(64)
NN	(64, 64)	(400, 300)	(400, 300)

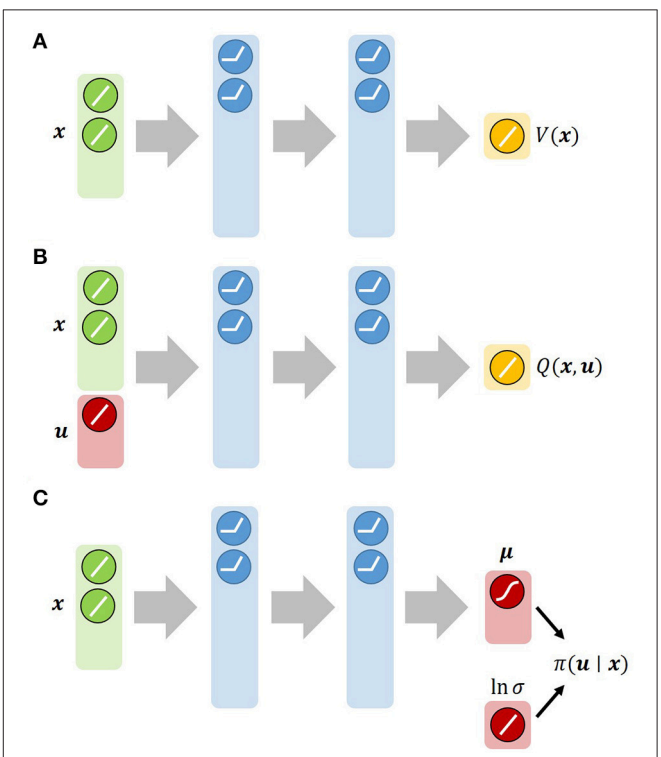

FIGURE 3 | Architectures of neural networks used by Soft Actor-Critic: **(A)** State value function network. **(B)** State-action value function network. **(C)** Gaussian-policy network. We approximate both V and Q with feed-forward neural networks. π is approximated by a Gaussian policy: $\pi(\boldsymbol{u} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{u} \mid \boldsymbol{\mu}, \sigma^2 \boldsymbol{I})$, where the mean $\boldsymbol{\mu}$ is given by a neural network and the log-standard deviation $\ln \sigma$ is parameterized by a global vector independent of the state.

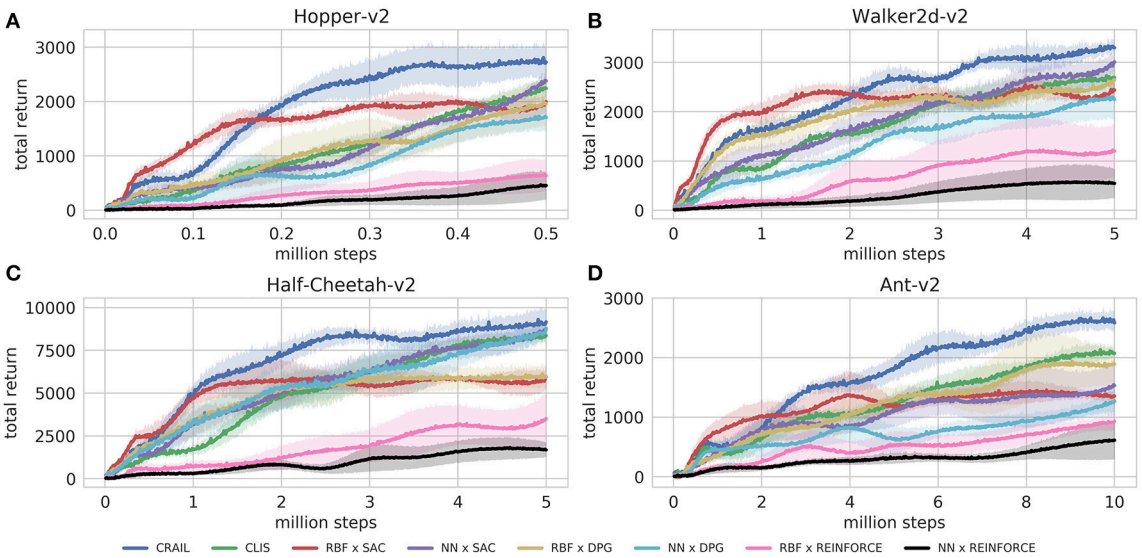

FIGURE 4 | Training curves on continuous control benchmarks: Performance was evaluated by cumulative rewards in 10 episodes for each learning module.

rule (Morimoto and Doya, 2001), V_i is interpreted as a linear neural network. Therefore, the module with the RBF networks is expected to learn faster than that with the nonlinear neural networks. **Figure 3** represents the architectures that approximate π_i, V_i and Q_i needed by the Soft Actor-Critic. Each was implemented by a feed-forward neural network with a Rectified Linear Unit (ReLU) as a nonlinear activation function of the hidden layers. In the first experiment, we chose three learning algorithms, Soft Actor-Critic, Deterministic Policy Gradient, and REINFORCE with importance sampling. We prepared $2 \times 3 = 6$ modules as a result. To apply **Algorithm 3** to this non-episodic task, the horizon length T is set to 300.

CRAIL was given the above six modules for parallel training. We also tested the six modules separately in addition to CLIS as baseline performances, where CLIS also used multiple importance sampling instead of an independent type because the original CLIS worked very poorly due to the unboundedness of the importance-sampling weight ratio. Note that the original CLIS selects one learning module at the beginning of each episode, and utilizes a truncated importance sampling ratio given by

$$\hat{\rho}_i(h) = \min\left(\prod_{t=1}^{T} \frac{\pi_i(\boldsymbol{u}_t \mid \boldsymbol{x}_t)}{\pi_{\text{selected}}(\boldsymbol{u}_t \mid \boldsymbol{x}_t)}, C\right),$$

where π_{selected} is the policy of the selected module and C is a positive constant determined by the experimenters. Although $\hat{\rho}_i(h)$ is upper-bounded, it is not trivial to tune C in practice. In addition, CLIS does not consider behavior cloning loss. Therefore, CLIS evaluated in the experiments uses Equation (4) as the importance weight. In this case, CLIS is identical to CRAIL with $\eta = 0$. Each method was evaluated in ten simulation runs, each of which was comprised of 2,000 episodes.

Figure 4 shows the learning performance of CRAIL, CLIS, and the six component modules, and we found that CRAIL learned faster than CLIS and the six modules trained separately on all the benchmark tasks. On the other hand, the learning performance of CLIS resembled that of the NN × SAC module. The RBF × SAC module showed the best learning curves on all the tasks at the early stage of learning, but its performance saturated before reaching a sufficient level because the normalized RBF networks could not precisely approximate the value functions and the policy as well as the neural networks. On the contrary, the NN policies trained by SAC or DPG learned very slowly, and their performance was much worse than RBF × SAC at the early stage of learning. The modules trained by REINFORCE needs a set of sequences, and therefore, they learned slower than the actor-critic methods such as DPG and Soft AC. As a result, the REINFORCE modules achieved worse performance, and the probabilities remained low during learning. **Figures 5A,B** respectively show the mixing weights $\{\alpha_i\}_{i=1}^{6}$ during the learning of Ant-v2 computed by CRAIL and CLIS. The probability of selecting the RBF × SAC module increased rapidly at the early stage of learning in both cases. However, CRAIL tended to gradually select the NN × SAC module after about four million steps, and CLIS continued to choose the RBF × SAC module's policy most frequently until about six million steps.

4.2. Adaptation to Changes in the Environment

Next, we experimentally tested the capability of adaptation to changes in the environment by changing the mass of the body of HalfCheetah-v2 from 6.36 (original) to 6.36 × 3 [kg] at the 5 millionth step. In this experiment, both CRAIL and CLIS possessed the same six learning modules used in the previous experiment. Each method was evaluated in ten simulation runs, each of which was comprised of 2,000 episodes.

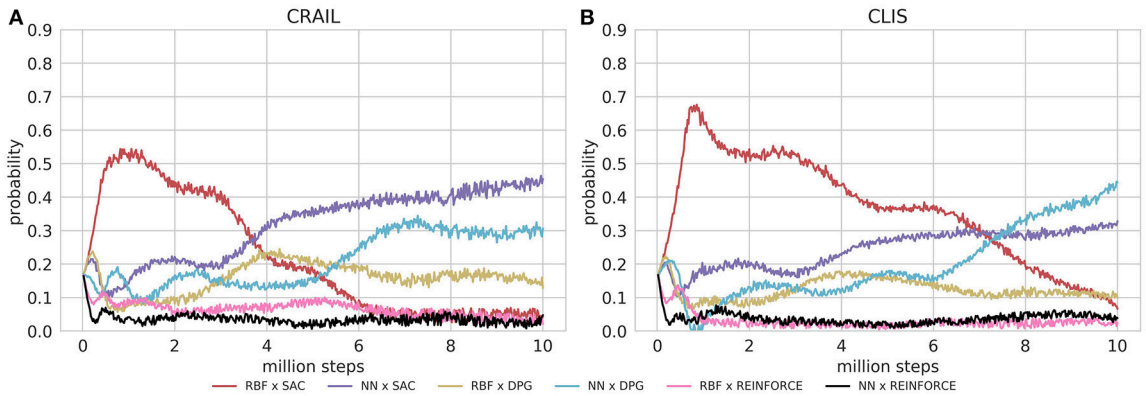

FIGURE 5 | Probabilities for changing learning modules during learning process: **(A)** Results obtained by CLIS architecture. **(B)** Results obtained by CL without importance sampling.

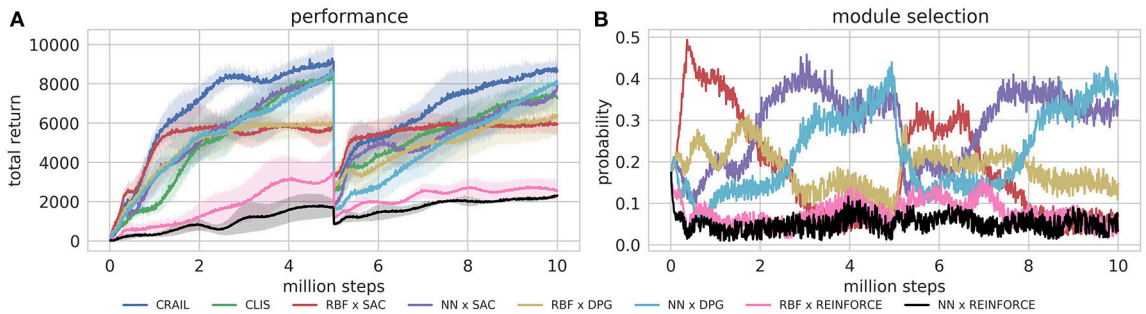

FIGURE 6 | Training curves on episodic Half-Cheetah-v2 task, in which body's mass was changed at 5 millionth step.

Figures 6A,B respectively show the cumulative rewards and module selection probability in each step. Note that the first half of **Figure 6A** is identical to **Figure 4C**. When the mass was changed at 5 millionth steps, the performance of the CRAIL, CLIS, and NN policies decreased significantly. However, the RBF policies maintained the pole without considerable deterioration in performance compared with the NN policies because the number of weights was smaller. In other words, the performances of the NN policies deteriorated drastically because their policies were fine-tuned for a particular weight. Therefore, the probability of selecting RBF × SAC increased temporarily from about 5 to 6.5 million steps. CRAIL prevented the body from falling and trained NN × SAC and NN × DPG by appropriately selecting RBF × SAC, as shown in **Figure 6B**.

4.3. Introducing a Fixed Policy

To investigate how CRAIL exploits a deterministic stationary policy, we added a CPG-based policy as prior knowledge to control HalfCheetah-v2 because periodic motion is quite useful to generate walking behaviors and many previous studies exist (Ijspeert, 2008) in this field. Since CRAIL uses multiple importance sampling, it is straightforward to use the deterministic policy as one of the sampling policies. Note that the CPG-based policy has internal states because the oscillator is implemented by a differential equation. Therefore, we selected the REINFORCE algorithm with importance sampling described in section 3.2.1 and **Algorithm 2** in this experiment because the evaluation of deterministic policies with internal states is difficult in stepwise update rules.

TABLE 3 | Network architectures of approximator in the third experiments: We denote the hidden layer sizes of a two-layer feedforward neural network as *(N, M)*.

Approximator	V	π
BASE	(64, 64)	(64, 64)
WIDE	(64, 64)	(400, 300)
DEEP	(64, 64)	(100, 50, 25)

For example, the WIDE module approximates V_i and π_i by (64, 64) and (400, 300), respectively.

As learning modules, we prepared three network architectures that are commonly seen in the literature (Henderson et al., 2018) as shown in **Table 3** to implement a stochastic policy. We used a ReLU nonlinear activation function. Note that the REINFORCE algorithm does not need Q_i. In addition, a deterministic stationary policy based on central pattern generators was prepared as prior knowledge, which was implemented by the modified Hopf oscillator (Uchibe and Doya, 2014). Since CRAIL uses multiple importance sampling, it is straightforward to use the deterministic policy as one of the sampling policies.

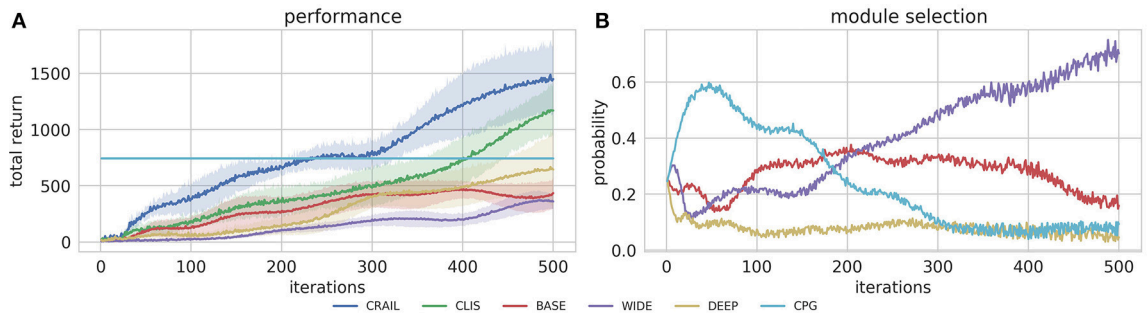

FIGURE 7 | Training curves on Half-Cheetah-v2 task, in which fixed stationary policy was used as prior knowledge.

In addition to evaluate the CRAIL's performance, we tested the four modules separately. **Figure 7A** shows that CRAIL learned much faster than the component modules trained alone. Since REINFORCE learns very slowly due to its simplicity (Duan et al., 2016), 500 iterations were insufficient to overcome the CPG-based controller. **Figure 7B** shows the mixing weights during the learning computed by CRAIL. The probability of selecting the CPG-controller module increased rapidly at the early stage of learning. Then, CRAIL tended to select the BASE module and the probability of selecting it was the highest among the NN modules from about 90 to 170 iterations. Finally, the WIDE module was frequently selected at the later stage of learning. The DEEP module trained alone achieved the highest performance among the three neural network policies. However, the probability of selecting it remained low during learning. Note that the original CLIS cannot utilize the deterministic policy because the importance weight ratio becomes infinity.

5. DISCUSSION

This paper proposed modular reinforcement learning (CRAIL), which collects task-relevant samples using multiple heterogeneous policies. One interesting feature of CRAIL is that a complex RL system can learn faster with the help of a simple RL system that cannot achieve the best performance. Experimental results also suggested that CRAIL efficiently adapted to changes in the learning conditions because it automatically selected simple modules with fewer parameters.

CRAIL implicitly assumes that state value functions are not initialized optimistically. Suppose that the reward function is always non-positive, and the state value functions are initialized to zero. In this case, some modules that are not selected by Equation (1) may have V values that are consistently higher than the selected modules. In this case, CRAIL selects the worst module if the inverse temperature is not tuned appropriately. As one possible extension to overcome this difficulty, the mixing weights are also trained by reinforcement learning in which the value functions are used as priors.

In the current implementation, since all the learning modules are prepared in advance CRAIL cannot obtain good performance if all of them are inappropriate for the given task. To design appropriate learning modules, we need to develop a mechanism to add or delete learning modules based on the selection probabilities calculated by Equation (1). If a simple learning module has a low probability for a long time, it can be replaced by a complicated module. This allows CRAIL to flexibly test heterogeneous modules without increasing computational costs. To overcome this problem, we consider an asynchronous version of the algorithms.

We did not address the effects of computational costs on the learning modules. Updating the parameters of the RBF networks was accomplished considerably faster than for the deep neural networks, but the modules with the RBF networks had to wait until the modules with deep neural networks completed their computations. In general, the sampling rate significantly affects the original performance of a robot. For example, the robot should reduce its moving speed when it uses a complex module. However, the effects of the differences in sampling rates have not been scrutinized.

One interesting future topic is the use of multiple meta-parameters. CRAIL has some meta-parameters used in an RL system, and their settings, such as the learning rate, the inverse temperature that controls the randomness in action selection, and the discount factor for future reward prediction, are crucial to perform a task successfully. A possible scenario is that when a small discount factor can be used in the initial learning process, a module with a larger discount factor can be selected as the learning progresses. We have not yet identified the tasks and situations in which different discount factors play an important role for accelerating the learning speed, but in the future we will seek good examples for this topic.

AUTHOR CONTRIBUTIONS

EU conceived, designed the research, performed the experiment, analyzed its results, and wrote the paper.

FUNDING

This work is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO) and JSPS KAKENHI Grant Numbers JP16K12504 and JP17H06042.

REFERENCES

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., et al. (2016). OpenAI Gym [preprint]. *arXiv:1606.01540*.

Czarnecki, M. W., Jayakumar, S. M., Jaderberg, M., Hasenclever, L., Teh, Y. W., Osindero, S., et al. (2018). "Mix & match - Agent curricula for reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning* (Stockholm), 1087–1095.

Doya, K., Samejima, K., Katagiri, K., and Kawato, M. (2002). Multiple model-based reinforcement learning. *Neural Comput.* 14, 1347–1369. doi: 10.1162/089976602753712972

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). "Benchmarking deep reinforcement learning for continuous control," in *Proceedings of the 33rd International Conference on Machine Learning* (New York, NY), 1329–1338.

Elfwing, S., Uchibe, E., and Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Netw.* doi: 10.1016/j.neunet.2017.12.012. [Epub ahead of print].

Gao, Y., Xu, H., Lin, J., Yu, F., Levine, S., and Darrell, T. (2018). "Reinforcement learning from imperfect demonstrations," in *ICLR 2018 Workshop* (Vancouver, BC).

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). "Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning* (Stockholm), 1861–1870.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). "Deep reinforcement learning that matters," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence* (New Orleans, LA).

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., et al. (2018). "Deep Q-learning from demonstrations," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence* (New Orleans, LA).

Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: a review. *Neural Netw.* 21, 642–653. doi: 10.1016/j.neunet.2008.03.014.

Kalyanakrishnan, S., and Stone, P. (2011). Characterizing reinforcement learning methods through parameterized learning problems. *Mach. Learn.* 84, 205–247. doi: 10.1007/s10994-011-5251-x

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: a survey. *Int. J. Robot. Res.* 32, 1238–1274. doi: 10.1177/0278364913495721

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (2016). "Continuous control with deep reinforcement learning," in *Proceedings of International Conference on Learning Representations* (San Juan).

Meuleau, N., Kim, K.-E., Kaelbling, L. P., and Cassandra, A. R. (1999). "Solving POMDPs by searching the space of finite policies," in *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence* (Stockholm), 417–426.

Meuleau, N., Peshkin, L., and Kim, K.-E. (2001). *Exploration in Gradient-Based Reinforcement Learning.* Technical report, Technical Report 2001-003, Cambridge, MA: MIT.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature* 518, 529–533. doi: 10.1038/nature14236

Morimoto, J., and Doya, K. (2001). Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robot. Auton. Syst.* 36, 37–51. doi: 10.1016/S0921-8890(01)00113-0

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). "Overcoming exploration in reinforcement learning with demonstrations," in *Proceedings of IEEE International Conference on Robotics and Automation* (Brisbane, QLD).

Precup, D., Sutton, R. S., and Dasgupta, S. (2001). "Off-policy temporal-difference learning with function approximation," in *Proceedings of the 18th International Conference on Machine Learning* (Williamstown, MA).

Ring, M., and Schaul, T. (2011). "Q-error as a selection mechanism in modular reinforcement-learning systems," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence* (Barcelona), 1452–1457.

Rummery, G., and Niranjan, M. (1994). *On-Line Q-Learning Using Connectionist Systems.* Technical report, Technical Report CUED/F-INFENG/TR 166, Engineering Department Cambridge University.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature* 529, 484–489. doi: 10.1038/nature1696

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning* (Bejing), 387–395.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of Go without human knowledge. *Nature* 550, 354–359. doi: 10.1038/nature24270

Singh, S. P. (1992). Transfer of learning by composing solution of elemental sequential tasks. *Mach. Learn.* 8, 323–340.

Smart, W. D., and Kaelbling, L. P. (2002). "Effective reinforcement learning for mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation* (Washington, DC), 3404–3410.

Sutton, R. S., and Barto, A. G. (1998). *Reinforcement Learning.* Cambridge, MA: MIT Press.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. *Artif. Intell.* 112, 181–211.

Todorov, E., Erez, T., and Tassa, Y. (2012). "MuJoCo: a physics engine for model-based control," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems* (Vilamoura), 5026–5033.

Uchibe, E., and Doya, K. (2004). "Competitive-cooperative-concurrent reinforcement learning with importance sampling," in *Proceedings of the Eighth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 8* (Los Angeles, CA), 287–296.

Uchibe, E., and Doya, K. (2005). "Reinforcement learning with multiple heterogeneous modules: a framework for developmental robot learning," in *Proceedings of the 4th IEEE International Conference on Development and Learning* (Osaka), 87–92.

Uchibe, E., and Doya, K. (2014). "Combining learned controllers to achieve new goals based on linearly solvable MDPs," in *Proceedings of the IEEE International Conference on Robotics and Automation* (Hong Kong), 5252–5259.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learn.* 8, 279–292.

Whiteson, S., and Stone, P. (2006). Evolutionary function approximation for reinforcement learning. *J. Mach. Learn. Res.* 7, 877–917. Available online at: http://www.jmlr.org/papers/v7/whiteson06a.html

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* 8, 229–256.

Xie, L., Wang, S., Rosa, S., Markham, A., and Trigoni, N. (2018). "Learning with training wheels : speeding up training with a simple controller for deep reinforcement learning," in *Proceedings of IEEE International Conference on Robotics and Automation* (Brisbane, QLD).

Conflict of Interest Statement: The author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.