Swarm-inspired Reinforcement Learning via Collaborative Inter-agent Knowledge Distillation

Zhang-Wei Hong*
National Tsing Hua University
Hsinchu, Taiwan
williamd4112@gapp.nthu.edu.tw

Prabhat Nagarajan
Preferred Networks
Tokyo, Japan
prabhat@preferred.jp

Guilherme Maeda
Preferred Networks
Tokyo, Japan
gjmaeda@preferred.jp

Abstract

Reinforcement Learning (RL) has demonstrated promising results across several sequential decision-making tasks. However, RL struggles to learn efficiently, thus limiting its pervasive application to several challenging problems. A typical RL agent learns solely from its own trial-and-error experiences, requiring many experiences to learn a successful policy. To alleviate this problem, we propose *collaborative inter-agent knowledge distillation* (CIKD). CIKD is a learning framework that uses an ensemble of RL agents to execute different policies in the environment while sharing knowledge amongst agents in the ensemble. Our experiments demonstrate that CIKD improves upon state-of-the-art RL methods in sample efficiency and performance on several challenging MuJoCo benchmark tasks. Additionally, we present an in-depth investigation on how CIKD leads to performance improvements.

1 Introduction

Reinforcement learning (RL) [45] has demonstrated impressive performance on solving sequential decision-making tasks in interactive environments (e.g. video game-playing [30] or robotic control [17]). In these tasks, the RL agent's goal is to find the optimal policy, which maximizes the expected return in the task. The agent learns this optimal policy through many trial-and-error interactions with the environment. Through these interactions, the agent explores the consequences of different decisions, in the form of a reward or punishment. Unfortunately, millions of interactions are required even to solve simple tasks. Such amounts of interaction are infeasible to acquire in real, physical tasks.

In order to learn the optimal policy, the agent performs policy improvement, which refers to the process of refining the agent's policy towards performing high-rewarding actions. Beginning with a random initial policy, an RL agent incrementally refines its policy through its experiences. However, the performance of the acquired policy is sensitive to the bias of initial policy [21]. This bias may trap the agent at a sub-optimal policy. Typically, when learning to improve a policy, an RL agent performs actions that are similar to its current estimate of the best policy. In such cases where the agent is trapped at a sub-optimal policy, it may require a large number of interactions before the agent acquires the experiences necessary for escaping its suboptimal policy. While standard self-experience-based incremental RL gives little hope to overcome this problem, collaboration amongst a number of agents with different behavioral characteristics can possibly alleviate this problem.

*This work was done during an internship at Preferred Networks.

5th Deep Reinforcement Learning Workshop at Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

For inspiration, we turn to the study of collective animal behavior [44]. Rather than exclusively learn from trial-and-error alone, the optimal behaviors for several tasks (e.g., flocking and foraging) can emerge through collaboration amongst animals. For example, consider ants foraging in a colony [8]. The ants search divergent paths, resulting in extensive exploration as a group, across a wide range of food-gathering policies. Furthermore, the ants that find food share their path with their companions via pheromones. This information-sharing amongst colony members offers each ant beneficial guidance toward food, so that individual ants need not exhaustively search several paths in order to find food. To make an analogy to the RL setting, the ants' foraging process can be viewed as an effective way of policy improvement. Motivated by these insights from collective animal behavior, we incorporate the notion of collective knowledge sharing into the RL setting in an attempt to accelerate and guide the search for the optimal policy.

Our method emulates the collaborative behaviors of ants via an ensemble of RL agents: a group of agents collectively search for the optimal policy in the same task, while periodically sharing knowledge. Each RL agent resembles an individual in the ant colony. To elicit diverse experiences amongst members of the ensemble, each RL is randomly initialized with different neural network parameters. Random initialization results in adequate behavioral diversity [21, 32, 35] needed for collective exploration. As these agents are diverse in nature, at any given time during the course of training, one agent naturally has a policy superior to its peers. This agent can guide its peers towards higher-performing policies, just as ants share superior paths through pheromones. In order to guide the other agents, our method employs the knowledge distillation framework [23] which is effective at transferring knowledge between neural networks, without assuming identical model architectures. Knowledge distillation encourages other agents to act in a manner similar to the leading agent, allowing underperforming agents to quickly improve. With knowledge distillation, the agents can continue searching for the optimal policy from better starting points. However, despite the distillation, the agents' knowledge is still preserved, retaining diversity amongst the ensemble.

This paper's primary contribution is *collaborative inter-agent knowledge distillation* (CIKD), a simple yet effective framework for RL that jointly trains an ensemble of RL agents while periodically performing knowledge sharing. We demonstrate empirically that our method can improve the state-of-the-art soft-actor critic (SAC) [19] on a suite of challenging MuJoCo tasks, exhibiting superior sample efficiency and performance. We further validate the effectiveness of distillation for knowledge sharing by comparing against other methods of sharing knowledge. In addition, we present an in-depth investigation to explain the underlying causes of CIKD's performance improvement. Finally, our ablation study shows that a small ensemble is sufficient for improving performance.

The remainder of this paper is organized as follows. Section 2 discusses the related work. Section 3 provides the requisite background on RL. Section 4 introduces CIK. Section 5 presents our experimental findings. Section 6 summarizes our contributions and outlines potential avenues for future work.

2 Related work

The idea of jointly training multiple policies through RL has emerged in prior works. The most relevant works [35, 36] train multiple policies for the same task through RL, as our method does. Osband et al. [35, 36] train several agents in an ensemble while storing these agents' experiences in a shared buffer. Thus, agents share knowledge by sharing experiences amongst members of the ensemble, which are then used for RL updates. Our method is complementary to Osband et al. [35, 36]'s, in that we can also use a shared buffer of experiences, but we additionally performing periodic knowledge distillation between members of the ensemble, in particular from the best agent to other agents. Other related methods aggregate multiple policies to select actions [2, 6, 10, 16, 29, 38, 39, 43, 48, 52]. Abel et al. [1], Tosatto et al. [50], Wang & Jin [51] sequentially train a series of policies, boosting the learning performance by using the errors of a prior policy. However, rather than perform decision aggregation or sequentially-boosted training, we focus on improving the performance of each individual agent via knowledge sharing amongst jointly trained agents. Again, our method can be considered to be a complementary approach to decision aggregation and sequentially boosted training methods.

We can view the sharing of knowledge from the best policy as exploiting successful behavior patterns. This general notion has been explored in several areas of RL. Rusu et al. [40] and Parisotto et al. [37] can train a single network to perform multiple tasks by transferring multiple pre-trained RL agents' policies to a single network. Hester et al. [22] and Nair et al. [33] accelerate the RL agents'

training progress via human experts' guidance. Instead of using experts' policies, Levine & Koltun [28], Nagabandi et al. [31] and Zhang et al. [53] leverage model-based controllers' behaviors (e.g. model predictive controllers [14] or linear-quadratic regulators Dorato et al. [7]), facilitating training for RL agents. Additionally, Hong et al. [24] and Oh et al. [34] train agents to imitate past successful self-experiences or policies. Orthogonal to the aforementioned works, our method periodically exploits the current best policy amongst the ensemble, and shares it with the ensemble, enabling a collaborative search for the optimal policy.

Collaborative learning approaches have emerged in other areas of machine learning research. In computer vision, Zhang et al. [54] present deep mutual learning, which trains multiple models that mutually imitate each other's outputs on classification tasks. Our distillation is not mutual, and flows in a single direction, from a superior teacher agent to other student agents in the ensemble. In subsequent work, Lan et al. [27] train an ensemble of models to imitate a stronger teacher model. Simultaneously, this teacher model learns to aggregate all of the ensemble models' predictions. They demonstrate that imitating this aggregate teacher leads to better performance than deep mutual learning, which performs mutual imitation amongst members. Unlike Lan et al. [27], we do not use an aggregate teacher for distillation, and instead we collect performance statistics periodically to elect a single teacher for distillation. Moreover, unlike deep mutual learning, our distillation occurs in a single direction, i.e., from a teacher to a student, as opposed to bidirectional distillation.

Collaborative learning has also been explored in RL as well. Teh et al. [47] and Ghosh et al. [15] jointly learn independent policies for multiple tasks or contexts and then distill these policies to a central multi-task policy. Galashov et al. [11] learn a task-specific policy while bounding the divergence between this task-specific policy and some generic policy that can perform basic task-agnostic behaviors. Czarnecki et al. [5] gradually transfer the knowledge of a simple policy to a complex policy during the course of joint training. While our method also collaboratively trains policies, we differ from the aforementioned works in several aspects. First, our method periodically elects a leading agent for sharing knowledge rather than either constraining the mutual policy divergence [13, 15, 47, 54] or imitating aggregated models [27]. The second difference is that our method does not rely on training heterogeneous policies (e.g. a simple policy and a complex policy [5]), which makes our method more generally applicable. Finally, as opposed to Teh et al. [47] and Ghosh et al. [15], we consider single-task setting rather than multi-tasking.

Evolutionary algorithms (EA) [12, 20, 25, 41] similarly employ multiple policies to find the optimal policy. EA repeatedly performs mutation, selection, and reproduction on a maintained population. Mutation randomly perturbs the parameters of policies in the population; selection eliminates the underperforming policies by testing the policies' performance in the environment; reproduction produces the next generation of policies from the remaining policies. Unlike EA, our method does not rely on mutation and reproduction on a population of policies. While EA and our method are similar in an abstract sense, in that they both share knowledge amongst agents, they are quite different in practice. EA often eliminates members from its population and performs destructive changes to members of the population. Our method focuses on continuously improving the existing agents, and does not perform destructive changes to its population. In fact, our method can be incorporated into EA, serving as a more effective way to optimize each individual policy within the same generation.

3 Background

In this section we describe the general framework of RL. RL formalizes a sequential decision-making task as a *Markov decision process* (MDP) [45]. An MDP consists of a state space $\mathcal{S}$, a set of actions $\mathcal{A}$, a (potentially stochastic) transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a discount factor $\gamma \in [0, 1]$. An RL agent performs *episodes* of a task where the agent starts in a random initial state s_0, sampled from the initial state distribution ρ_{s_0}, and performs actions, experiencing new states and rewards. More generally, at timestep t, an agent in state s_t performs an action a_t, receives a reward r_t, and transitions to a new state s_{t+1}, according to the transition function $\mathcal{T}$. The discount factor γ is used to indicate the agent's preference for short-term rewards over long-term rewards.

An RL agent performs actions according to its policy, a conditional probability distribution $\pi_\phi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$, where ϕ denotes the parameters of the policy, which may be the parameters of a neural network. RL methods iteratively update ϕ via rollouts of experience $\tau = \{(s_t, a_t, r_t, s_{t+1})\}_{t=0}^{T}$, seeking within the parameter space Φ the optimal ϕ^* that maximizes the expected return $\mathbb{E}_{s\sim\rho_{s_0}}\left[\sum_{t=0}^{T} \gamma^t r_t | s_0 = s\right]$ within an episode.

4 Method

In this section, we formally present the technical details of our method, Collaborative Inter-agent Knowledge Distillation (CIKD). We first provide an overview of CIKD and then describe its components in detail.

4.1 Overview

Emulating collaborative foraging behaviors of ants, CIKD employs an ensemble of RL agents to perform a wide range of policies on the same task and periodically share knowledge amongst agents. CIKD can be divided into three phases: ensemble initialization, joint training, and inter-agent knowledge distillation. First, in the ensemble initialization phase, we randomly initialize an ensemble of RL agents to achieve behavioral diversity. In the joint training stage, each agent acts in the environment, adding its experiences to a shared buffer, and these shared experiences are used to update the parameters of the agent. Intermittently, we perform inter-agent knowledge distillation, where we elect a leading agent to guide the other agents towards superior policies. To this end, we distill [23] the best-performing agent's policy to the others. Algorithm 1 summarizes our method.

Algorithm 1 Collaborative Inter-agent Knowledge Distillation for Reinforcement Learning

Input: an environment $\mathcal{E}$, an ensemble size K, a parameter space Φ, a set of parameterized policies $\{\pi_{\phi_k}\}_{k=0}^{K}$, recent episodic performance statistics $\{R_k\}_{k=0}^{K}$, an episode length T, a distillation interval I, an experience buffer $\mathcal{D}$

i. Ensemble initialization
Randomly initialize policy parameters: $\phi_k \sim \text{Uniform}(\Phi), \forall k \in [0, K)$
Initialize the experience buffer: $\mathcal{D} \leftarrow \varnothing$
Initialize each recent episodic performance statistic $R_k, \forall k \in [0, K)$
Initialize timestep counter: $t_{acc} \leftarrow 0$
while not converged **do**
  ii. Joint training
  Policy selection: $k_e \sim \text{Uniform}([0, K))$
  Perform a rollout in the environment: $\tau \leftarrow \textsc{Rollout}(\mathcal{E}, \pi_{\phi_{k_e}})$
  Store the experience: $\mathcal{D} \leftarrow \mathcal{D} \cup \tau$ $(\tau = \{(s_t, a_t, r_t, s_{t+1})\}_{t=0}^{T})$
  Update all policies $\pi_{\phi_k}, \forall k \in [0, K)$ by off-policy RL methods using batches sampled from $\mathcal{D}$
  Update the selected recent-performance statistics: $\textsc{UpdateStat}(R_{k_e}, \tau)$
  Accumulate timestep counter: $t_{acc} \leftarrow t_{acc} + T$
  iii. Inter-agent Knowledge Distillation
  if $t_{acc} \geq I$ **then**
    Elect the teacher agent: $k_t = \text{argmax}_k R_k$
    Minimize $L(\phi_k, \phi_{k_t}), \forall k \in [0, K) \setminus k_t$ using $\mathcal{D}$ (Equation. 1)
    Reset counter: $t_{acc} \leftarrow 0$
  end if
end while

4.2 Ensemble initialization

We randomly initialize K RL agents in the ensemble. Each RL agent's policy is instantiated with a model parameterized by ϕ_k, where k denotes the agent's index in the ensemble. We initialize ϕ_k by sampling uniformly over parameter space Φ: $\phi_k \sim \text{Uniform}(\Phi)$. Despite the simplicity of uniform distributions, Osband et al. [35] show that uniform random initialization can provide adequate behavioral diversity. In this paper, we represent each $\phi_k, \forall k \in [0, K)$ as a neural network, though other parametric models (e.g. linear models) can be used.

CIKD can be easily applied to off-policy RL methods, including off-policy actor-critic methods. In an actor-critic method, the agent learns both a policy and a critic function that values states or state-action pairs. In an off-policy actor-critic method, the agents stores a replay buffer of past experiences which are used for training. In this paper, we use soft actor-critic (SAC) an off-policy actor-critic method that has achieved state-of-the-art results on challenging tasks [18, 19]. To apply CIKD to SAC, we create a shared replay buffer for all agents and randomly initialize a critic function for each policy π_{ϕ_k}.

4.3 Joint training

Each joint training phase consists of I timesteps. At the beginning of each episode, we select an agent in the ensemble to act in the environment (hereinafter this process referred to as "policy selection") and then store this episode in the replay buffer. Below, we describe the policy selection and policy update procedures.

The policy selection strategy is a way to select one policy $\pi_{\phi_{k_e}}$ from the ensemble, to perform one episode τ in the environment. This episode τ is then stored in a shared experience buffer $\mathcal{D}$, and the agent's recent episodic performance statistic R_{k_e} is updated according to the return achieved in τ, where R_{k_e} stores the average episodic return in the most recent M episodes. $\{R_k\}_{k=0}^{K}$ and $\mathcal{D}$ will later be used in inter-agent distillation (Section 4.4). In this paper, we adopt a simple uniform policy selection strategy, whereby at the beginning of each episode we select a policy from the ensemble at random to act in the episode. In doing so, we ensure that we have diverse data by collecting trajectories from all agents in the ensemble. After selecting a policy $\pi_{\phi_{k_e}}$ which performs an episode τ, we store this τ in $\mathcal{D}$. Then, we can sample data from $\mathcal{D}$ and update all policies using any arbitrary off-policy update method. Since off-policy update methods do not require that τ necessarily comes from a specific policy, they enable our policies to learn from trajectories generated by other members of the ensemble.

4.4 Inter-agent knowledge distillation

The inter-agent knowledge distillation phase consists of two stages: *teacher election* and *knowledge distillation*. Teacher election refers to the selection of an agent in the ensemble to serve as the teacher for the other ensemble members. In our experiments, we elect the teacher that has the highest performance, as determined by each agent's recent episodic performance statistics recorded during the joint training stage. Thus, the selected teacher agent is the agent with the highest recent performance, namely $k_t = \arg\max_k R_k$, where k_t is the index of the teacher. Rather than use the agent's most recent episodic performance, we use its average return over its previous M episodes, to minimize the noise in our estimate of the agent's performance.

Next, the teacher serves as a guide that leads the other agents towards higher-performing policies. This is done through knowledge distillation [23], which has been shown to be effective at guiding a neural network to behave similar to another neural network. To distill from the teacher to the students (i.e., the other ensemble members), the teacher samples experiences from the buffer $\mathcal{D}$ and instructs each student to match its outputs on these samples. The intuition is that after distillation, the students acquire the teacher's knowledge, enabling them to correct their sub-optimal behaviors and reinforce their correct behaviors. Specifically, the distillation process can be formalized as minimizing the following loss function:

$$L(\phi_k, \phi_{k_t}) = \mathbb{E}_{s\sim\mathcal{D}}\big[D_{KL}(\pi_{\phi_{k_t}}(.|s)||\pi_{\phi_k}(.|s))\big], \tag{1}$$

where s denotes historical states sampled from the experience buffer $\mathcal{D}$ and D_{KL} stands Kullback-Leibler (KL) divergence. Note that inter-agent distillation is also compatible with actor-critic methods. For actor-critic methods, we additionally distill the critic function from the teacher to the students. In this paper, we distill the critic by replacing D_{KL} with the l_2-loss function.

5 Experiments

The experiments are designed to answer the following questions: (1) Can CIKD improve upon the data efficiency of state-of-the-art RL? (2) Is knowledge distillation effective at sharing knowledge? (3) What is the impact of the ensemble size? (4) Can CIKD utilize the advantages of agents with different model architectures? Next, we show our experimental findings for each of the aforementioned questions, and discuss their implications.

5.1 Experimental setup

Implementation. Our goal is to demonstrate how CIKD improves the sample efficiency of an RL algorithm. Since soft actor-critic (SAC) [19] exhibits state-of-the-art sample efficiency across several simulated benchmarks [49] and even in real robots [17, 18], we build on top of SAC. For the remainder of this section, we term CIKD applied to SAC as *SAC-CIKD*. We directly use the hyperparameters for SAC from the original paper [19] in all of our experiments. Unless stated otherwise, the hyperparameters we use for *SAC-CIKD* (Algorithm 1) are $I = 5000$, and $K = 3$ for all experiments. The values for I and M are tuned via grid search over $[1000, 2000, \cdots, 10000]$, and $[1, 2, \cdots, 10]$ respectively. We experiment with $K \in \{2, 3, 5\}$ in Section 5.4.

Benchmarks. We use OpenAI gym [3]'s MuJoCo [49] benchmark tasks, as used in the original SAC [19] paper. We choose all tasks selected in the original paper [19] and two additional tasks to evaluate the performance of our method. The description for each task can be found in the source code for OpenAI gym[2].

Evaluation. We adapt the evaluation approach from the original SAC paper [19]. We train each agent for 1 million timesteps, and run 20 evaluation episodes after every 10,000 timesteps (i.e., 10,000 interactions with the environment), where the performance is the mean of these 20 evaluation episodes. We repeat this entire process across 5 different runs, each with different random seeds. We plot the mean value and confidence interval of mean episodic return at each stage of training. The mean value and confidence interval are depicted by the solid line and shaded area, respectively. The confidence interval is estimated by the bootstrapped method [9]. At each evaluation point, we report the highest mean episodic return amongst the agents in the ensemble. In some plots, we additionally report the lowest mean episodic return amongst the agents in the ensemble.

5.2 Effectiveness of inter-agent knowledge distillation

In order to evaluate the effectiveness of inter-agent knowledge distillation, we compare our method, *SAC-CIKD*, against two baselines: *Vanilla-SAC* and *Ensemble-SAC*. *Vanilla-SAC* stands for the original SAC; *Ensemble-SAC* resembles the implementation of Osband et al. [35]'s method on SAC (analogous to CIKD-RL without inter-agent knowledge distillation). Ensemble sizes (K) for *Ensemble-SAC* and *SAC-CIKD* are set to 3. Our results are shown in Figure 1. Note that we also plot the worst evaluation in our ensemble at each evaluation phase to provide some insight into the general performance of the ensemble. In all tasks, we outperform all baselines, including *Vanilla-SAC* and *Ensemble-SAC*. Moreover, *SAC-CIKD* has far better sample efficiency, usually reaching the best baseline's final performance in half of the environment interactions. We even find that in the majority of tasks, our worst evaluation in the ensemble outperforms the baseline methods. This demonstrates that all members of the ensemble are significantly improving, and our method's superior performance is not simply a consequence of selecting the best agent in the ensemble. Notice that *Ensemble-SAC* does not significantly improve the performance of *Vanilla-SAC*. *Ensemble-SAC* only outperforms *Vanilla-SAC* in 4 out of 7 tasks, suggesting that the diversity of the ensemble is alone insufficient for achieving large performance gains over *Vanilla-SAC*. In particular, *SAC-CIKD*'s superiority over *Ensemble-SAC* highlights the effectiveness of supplementing shared experiences (*Ensemble-SAC*) with knowledge distillation. In summary, Figure 1 demonstrates the effectiveness of inter-agent knowledge distillation at enhancing the performance and data efficiency of state-of-the-art RL algorithms.

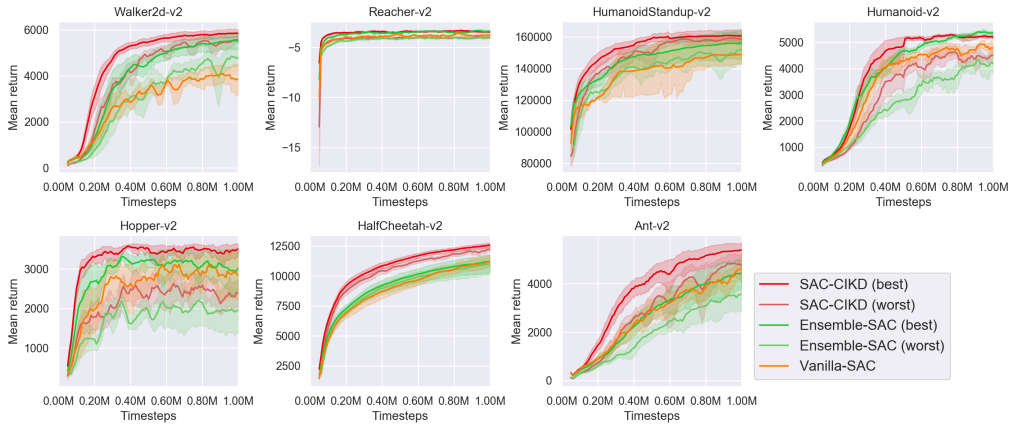

Figure 1: **Performance of inter-agent knowledge distillation**. *SAC-CIKD* refers to the application of CIKD to SAC; *Vanilla-SAC* refers to the original SAC; *Ensemble-SAC* refers to the application of Osband et al. [35]'s method to *Vanilla-SAC* (analogous to *SAC-CIKD* without inter-agent knowledge distillation). See Section 5.2 for details. Notice that in most domains, *SAC-CIKD* is able to reach the final performance of the baselines in less than half the training time.

[2]https://github.com/openai/gym

5.3 Effectiveness of knowledge distillation for knowledge sharing

Though Section 5.2 has shown that *Ensemble-SAC*, which updates all agents' policies through shared experiences, fails to perform as well as *SAC-CIKD*, *SAC-CIKD* uses additional gradient updates during the inter-agent knowledge distillation phase, whereas *Ensemble-SAC* only performs joint training. It is unclear whether extra policy updates in lieu of knowledge distillation can achieve the same effects as knowledge distillation. To investigate this, we compare our method to *Vanilla-SAC (extra)* and *Ensemble-SAC (extra)*, which respectively correspond to *Vanilla-SAC* and *Ensemble-SAC* (see Section 5.2) agents that are trained with extra policy update steps. Specifically, instead of the knowledge distillation phase, we provide these baseline agents with the same number of policy updates and minibatch sizes as we give the *SAC-CIKD* agent for knowledge distillation. A policy update here refers to a training step for updating the policy [42, 46] and the value function [19, 26], if required, by RL algorithms. Figure 2 shows the performance of the above baselines and *SAC-CIKD*. It can be seen that *SAC-CIKD* outperforms all the baselines. This observation shows that knowledge distillation is more effective than policy updates for knowledge sharing.

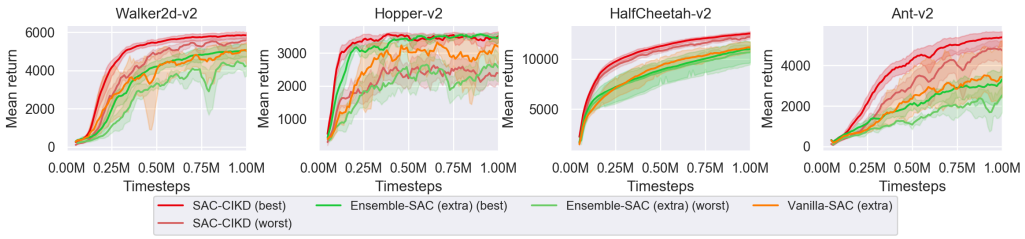

Figure 2: **Comparison between knowledge distillation and extra policy updates.** *Vanilla-SAC (extra)* and *Ensemble-SAC (extra)* refer to *Vanilla-SAC* and *Ensemble-SAC* variants that use extra policy updates, respectively (see Section 5.3 and Section 3 for details). This figure indicates that knowledge distillation is more effective.

5.4 Ablation study on Ensemble Size

In this section, we study the influence of the ensemble size K, since the ensemble size is directly related to scalability. We test our method with three ensemble sizes $K = 2$, $K = 3$, and $K = 5$. Figure 3 shows the performance of these configurations. We find that *SAC-CIKD* performs approximately the same across all three ensemble sizes. Even with an ensemble size of 2, we see better performance than *Ensemble-SAC* (as $K = 2$ is on-par with $K = 3$, which outperforms *Ensemble-SAC*, as shown before in Figure 1). Thus, our method can reap benefits even from small ensembles, and is not extremely sensitive to the ensemble size.

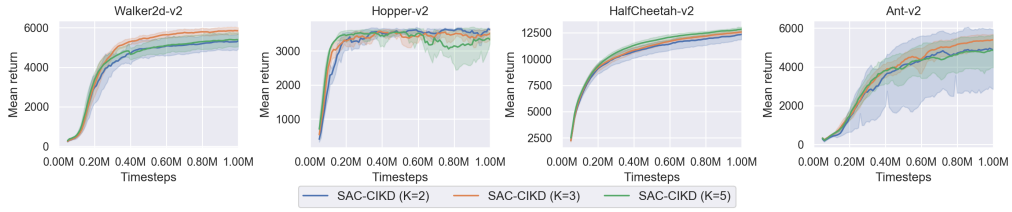

Figure 3: **Performance comparison under different ensemble sizes.** Three different ensemble configurations with 2, 3, and 5 agents lead to similar performance. This result demonstrates that CIKD does not require a large ensemble size.

5.5 Towards training complex neural network policies along with simple ones

A recent work [5] shows that a simple neural network can learn faster than a complex neural network does, while a complex one can attain better performance in a long run. We investigate whether CIKD can leverage the best of both networks via knowledge sharing. To this end, we compare the performance of *SAC-CIKD*, *Ensemble-SAC*, and *Vanilla-SAC* on training SAC agents that use 256 and 1024 hidden units for each layer in their policy and critic networks (hereinafter, the number of hidden units per layer is referred to as "width"). Note that we choose 256 for a simple network since 256 is the default setting in the original SAC paper [19]. We also change the composition of ensemble in *SAC-CIKD* and *Ensemble-SAC*, where the new ensemble consists of only two agents with

different neural network architectures. In Figure 4, we plot the mean returns of two architecturally different agents during the course of training. Figure 4 shows that the complex agent trained by *SAC-CIKD* outperforms those trained by the other methods, indicating that *SAC-CIKD* facilitates training complex neural network policies/critics by the knowledge of the simple agent. In addition, Figure 4 shows that the simple agent trained by *SAC-CIKD* achieves the highest performance among all methods, suggesting that *SAC-CIKD* also enables the simple agent to benefit from the complex agent. Furthermore, it can be seen that *Ensemble-SAC* is failed to help training the complex agent even though the experience is shared among both agents, further confirming that CIKD enables the notion of the best-performing agent better to be transferred to the other agents and could be an effective way to utilize the best of a simple and a complex neural network policy.

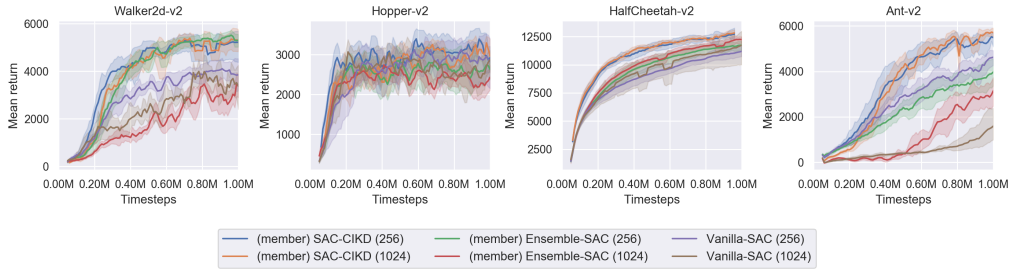

Figure 4: **The result of CIKD on training a complex neural network policy** The number between the parentheses denotes the number of hidden units in the agent's policy/critic networks (or simply "width"). For instance, *Vanilla-SAC (256)* stands for training a *Vanilla SAC* agent of width 256. Note that *(member) SAC-CIKD* and *(member) Ensemble-SAC* indicates the performance of a member in an ensemble trained by *SAC-CIKD* and *Ensemble-SAC* respectively. This figure shows that *SAC-CIKD (256/1024)* learn better policies than *Vanilla-SAC (256/1024)* and *Ensemble-SAC (256/1024)*, indicating that *SAC-CIKD (256/1024)* benefit from inter-agent knowledge distillation.

6 Conclusion

In this paper, we introduce collaborative inter-agent knowledge distillation (CIKD), a method that jointly trains an ensemble of RL agents while continually sharing information via knowledge distillation. Our experimental results demonstrate that CIKD improves the performance and data efficiency of a state-of-the-art RL method on several challenging MuJoCo tasks. We also show that knowledge distillation is more effective than other approaches for knowledge sharing. We found that electing the best-performing agent to serve as the teacher plays a significant role in improving performance. Our investigation further showed that the combination of students' and teachers' knowledge is crucial for improving performance. Our ablation study demonstrates that a large ensemble is not needed for improving performance. Lastly, we demonstrate that CIKD can benefit the training of heterogenous model architectures.

CIKD opens several avenues for future work. First encouraging diversity within the ensemble may lead to more efficient exploration [4, 24]. Additionally, while we used a simple uniform policy selection strategy, a more efficient policy selection strategy may further accelerate learning. Lastly, while our ensemble members used identical architectures, CIKD may benefit from using heterogeneous ensembles. For example, different networks may have different architectures that are conducive to learning different skills, which can then be distilled within the ensemble.

Acknowledgement

The authors would like to thank Aaron Havens for suggestions for interesting experiments. We thank Yasuhiro Fujita for suggesting experiments and providing technical support. We thank Jean-Baptiste Mouret for useful feedback on our draft and formulation. Lastly, we thank Pieter Abbeel and Daisuke Okanohara for helpful advice on related works and the framing of the paper.

References

[1] David Abel, Alekh Agarwal, Fernando Diaz, Akshay Krishnamurthy, and Robert E Schapire. Exploratory gradient boosting for reinforcement learning in complex domains. *arXiv preprint arXiv:1603.04119*, 2016.

[2] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[4] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Advances in Neural Information Processing Systems*, pp. 5027–5038, 2018.

[5] Wojciech Marian Czarnecki, Siddhant M Jayakumar, Max Jaderberg, Leonard Hasenclever, Yee Whye Teh, Simon Osindero, Nicolas Heess, and Razvan Pascanu. Mix&match-agent curricula for reinforcement learning. *arXiv preprint arXiv:1806.01780*, 2018.

[6] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pp. 271–278, 1993.

[7] Peter Dorato, Vito Cerone, and Chaouki Abdallah. *Linear quadratic control: an introduction*. Krieger Publishing Co., Inc., 2000.

[8] Marco Dorigo and Mauro Birattari. *Ant colony optimization*. Springer, 2010.

[9] Bradley Efron. Better bootstrap confidence intervals. *Journal of the American statistical Association*, 82(397):171–185, 1987.

[10] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*, 2017.

[11] Alexandre Galashov, Siddhant M Jayakumar, Leonard Hasenclever, Dhruva Tirumala, Jonathan Schwarz, Guillaume Desjardins, Wojciech M Czarnecki, Yee Whye Teh, Razvan Pascanu, and Nicolas Heess. Information asymmetry in kl-regularized rl. *arXiv preprint arXiv:1905.01240*, 2019.

[12] Tanmay Gangwani and Jian Peng. Policy optimization by genetic distillation. *arXiv preprint arXiv:1711.01012*, 2017.

[13] Tanmay Gangwani, Qiang Liu, and Jian Peng. Learning self-imitating diverse policies. *arXiv preprint arXiv:1805.10309*, 2018.

[14] Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[15] Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. *arXiv preprint arXiv:1711.09874*, 2017.

[16] Michael Gimelfarb, Scott Sanner, and Chi-Guhn Lee. Reinforcement learning with multiple experts: A bayesian model combination approach. In *Advances in Neural Information Processing Systems*, pp. 9528–9538, 2018.

[17] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6244–6251. IEEE, 2018.

[18] Tuomas Haarnoja, Aurick Zhou, Sehoon Ha, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[19] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[20] Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366, 2014.

[21] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[22] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[23] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[24] Zhang-Wei Hong, Tzu-Yun Shann, Shih-Yang Su, Yi-Hsiang Chang, Tsu-Jui Fu, and Chun-Yi Lee. Diversity-driven exploration strategy for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 10489–10500, 2018.

[25] Shauharda Khadka and Kagan Tumer. Evolution-guided policy gradient in reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1188–1200, 2018.

[26] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pp. 1008–1014, 2000.

[27] Xu Lan, Xiatian Zhu, and Shaogang Gong. Knowledge distillation by on-the-fly native ensemble. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 7528–7538. Curran Associates Inc., 2018.

[28] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pp. 1–9, 2013.

[29] Josh Merel, Arun Ahuja, Vu Pham, Saran Tunyasuvunakool, Siqi Liu, Dhruva Tirumala, Nicolas Heess, and Greg Wayne. Hierarchical visuomotor control of humanoids. *arXiv preprint arXiv:1811.09656*, 2018.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[31] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566. IEEE, 2018.

[32] Prabhat Nagarajan, Garrett Warnell, and Peter Stone. Deterministic implementations for reproducibility in deep reinforcement learning. In *AAAI 2019 Workshop on Reproducible AI*, January 2019.

[33] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6292–6299. IEEE, 2018.

[34] Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. *arXiv preprint arXiv:1806.05635*, 2018.

[35] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pp. 4026–4034, 2016.

[36] Ian Osband, Benjamin Van Roy, Daniel Russo, and Zheng Wen. Deep exploration via randomized value functions. *arXiv preprint arXiv:1703.07608*, 2017.

[37] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[38] Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):81, 2016.

[39] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

[40] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[41] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[42] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

[43] Satinder P Singh. The efficient learning of multiple task sequences. In *Advances in neural information processing systems*, pp. 251–258, 1992.

[44] David JT Sumpter. *Collective animal behavior*. Princeton University Press, 2010.

[45] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[46] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[47] Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 4496–4506, 2017.

[48] Chen K Tham. Reinforcement learning of multiple tasks using a hierarchical cmac architecture. *Robotics and Autonomous Systems*, 15(4):247–274, 1995.

[49] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

[50] Samuele Tosatto, Matteo Pirotta, Carlo d'Eramo, and Marcello Restelli. Boosted fitted q-iteration. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3434–3443. JMLR. org, 2017.

[51] Yu Wang and Hongxia Jin. A boosting-based deep neural networks algorithm for reinforcement learning. In *2018 Annual American Control Conference (ACC)*, pp. 1065–1071. IEEE, 2018.

[52] Marco A Wiering and Hado Van Hasselt. Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):930–936, 2008.

[53] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *2016 IEEE international conference on robotics and automation (ICRA)*, pp. 528–535. IEEE, 2016.

[54] Ying Zhang, Tao Xiang, Timothy M Hospedales, and Huchuan Lu. Deep mutual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4320–4328, 2018.