Curriculum-Based Asymmetric Multi-Task Reinforcement Learning

Hanchi Huang*, Deheng Ye*, Li Shen*, and Wei Liu*, Fellow, IEEE

Abstract—We introduce CAMRL, the first curriculum-based asymmetric multi-task learning (AMTL) algorithm for dealing with multiple reinforcement learning (RL) tasks altogether. To mitigate the negative influence of customizing the one-off training order in curriculum-based AMTL, CAMRL switches its training mode between parallel single-task RL and asymmetric multi-task RL (MTRL), according to an indicator regarding the training time, the overall performance, and the performance gap among tasks. To leverage the multi-sourced prior knowledge flexibly and to reduce negative transfer in AMTL, we customize a composite loss with multiple differentiable ranking functions and optimize the loss through alternating optimization and the Frank-Wolfe algorithm. The uncertainty-based automatic adjustment of hyper-parameters is also applied to eliminate the need of laborious hyper-parameter analysis during optimization. By optimizing the composite loss, CAMRL predicts the next training task and continuously revisits the transfer matrix and network weights. We have conducted experiments on a wide range of benchmarks in multi-task RL, covering Gym-minigrid, Meta-world, Atari video games, vision-based PyBullet tasks, and RL Bench, to show the improvements of CAMRL over the corresponding single-task RL algorithm and state-of-the-art MTRL algorithms. The code is available at: https://github.com/huanghanchi/CAMRL.

Index Terms—Reinforcement learning, curriculum learning, asymmetric multi-task learning

1 INTRODUCTION

MULTI-TASK learning (MTL) trains multiple related tasks simultaneously while taking advantage of similarities and differences between tasks [38]. However, in MTL, negative transfer may occur since not all tasks can benefit from joint learning. To handle this, asymmetric transfer between every pair of tasks has been developed [19], such that the amount of transfers from a confident network to a relatively less confident one is larger than the other way around. The core idea is to learn a sparse weighted directed regularization graph between every two tasks through a curriculum-based asymmetric multi-task learning (AMTL) mode, which has been proven to be very effective in supervised learning [19], [20], [24], while remaining to be unexplored in reinforcement learning (RL).

In this paper, inspired by AMTL, we study the problem of mitigating negative transfer in multi-task reinforcement learning (MTRL). Note that incorporating AMTL techniques into the context of RL is not easy. Due to the non-stationarity when training RL tasks and the lack of prior knowledge on RL tasks’ properties [33], [34], it is likely that we end up with learning a poor regularization graph between tasks, if we directly adopt curriculum-based AMTL to train tasks one by one without proper corrections to the training order. Besides, some important factors for flexibilizing the transfer among tasks and for avoiding negative transfer, such as indicators representing relative training progress, the transfer performance between tasks, and the behavioral diversity described in [25], have been neglected by existing works on AMTL [19], [20], [24], which may lead to slow convergence for a RL task, and cause serious negative transfer before convergence. Also, it is worth noting that performing curriculum-based AMTL all the time is time-consuming and can lead to performance degradation when each task is poorly trained or has already been well-trained. Consequently, RL tasks require different demands for curriculum learning at different training stages and the amount of experience learned from different tasks needs to be dynamically adjusted. To deal with the above issues, we propose CAMRL, the first Curriculum-based Asymmetric Multi-task algorithm for Reinforcement Learning. To balance both the efficiency and the amount of curriculum-based transfers across tasks, we design a composite indicator weighting multiple factors to decide which training mode to switch between parallel single-task learning and curriculum-based AMTL. To avoid customizing the one-off order of training tasks, CAMRL re-calculates the above indicator at the beginning of every epoch to switch the training mode. When entering the AMTL training mode, CAMRL updates the training order, the regularization graph, and the network weights by optimizing a composite loss function through alternating optimization and the Frank-Wolfe algorithm. The loss function regularizes both the amount of outgoing transfers and the similarity across multiple network weights. Furthermore, three novel differentiable ranking functions are proposed to flexibly incorporate various prior

* Hanchi Huang is with the Nanyang Technological University, Singapore 639798. E-mail: hchuang@ntu.edu.sg.
* Deheng Ye and Wei Liu are with the Tencent Inc., Shenzhen 518052, China. E-mail: dericye@tencent.com, w2223@columbia.edu.
* Li Shen is with the JD.com Inc., Beijing 101111, China. E-mail: mathshenli@gmail.com.

Manuscript received 2 September 2021; revised 31 July 2022; accepted 7 November 2022. Date of publication 23 November 2022; date of current version 5 May 2023. This work was supported in part by Science and Technology Innovation 2030 “Brain Science and Brain-like Research” key Project under Grant 2021ZD0201405.

(Corresponding authors: Deheng Ye and Wei Liu.)

Recommended for acceptance by L. Li.

This article has supplementary downloadable material available at https://doi.org/10.1109/TPAMI.2022.3223872, provided by the authors.

Digital Object Identifier no. 10.1109/TPAMI.2022.3223872, provided by the authors.
knowledge into the loss, such as the relative training difficulty, the performance of mutual evaluation, and the similarity between tasks. Finally, we discuss the flexibility of CAMRL from multiple perspectives and its limitations, so as to illuminate further incorporations of curriculum-based AMTL into RL.

Our contributions are summarized as follows:

- We propose the CAMRL (curriculum-based asymmetric multi-task reinforcement learning) algorithm, which is designed with a training-mode-switching mechanism and a combinatorial loss containing multiple differentiable ranking functions. These designs tackle the common issues in multi-task RL, e.g., negative transfer and poor utilization of prior knowledge.
- CAMRL can be paired with various RL-based algorithms and training modes, as well as absorbing various prior knowledge and ranking information of training factors of any amount, which are rarely seen in previous MTRL works [28], [32], [35]. Moreover, by dynamically adjusting parameters, CAMRL can adapt to the arrival of a new task via a simple correction.
- Experiments on a series of low-/high-dimensional RL tasks show that CAMRL remarkably outperforms the corresponding single-task RL algorithm and state-of-the-art multi-task RL algorithms.

2 RELATED WORK

2.1 Multi-Task Reinforcement Learning (MTRL)

Before stepping into the deep RL era, most works [6], [27] on multi-task-oriented algorithms within RL attempted to rely on the assistance from transfer learning. Later on, knowledge sharing in multi-task deep RL was proposed to distill the common traits among all tasks [12], [28]. Since negative interference among gradients from various tasks may occur frequently during the distilling process, the work [35] presented a gradient surgery method that alters gradients by projecting each onto the normal plane of the other when conflicts happen between gradients. Researchers also tried to capture the similarity between gradients from multiple tasks [7], [40]. Furthermore, apart from driving gradients to be similar, applying compositional models is also a natural approach to handle gradients’ conflicts. By breaking down a multi-task problem into modules and supporting composability of separate modules in new modular RL agents, the compositional model enables to relieve the conflicts among gradients and facilitate better generalization [9]. However, training sub-policies separately often requires well-defined subtasks, which may be infeasible in real-world applications. Rather than manually defining the modules or sub-policies and the specification of their combination, the authors of [32] introduced the soft module, a fully end-to-end method which generates soft combinations of multiple modules automatically without specifying the policy structure in advance.

2.2 Curriculum Learning

Curriculum learning (CL) describes a learning mode in which we start with easy tasks and then gradually increase the task difficulty. The idea of using curricula to train learning agents was first proposed in [11] and first introduced for deep learning by [5]. Over the years, curricula have been applied to train complex tasks or tasks with sparse rewards [2], [3], [31], [33]. Early research efforts focused on the manual configuration of the curriculum which can be time-consuming and requires domain knowledge. To deal with the above issue, automatic CL has gained popularity recently, which aims to automatically adapt the distribution of training data by learning to adjust the selection of curriculum according to some factors such as diversity and learning progress [25]. Our work is, however, based on another line of works [19], [24] that select the next task to train by performing joint optimization on the loss of supervised learning tasks with $\ell_2$-norm regularizations on task parameters.

3 METHODOLOGY

3.1 Overview

The pipeline of CAMRL is shown in Fig. 1. In CAMRL, we customize a learning progress indicator to determine whether to perform parallel single-task training or curriculum-based AMTL at the beginning of each epoch. When performing curriculum-based AMTL, a composite loss function that learns the transfer between tasks and considers several factors to avoid negative training is adopted. We apply the alternating optimization and Frank-Wolfe algorithm to decide the next task for training and update the transfer matrix. In the meantime, the hyper-parameters in the loss are automatically adjusted according to each term’s historical uncertainty. When a new task arrives, CAMRL can quickly adapt to the new scenario without much side effect to the original tasks, by merely modifying the transfer matrix.
Algorithm 1. CAMRL Algorithm

1: Inputs: $\mu_1, \mu_2, \lambda_i (i = 0, 1, 2, 3, 4)$, number of tasks $T$.
2: Initialization: The transfer matrix $B = [I^{T \times T}]$, the soft actor-critic network $SAC_i$ with parameters $w_i$ for $t \in [T]$; $W = [w_1, w_2, \ldots, w_T]$.
3: for $n = 1, 2, \ldots, N$ do
4: Perform parallel training of all $SAC_i$ for $t \in [T]$ for $K$ episodes without interference among losses.
5: end for
6: for $n = N + 1, N + 2, \ldots$ do
7: Calculate $I_{\text{mul}}$ and sample $u \sim \text{Uniform}(0, 1)$.
8: if $u < I_{\text{mul}}$ then
9: Perform parallel training of $SAC_i (t \in [T])$ for $K$ episodes without interference among losses.
10: else
11: Predict the next training task $t$ and update $b^t = (B_{t1}, B_{t(t-1)}, B_{t(t+1)}, \ldots, B_{tT})$ and $w_t$ by optimizing:
$$
(t, b^t) \leftarrow \arg \min_{t \in [T], b^t} \left\{ \lambda_0 \left\| (1 + \mu t \| b^t \| _1) L(w_t) \right\| - \mu_2 (b^t)^T b^t + \lambda_1 \sum_{s \in [T]} \left[ \sum_{j=1}^{T-1} B_{ij} w_{s(j)} - B_{ij} w_t \right] ^2 \\
+ \lambda_2 \sum_{j \in [T]} \left( j - y_j \right)^2 + \lambda_3 \sum_{j \in [T]} \left( \frac{\text{rank} (k_j)}{T} - y_j \right)^2 \right\}. 
$$
(1)
12: Fix $b^t, W$ and select the task $t$ which minimizes the objective in Eq. (1).
13: Fix $t$, $W$ and run vanilla Frank-Wolfe on Eq. (7) for several iterations to optimize $b^t$.
14: Fix $b^t$ and train task $t$ with the policy loss Eq. (1) for $K$ episodes.
15: end if
16: end for

The overall training procedure of CAMRL is summarized in Algorithm 1. Assume that we have $T$ tasks with varying degrees of difficulty. In the beginning, we simply equip each task $t \in [T]$ with a soft actor-critic (SAC) network $SAC_i$ [15], and perform parallel single-task training for several epochs without interference among losses for different tasks, that is, each $SAC_i (t \in [T])$ is trained independently. Denote $w_t$ as the network parameters of the $t$th SAC network. Next, according to an indicator related to the learning progress, CAMRL decides whether to switch the training mode into curriculum-based AMTL, which trains the tasks one by one by optimizing a composite loss term regularizing the transfer matrix between tasks. The indicator $I_{\text{mul}} = \frac{\sum_{t=1}^{T} I_{\text{mul}}(t)}{T}$ is the positive weighted sum of the following terms: (i) $\exp(-n/a)$, where $n$ refers to the number of the current epochs. We use $\exp(-n/a)$ to encourage larger probability of multi-task training in the early stage so that hard-to-train tasks could benefit from easy-to-train tasks as early as possible. As we will show in Table 12, the performance of CAMRL downgrades obviously when we remove this term in $I_{\text{mul}}$; (ii) $\exp(-L_{\text{nor}} * b)$, where $L_{\text{nor}}$ means the average normalized policy loss of all tasks during the last epoch. We use $L_{\text{nor}}$ to judge the overall training progress. If the overall training performance is poor, i.e., with a large $L_{\text{nor}}$, then the possibility of performing inter-task transfer tends to be smaller. Note that in experiments, we set $L_{\text{nor}}$ to be $\left(\frac{\text{policy_loss} - \text{average policy_loss}}{\text{std policy loss}}\right)$. See the following remark for the details of the policy loss; (iii) the percentage of tasks whose average normalized reward during the last epoch locates outside the interval $[R_{\text{nor}} - \sigma * \text{std}_{\text{nor}}, R_{\text{nor}} + \sigma * \text{std}_{\text{nor}}]$, where $\text{std}_{\text{nor}}$ is the standard deviation of the normalized rewards for all tasks during that period. This term indicates that the larger the learning process gap among tasks, the larger the probability of performing multi-task training.

Remark Policy loss of SAC in [32]. Due to the implementation issue of SAC inheriting from the code of the soft-module [32], we find it much more convenient to use the negative normalized version of the policy loss, the direct output of the ‘train’ function in the code of the soft-module, instead of the normalized return. Besides, when encountering into difficult tasks with sparse rewards, the normalized return might keep nearly unchanged from time to time. In such scenarios, utilizing the negative normalized policy loss can help speed up model training more than the normalized return. Specifically, the policy loss of the SAC used in [32] can be characterized as $\log \pi _{\theta} (a_t | s_t) - Q_{\phi} (a_t, s_t)$, where $\pi _{\theta} (a_t | s_t)$ is the probability of adopting action $a_t$ in face of the state $s_t$ and $Q_{\phi} (a_t, s_t)$ is the estimated value of the state-action pair $(s_t, a_t)$. If $u \sim \text{Uniform}(0, 1)$ is smaller than $I_{\text{mul}}$, then the curriculum-based AMTL training mode will be temporarily adopted in the next epoch; otherwise, we will perform the single-task training mode (see line 8-10 in Algorithm 1). Here we need to note that: $I_{\text{mul}}$ is constantly changing, it does not matter if its value is temporarily greater than 1, which indicates that performing curriculum-based AMTL training is necessary at this time. When curriculum-based AMTL is unnecessary, $I_{\text{mul}}$ falls back down again and becomes smaller than 1.

3.2 Curriculum Multi-Task Training

In this subsection, we follow [19] to establish the foundation of curriculum multi-task training for our CMARL.

Let $B$ be a $T \times T$ matrix symbolizing the count of transfers between each pair of tasks. For $t \in [T]$, let $w_t$ be the parameters of the $t$th soft actor-critic network. We follow the assumption in [19] that for all $t \in [T], w_t \approx \sum_{s=1} T B_{ts} w_s$, that is, $B_{ts}$ refers to the positive weight of basis $w_s$ representing $w_t$. Denote $W := (w_1, w_2, \ldots, w_T)$.

The core of CAMRL is adapted from the following composite loss [19]:
$$
L(W, B) = \sum_{t=1}^{T} \left\{ (1 + \mu \| b^t \| _1) L(w_t) + \lambda \| w_t - \sum_{s \neq t} B_{ts} w_s \| _2^2 \right\}, 
$$
(2)
where $b^t = (B_{t1}, B_{t(t-1)}, B_{t(t+1)}, \ldots, B_{tT}) \in \mathbb{R}^{T-1}$ represents the count of outgoing transfers from task $t$ to other tasks and we use $\| b^t \| _1$ to satisfy the sparsity property of the transfer matrix $B$; $L(w_t)$ is the policy loss for task $t$ under $w_t$, and $w_t$ is the parameters of the $t$th SAC network; $(\lambda, \mu)$ are the coefficients of weights for different terms in Eq. (2).
Since simultaneous training of W and B may result in a serious negative transfer and a dimensional disaster when performing optimization, Lee et al. [19] formulated the curriculum-based training mode leveraging the paradigm in Eq. (2), for the purpose of finding the optimal order of training tasks and only optimizing $b_i^q$ and $w_i$ rather than B and W when training the task t.

Denote $\mathcal{S}$ as the permutation space over T elements, and for $\pi \in \mathcal{S}$, denote $\pi(i)$ as the ith element in permutation $\pi$. To perform curriculum learning of multiple tasks, Lee et al. [19] adapted their goal to cope with Eq. (3)

$$\min_{\pi \in \mathcal{S}, \theta \geq 0} \sum_{i=1}^{T} \left\{ (1 + \mu \|b_i^\pi\|_1) \mathcal{L}(w_i) + \lambda \|w_i - \sum_{j=1}^{i-1} B_{(\pi(j))} w_{\pi(j)} \|_2^2 \right\}. \quad (3)$$

Let $T := \{\pi(1), \ldots, \pi(i-1)\}$ be trained tasks and $U := \{1, \ldots, T\} - T$ be untrained tasks in the current epoch, respectively. Lee et al. [19] then found a task $i \in U$ to be learned next, so as to improve the future learning process the most

$$(t, b_t^\pi) \leftarrow \arg \min_{t \in U, b_t^\pi} \left\{ (1 + \mu \|b_t^\pi\|_1) \mathcal{L}(w_t) + \lambda \|w_t - \sum_{j=1}^{i-1} B_{(\pi(j))} w_{\pi(j)} \|_2^2 \right\}. \quad (4)$$

After selecting the task to be learned next and updating $b_t^\pi$ by performing alternating optimization on Eq. (4), Lee et al. [19] solved problem Eq. (5) to greedily minimize Eq. (3)

$$w_t \leftarrow \arg \min_{w_t} \left\{ (1 + \mu \|b_t^\pi\|_1) \mathcal{L}(w_t) + \lambda \|w_t - \sum_{j=1}^{i-1} B_{(\pi(j))} w_{\pi(j)} \|_2^2 \right\}. \quad (5)$$

### 3.3 Loss Modification

To reasonably distribute the count of transfers between every two tasks and reduce negative transfers, we have the following expectations and we modify the loss term in CAMRL based on these expectations:

**[Expectation 1]** Denote $p_{t,i}$ as the performance, usually the average rewards over several evaluation episodes, on training task $i$ by using the network originally for training task task t when training task t, if we can test the transferability among several tasks and approximately obtain $p_{t,i} > p_{t,i+1} > \cdots > p_{t,i,q}$ for some tasks $i_1, i_2, \ldots, i_q$, then we expect that $B_{t,i_1} > B_{t,i_2} > \cdots > B_{t,i_q}$ holds as much as possible, that is, the better evaluation performance on other tasks, the larger number of transfers on those tasks. To meet the above expectation, we expect $\sum_{j=1}^{q} (1 - y_{t,i_j})^2$ to be as small as possible, where $y_{t,i_j}$ is the ranking of $p_{t,i_j}$ among $p_{t,i_1} > p_{t,i_2} > \cdots > p_{t,i_q}$ and $y_{t,i_j}$ is the ranking of $B_{t,i_j}$ among $B_{t,i_1}, B_{t,i_2}, \ldots, B_{t,i_q}$.

In the normal case, $y_{t,i_j}$ is non-differentiable on $B_{t,i_1}, B_{t,i_2}, \ldots, B_{t,i_q}$. Therefore, we construct a novel differentiable ranking function $y_{t,i}^q = q + 1 - \sum_{j=1}^{q} (0.5 + \tanh(d[B_{t,i_j} - B_{t,i_q}]) + 0.5) \; \text{to replace } y_{t,i_j}$, and the ranking loss thus becomes $\sum_{j=1}^{q} (1 - y_{t,i_j})^2$.

**Fig. 2** Ranking functions with various $d$. The points displayed by the dashed lines refer to our randomly generated $(p_{t,i_j})_{j \in [q]}$.

Here note that the idea of our differentiable ranking function is inspired by [10]. Ayman [10] customized an activation function in the form of multiple tanh functions, which approximates the step function with equidistant points. Here we modify the intercept representation according to different combinations for the point set so as to obtain our differentiable ranking loss which allows cut-off points with unequal distance. With our modification, the ranking loss can incorporate various prior knowledge and training factors to avoid negative transfer, besides the relative training difficulty and the performance of mutual evaluation between tasks, as stated below.

**Fig. 2** depicts the shapes of our adapted composite tanh function under different d. At the end of every epoch with length $K$, we can randomly select some tasks, measure the performance of the network which is originally trained for task $i$ on task $j$, and update $p_{t,i}$.

**[Expectation 2]** If task $i$ is easier to train, that is, with a smaller $\mathcal{L}(w_i)$, then we expect a smaller number of transfers from task t to task i. This is because if a task is easy-to-train, i.e., with a faster training progress than other tasks, then the number of transfer to that task from other hard-to-train (i.e., with slower training progress) tasks should intuitively be smaller. To achieve this, we define $l_i^p$ as $\{\mathcal{L}(w_1), \ldots, \mathcal{L}(w_{t-1}), L(w_{t+1}), \ldots, L(w_T)\}$. Then add the $- (b_t^\pi)_{j=1}^{q} l_i^p$ term and the corresponding differentiable ranking loss $\sum_{j \in [p]} (rank_{j}^{(1)} - y_{t,i_j})^2$ to Eq. (5), where $rank_{j}^{(1)}$ is the ranking of $\mathcal{L}(w_j)$ among $\mathcal{L}(w_1), \mathcal{L}(w_2), \ldots, L(w_T)$ and $y_{t,i_j}$ is the ranking of $B_{t,i_j}$ among $B_{t,i_1}, B_{t,i_2}, \ldots, B_{t,i_T}$.

Note that the reason to add the $- (b_t^\pi)_{j=1}^{q} l_i^p$ is that to minimize $- (b_t^\pi)_{j=1}^{q} l_i^p$, we need to maximize $(b_t^\pi)_{j=1}^{q} l_i^p$. The maximum $y_{t,i_j}^q$ corresponds to the scenario where larger $B_{t,i_j}$ is multiplied with larger $\mathcal{L}(w_i)$ and smaller $B_{t,i_j}$ is multiplied with smaller $\mathcal{L}(w_i)$, which perfectly matches our second expectation.

**[Expectation 3]** If task i is more similar to task t, then we expect a larger number of transfers from task t to task i. Denote the similarity between task i and task t as $s_{i,t}$ and $rank_{j}^{(2)}$ as the ranking of $s_{i,t}$ among $(s_{1,t}, s_{2,t}, \ldots, s_{T,t})$. Then in summary, the problem in Eq. (4) now becomes

$$(t, b_t^\pi) \leftarrow \arg \min_{t \in U, b_t^\pi} \left\{ \lambda_0 [(1 + \mu_1 \|b_t^\pi\|_1) \mathcal{L}(w_t) - \mu_2 (b_t^\pi)_{j=1}^{q} l_i^p] + \lambda_1 \sum_{s \in U - t} \|w_s - \sum_{j=1}^{i-1} B_{(\pi(j))} w_{\pi(j)} \|_2^2 + \lambda_2 \sum_{j \in [q]} (j - y_{t,i_j})^2 + \lambda_3 \sum_{j \in [p]} (rank_{j}^{(1)} - y_{t,i_j})^2 + \lambda_4 \sum_{j \in [p]} (rank_{j}^{(2)} - y_{t,i_j})^2 \right\}. \quad (6)$$
In order to optimize the objective in Eq. (6), we first fix $b_t'$ and select task $t$ with the minimum objective in Eq. (6). Then fix $t$ and apply the Frank-Wolfe algorithm to optimize $b_t'$ with a convergence guarantee. Finally, fix $b_t'$ and train task $t$ with the policy loss Eq. (1) for $K$ episodes.

**Remark** Similarity measures for expectation 3. Regarding the expectation 3 for our customized ranking loss function, we take the following three similarity measures, respectively: a) the cosine similarity between the critic networks; b) the cosine similarity between the policy networks; c) the negative embedding distance in the state space, which will be explained in the Discussion Section. Among them, the first measure has the best performance in initial trials and hence we utilize this measure in all our experiments.

As for the negative embedding distance between task $i$ and task $j$, denote the state space as $S$ and the embedding distance between task $i$ and task $j$ is calculated as follows: (i) Sample 100 states, $\{s_1, s_2, \ldots, s_{100}\}$, from $S$ uniformly at random. (ii) Denote the embeddings of $\{s_1, s_2, \ldots, s_{100}\}$ calculated by the embedding layers of task $i$ and task $j$ as $\{e_{i, 1}, e_{i, 2}, \ldots, e_{i, 100}\}$ and $\{e_{j, 1}, e_{j, 2}, \ldots, e_{j, 100}\}$, respectively. (iii) Then, $-\frac{1}{100} \sum_{i=1}^{100} ||e_{i, m} - e_{j, m}||^2$ is just the negative embedding distance between task $i$ and task $j$.

### 3.4 Loss Optimization

**Optimization of $b_t'$.** To make the loss differentiable and ensure the sparsity of the transfer matrix $B$, we replace $\mu_t ||b_t'||_1$ in Eq. (6) with $\mu_t \sum_{j \in \mathcal{T} - \{t\}} B_{tj}$ and add the $b_t' \geq 0$, $||b_t'||_2 \leq \text{radius}$ constraint to Eq. (6), where $\text{radius}$ is the upper bound of $||b_t'||_1 (t \in \{T\})$ that we set in experiments. Define $f(b_t') = \lambda_0 \left[ \sum_{j \in \mathcal{T} - \{t\}} B_{tj} \Sigma(w_j) - \mu_t \sum_{j \in \mathcal{T}} b_{tj}^2 + \lambda_1 \sum_{s \in \mathcal{T} - \{t\}} ||w_s - \sum_{j=1}^{j-1} B_{s, j} w_{u_j} + B_{s, j} w_{tj}||^2 + \lambda_2 \sum_{j \in \mathcal{T} \setminus \{t\}} (j - y_{tj})^2 + \lambda_3 \sum_{j \in \mathcal{T} \setminus \{t\}} (\text{rank}(j) - y_{tj})^2 + \lambda_4 \sum_{j \in \mathcal{T} \setminus \{t\}} (\text{rank}(j) - y_{tj})^2 \right]$.

Then, we choose the task with the biggest complexity for each selected environment and form the 9 tasks that we conduct experiments on with regard to Gym-minigrid.

**Meta-World** [36] is a simulated benchmark for meta reinforcement learning and multi-task learning, containing 50 tasks related to robotic manipulation. In Meta-World, $MT_1$, $MT_{10}$, and $MT_{50}$ are multi-task-RL environments that have 1, 10, and 50 tasks, respectively.

The tasks for $MT_{10}$ from Task 1 to Task 10 are ‘Pick and place,’ ‘Pushing,’ ‘Reaching,’ ‘Door opening,’ ‘Button press,’ ‘Peg insertion side,’ ‘Window opening,’ ‘Window closing,’ ‘Drawer opening,’ and ‘Drawer closing,’ respectively.

The tasks for $MT_{50}$ from Task 1 to Task 50 are ‘Turn on faucet,’ ‘Sweep,’ ‘Stack,’ ‘Unstack,’ ‘Turn off faucet,’ ‘Push back,’ ‘Pull lever,’ ‘Turn dial,’ ‘Push with stick,’ ‘Get coffee,’ ‘Pull handle side,’ ‘Basketball,’ ‘Pull with stick,’ ‘Sweep into hole,’ ‘Disassemble nut,’ ‘Place onto shell,’ ‘Push mug,’ ‘Press handle side,’ ‘Hammer,’ ‘Slide plate,’ ‘Slide plate side,’ ‘Press button wall,’ ‘Press handle,’ ‘Push handle,’ ‘Soccer,’ ‘Retrieve plate side,’ ‘Retrieve plate,’ ‘Close drawer,’ ‘Press button top,’ ‘Reach,’ ‘Press button top w/ wall,’ ‘Reach with wall,’ ‘Insert peg side,’ ‘Push,’ ‘Push with wall,’ ‘Pick & place w/wall,’ ‘Press button,’ ‘Pick & place,’ ‘Pull mug,’ ‘Unplug peg,’ ‘Close window,’ ‘Open window,’ ‘Open door,’ ‘Close door,’ ‘Open drawer,’ ‘Open box,’ ‘Close box,’ ‘Lock door,’ ‘Unlock door,’ and ‘Pick bin,’ respectively.

### 4 Experiments

We compare our CAMRL with existing state-of-the-art algorithms on the Gym-minigrid, Meta-world, Atari games, Ravens, and RLbench, which are the benchmarks for multi-task RL widely used in the community [4], [8], [36].

#### 4.1 Environments

**Gym-Minigrid**: Gym-minigrid environments [8] are partially observable grid environments consisting of tasks with increasing complexity levels. In Gym-minigrid, the agent should find a key to open a locked door in a grid. By setting different obstacles in the way and adding different requirements, we obtain different environments, such as DistShift, Doorkey, DynamicObstacles, etc. Each environment can be easily tuned in terms of size/complexity, which contributes to fine-tuning the difficulty of tasks and performing curriculum learning. In our experiments, we first randomly select 9 environments, each of which owns more than 3 tasks with different complexities. Then, we choose the task with the biggest complexity for each selected environment and form the 9 tasks that we conduct experiments on with regard to Gym-minigrid.

**Meta-World** [36] is a simulated benchmark for meta reinforcement learning and multi-task learning, containing 50 tasks related to robotic manipulation. In Meta-World, $MT_1$, $MT_{10}$, and $MT_{50}$ are multi-task-RL environments that have 1, 10, and 50 tasks, respectively.

The tasks for $MT_{10}$ from Task 1 to Task 10 are ‘Pick and place,’ ‘Pushing,’ ‘Reaching,’ ‘Door opening,’ ‘Button press,’ ‘Peg insertion side,’ ‘Window opening,’ ‘Window closing,’ ‘Drawer opening,’ and ‘Drawer closing,’ respectively.

The tasks for $MT_{50}$ from Task 1 to Task 50 are ‘Turn on faucet,’ ‘Sweep,’ ‘Stack,’ ‘Unstack,’ ‘Turn off faucet,’ ‘Push back,’ ‘Pull lever,’ ‘Turn dial,’ ‘Push with stick,’ ‘Get coffee,’ ‘Pull handle side,’ ‘Basketball,’ ‘Pull with stick,’ ‘Sweep into hole,’ ‘Disassemble nut,’ ‘Place onto shell,’ ‘Push mug,’ ‘Press handle side,’ ‘Hammer,’ ‘Slide plate,’ ‘Slide plate side,’ ‘Press button wall,’ ‘Press handle,’ ‘Push handle,’ ‘Soccer,’ ‘Retrieve plate side,’ ‘Retrieve plate,’ ‘Close drawer,’ ‘Press button top,’ ‘Reach,’ ‘Press button top w/ wall,’ ‘Reach with wall,’ ‘Insert peg side,’ ‘Push,’ ‘Push with wall,’ ‘Pick & place w/wall,’ ‘Press button,’ ‘Pick & place,’ ‘Pull mug,’ ‘Unplug peg,’ ‘Close window,’ ‘Open window,’ ‘Open door,’ ‘Close door,’ ‘Open drawer,’ ‘Open box,’ ‘Close box,’ ‘Lock door,’ ‘Unlock door,’ and ‘Pick bin,’ respectively.
Atari Learning Environment (ALE) was first proposed in [4] which includes exploration, planning, reactive play, and complex visual input [12]. In our experiments, we randomly select 10 tasks with visual observations in ALE to perform experiments, namely, YarsRevenge, Jamesbond, FishingDerby, Venture, DoubleDunk, Kangaroo, IceHockey, ChopperCommand, Krull, and Robotank.

Ravens. Ravens is a vision-based robotic manipulation environment based on PyBullet. We follow [38] to select 10 typical discrete-time tabletop manipulation tasks in Ravens, namely, Block-insertion, Place-red-in-green, Towers-of-Hanoi, Align-box-corner, Stack-block-pyramid, Palletizing-boxes, Assembling-kits, Packing-boxes, Manipulating-rope, and Sweeping-piles. Similar to [38], we generate 1,000 expert demonstrations for each task by the oracle provided by Ravens and use these demonstrations to perform imitation learning for each adopted baseline. After the imitation learning phase, we train each baseline without demonstrations for another 10,000 epochs and compare the average success rate of each baseline.

RLBench. RLBench is a large-scale environment designed to speed up vision-guided manipulation research. We follow [21] to test our selected baselines on 10 typical RLBench tasks, i.e., Reach Target, Push Button, Pick And Lift, Pick Up Cup, Put Knife on Chopping Board, Take Money Out Safe, Put Money In Safe, Pick Up Umbrella, Stack Wine, and Slide Block To Target.

4.2 Baselines

To demonstrate the efficiency of the proposed CAMRL algorithm, we utilize actor critic (AC) [17], soft actor critic (SAC) [15], mastering rate based online curriculum learning (MRCL) [30], YOLOR [29], distral [28], gradient surgery [35], and soft module [32] as our baselines for comparison. The criteria of choosing baselines depend on whether they are commonly-used or state-of-the-art methods.

Among the above algorithms, regarding the mastering rate based online curriculum learning (MRCL), [30] made the assumption that the good next tasks are those that are learnable but not learned thoroughly yet, and thus introduced a new algorithm based on the notion of mastering rate so as to choose which task to train next in an online manner. Due to the fact that the rewards of tasks in gym-minigrid are pretty sparse, we customize a curriculum by the size of grid for each task, apply MRCL to switch the sub-tasks in each curriculum from the micro perspective, and utilize other baselines to perform multi-task or single-task training for all the tasks in the macro aspect.

4.3 Evaluation Metric

We adopt the reward metric in the Gym-minigrid, meta-world, and ALE environments and the success rate metric in the Ravens and RLBench environments. Although with regard to the meta-world environments, the success rate has been utilized in [32], [35], this evaluation metric can be zero often, which leads to sparse feedback and is bad for comparing the difficulties of training different tasks. Besides, [32], [35] displayed performance of their algorithms with the averaged success rate of all tasks instead of individual success rates for each task, which makes the success rate less informative than the reward metric for each task that we adopt in this paper. Last but not the least, the simulation steps in [35], the number of samples in [32], and the number of epochs in this paper are not in the same magnitude, which makes our reference for the success rate in [32], [35] less meaningful. As a result, in the meta-world environment, we use reward instead of the averaged success rate to formulate the evaluation metric, the policy loss in $I_{\text{trals}}$ and the composite loss terms.

4.4 Implementation Details

We adopt the same network structure for each single actor network and single critic network in each baseline. For soft module and gradient surgery, we follow the same parameters as in their public codes. For distral, we follow [28] to set $a = 0.5$ and use a set of 9 hyper-parameters $(1/\beta, \epsilon) \in \{3 \cdot 10^{-4}, 10^{-3}, 3 \cdot 10^{-3}\} \times \{2 \cdot 10^{-4}, 4 \cdot 10^{-4}, 8 \cdot 10^{-4}\}$ for the entropy costs and the initial learning rate. Hyper-parameters with the optimal result are utilized with regard to the distral algorithm.

Concerning our CAMRL, we set $a = 1000, b = \frac{1}{30}, c = 2, d = 200, \mu_1 = \mu_2 = 0.01$, and $radius = 0.05$ in all tasks. At the end of every epoch, instead of performing mutual test for each pair of tasks which is computationally-inefficient, we 1) randomly select two tasks $i$ and $j$; 2) test the performance of $SAC_i$ on task $j$ and the performance of $SAC_j$ on task $i$; 3) update $p_{i,j}$ and $p_{j,i}$; and 4) repeat the above mutual test procedure for 3 times. Besides, to avoid the abrupt increase in loss when switching to the curriculum-based AMTL mode, the actual loss for task $t$ is set to be $L(w_t) + 0.01 \times 0.01$ times the objective in Eq. (1). In addition, the proposed algorithms are run over 5 seeds to report the averaged results.

4.5 Results

Results on Gym-Minigrid. As shown in Table 1, among the cooperation between mastering rate based online curriculum learning (MRCL) [30] and the single-task actor critic/distral/gradient surgery/CAMRL algorithm, CAMRL achieves the best overall performance over the 9 minigrid tasks, without showing obvious signs of negative transfer.

Results on MT10 and MT50. As can be seen in Table 2, CAMRL significantly beats all baselines over 6 tasks out of the 10 tasks in MT10. The second-best algorithm overall is single SAC and is obviously worse than CAMRL in 7 tasks out of all 10 tasks. In Table 3, the best and second-best algorithms overall are CAMRL and soft module, respectively. The latter is at least 20% worse than CAMRL in 32 tasks out of the total 50 tasks and at least 20% better than CAMRL in 8 tasks out of all tasks. To sum up, in existing experiments, our CAMRL works well whether the task scale is large or small. Moreover, it seldom exposes serious negative transfer, and can sometimes contribute greatly to the tasks that are hard to train for other baselines.

Results on Atari, Ravens, and RLBench. See the performance of all algorithms on the selected Atari, Ravens, and RLBench tasks in Tables 4, 5, and 6. From Tables 4, 5, and 6, we can see that our CAMRL algorithm ranks first in 6 tasks out of the total selected 10 Atari tasks, 8 tasks out of the total selected 10 Ravens tasks, and 9 tasks out of the total selected
we can see that the magnitude of each is the identity matrix in by the order from mini-
the better evaluation performance that PR (0.02) 0.02 (0.01) 0.01 (0.01) 0.02 (0.01)
(10.51). The visualization To demonstrate the switch-
(0.02) 2½ 1.17
(5.39) -50.36 (2.55)
/C2 1329.39
213.27 (3.81)
I 0.95
asymmetric matrix representing the number of trans-
I 0.87
is small enough, say no
I (2049.4) 23.64 (3.25)
has on task I, the performance matrix of mutual evalua-
0.51
I 0.61
As stated in Section
52.91
matrix
10
2½
does not differ much and
0.03)
tends to perform a smaller number of transfers
2½
is easier to train, that is, with better public performance,
I (0.02) 0.02
I (0.04)
I (0.02)
I (0.05)
I 0.13
2539.78
I 0.89
-38.36 (0.02)
I 0.95
2½
is also obvi-
I 0.87
0.89
I 0.61
is also small

| TABLE 1 | Averaged Reward for Gym-Minigrid Tasks (Each Experiment Repeated 5 Times, the Average Scores With Standard Errors (in Brackets) Reported) |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| Single AC | Single AC+MRCL | YOLOR-SAC+MRCL | Distral+MRCL | Gradient Surgery+MRCL | CAMRL+MRCL | CAMRL(ours) |
| DistShiftEnv | 0.05 (0.02) | 0.78 (0.05) | 0.09 (0.03) | 0.07 (0.04) | 0.08 (0.02) | 0.95 (0.03) |
| DoorKeyEnv16x16 | 0.00 (0.00) | 0.08 (0.03) | 0.01 (0.01) | 0.00 (0.00) | 0.01 (0.01) | 0.13 (0.02) |
| DynamicObstaclesEnv16x16 | -0.99 (0.03) | -0.83 (0.04) | -0.88 (0.03) | -0.98 (0.01) | -0.97 (0.02) | -0.01 (0.02) |
| EmptyEnv16x16 | 0.10 (0.04) | 1.10 (0.04) | 0.12 (0.03) | 0.08 (0.03) | 0.11 (0.03) | 1.17 (0.05) |
| FetchEnv | 0.07 (0.03) | 0.83 (0.05) | 0.14 (0.03) | 0.09 (0.02) | 0.09 (0.03) | 0.89 (0.04) |
| KeyCorridor | 0.01 (0.01) | 0.02 (0.01) | 0.09 (0.03) | 0.02 (0.01) | 0.01 (0.01) | 0.08 (0.02) |
| LavaCrossing59N3Env | 0.02 (0.01) | 0.13 (0.02) | 0.02 (0.01) | 0.01 (0.01) | 0.02 (0.01) | 0.04 (0.01) |
| LavaGap57Env | 0.03 (0.02) | 0.53 (0.04) | 0.21 (0.02) | 0.03 (0.01) | 0.05 (0.02) | 0.87 (0.02) |
| MemoryS17Random | 0.02 (0.01) | 0.61 (0.04) | 0.13 (0.02) | 0.04 (0.02) | 0.20 (0.04) | 0.20 (0.03) |

The best results are bolded. The higher the metric, the better performance of the model.

10 RLBench tasks. No significant negative transfer is found when performing CAMRL in all these tasks.

Analysis on the Transfer Matrix B. As stated in Section 3, B is a \( T \times T \) asymmetric matrix representing the number of transfers between tasks. Fig. 3a visualizes the \( B = I^{10 \times 10} \) matrix where \( I^{10 \times 10} \) is the identity matrix in \( \mathbb{R}^{10 \times 10} \), and Fig. 3c visualizes the \( B = I^{10 \times 10} \) matrix of the first 10 tasks in MT50.

To discover some in-depth rules related to \( B \), we calculate \( PM \in \mathbb{R}^{T \times T} \), the performance matrix of mutual evaluation across tasks, and compare its variant \( PR \), a performance rank matrix, with \( B = I^{10 \times 10} \): First, test the average performance of the single SAC net trained for task \( s \) over 10 \( k \) episodes on task \( t \), and regard it as \( PM_{s,t} \) (\( s,t \in [T] \)) (the evaluation is run for 20 episodes); next, for each task \( t \in [T] \), rank \( PM_{s,t} \) by the order from minimum to maximum and regard the rank as the \( t \)-th column of the performance rank matrix \( PR \in \mathbb{R}^{T \times T} \). The visualization of \( PR \) for MT10 is shown in Figs. 3b, and 3d displays the \( PR \) for the first 10 tasks out of the 50 tasks in MT50. The bigger the number in \( PR_{s,t} \), the better evaluation performance that the SAC network originally trained for task \( s \) has on task \( t \). As shown in Fig. 3, when \( PR_{s,t} \) is small enough, say no larger than 2 (see Figs. 3b and 3d), \( B_{s,t} = I^{T \times T} \) is also obviously smaller than other elements (see Figs. 3a and 3c). This, to some extent from the opposite side, confirms our previous demand that the better evaluation performance on other tasks, the larger number of transfers on those tasks.

Incorporation of Prior Knowledge. To show the capability of incorporating prior knowledge for CAMRL, we perform extra experiments that formulate a new differentiable ranking loss for tasks where the relative magnitudes of the task difficulty are readily apparent in part. Specifically, we follow [26], [37], [38], and [21] to obtain the public performance of existing state-of-the-arts methods for MT10, MT50, Atari, Ravens, and RLBench, respectively. Then we use the relative ranking of the public performance to formulate a new \( \text{tanh} \)-based differentiable ranking loss and incorporate it with Eq. (1), in the hope that for task \( t \in [T] \), if task \( i \) is easier to train, that is, with better public performance, then task \( t \) tends to perform a smaller number of transfers to task \( i \). The results of adding this new differentiable ranking loss term are shown in Tables 7, 8, 9, 10, and 11, available online. We can see that after adding the new loss term, the performance of CAMRL beats its original version over 5 tasks out of the 10 tasks in MT10, 38 tasks out of the 50 tasks in MT50, 8 tasks out of the 10 tasks in Atari, 7 tasks out of the 10 tasks in Ravens, and 9 tasks out of the 10 tasks in RLBench, which demonstrates CAMRL’s capability of incorporating prior knowledge as well as the great performance of the differentiable ranking loss in CAMRL.

Changes of Each Term in \( I_{mul} \). To demonstrate the switching process of CAMRL’s training mode and visualize the magnitude change of each term in \( I_{mul} \), we plot the changes of \( I_{mul} \) and its three terms for CAMRL in the MT10 environment. From Fig. 4 we can see that the magnitude of each term in \( I_{mul} \) does not differ much and \( I_{mul} \) keeps ranging from 0.2 to 0.8, which does not show abnormal fluctuation and magnitude much.

Hyper-Parameter Analysis. The results of the hyper-parameter analysis for our CAMRL on MT10 are shown in Fig. 8.
available online. We first set $\lambda_0 = 0$, $\mu_1 = \mu_2 = \lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 0.01$, $a = 1000$, $b = \frac{1}{40}$, $c = 2$, $d = 200$, and radius = 0.05. And the evaluation metric is defined as the average reward of all 10 tasks over the first 10k episodes by the CAMRL algorithm. Then, we

(i) fix $\mu_2$, $\lambda_1$, $\lambda_2$, $\lambda_3$, $\lambda_4$, $a$, $b$, $c$, $d$, and radius. Set parameter $\mu_1 = 0$, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, respectively. The results under different values of $\mu_1$ are shown in Fig. 8 (line $\mu_1$);

(ii) repeat the same procedure for $\lambda_i (i \in [4])$, $a$, $\frac{1}{4}$, $c$, $d$, and $\mu_2$ as that for $\mu_1$.

From Fig. 8a we can see that, as for $\mu_1 (i \in [2])$ and $\lambda_i (i \in [4])$, when all of these hyper-parameters are around 0.01, decreasing one parameter among them to 0 may hurt the performance. In the meantime, if one of the parameters is increased too much, the great performance of CAMRL may not be guaranteed, which may result from the abrupt change of loss when switching the training mode and from the exceeded negative transfer. In summary, according to the above experimental evaluation, as for the time-invariant design of $\lambda_i (i \in [4])$, a reasonable setting, e.g., $\lambda_i \approx 0.01$, $\mu_i \approx 0.01$ for $i \in [4], j \in [2]$, is indeed helpful in improving SAC's

---

**TABLE 3**

| Task               | Single AC | Single SAC | YOLOR-SAC | Distral | Gradient Surgery | Soft Module | CAMRL(ours) |
|--------------------|-----------|------------|-----------|---------|------------------|-------------|-------------|
| Turn on faucet     | 201200.24 | 101600.24  | 091600.24 | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Sweep              | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Stack              | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Unstack            | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Turn off faucet    | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Push back          | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Pull lever         | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Turn dial          | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Pull with stick    | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Get coffee         | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Pull handle side   | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Basketball         | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Pull with stick    | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Sweep into hole    | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Disassemble nut    | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Place on shelf     | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Press push button  | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Press handle       | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Slide plate        | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Slide plate side   | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Pull handle        | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Pull with handle   | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Press button top   | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Reach              | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Reach with wall w/| 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Pick & place w/wall| 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |
| Press button       | 101600.24 | 091600.24  | 91600.24  | 91600.24 | 91600.24         | 91600.24    | 91600.24    |

The higher the metric, the better performance of the model.
In fact, due to our elaborate design, each term in the ranking function, and if one of the terms reaches zero, the performance of CAMRL will not deteriorate too quickly when performing a hyper-parameter analysis on $a$, $b$, and $c$.

**Ablation Study.** For the ablation study, on tasks MT10, we test our method in the lack of the mode switching component (i.e., entirely performing curriculum learning), and when setting $\frac{1}{\lambda_i}$ of $\lambda_i$ to be 0, respectively. Also, we test the performance of AMRL which lacks the mode switching mechanism and multiple ranking functions compared with our CAMRL. As shown in Table 12, the lack of any component
of our CAMRL and setting $\frac{1}{b}/b/c/d/\lambda_i (i = 2, 3, 4)$ to be 0 or positive infinity (so that the corresponding term in $I_{mul}$ is 0) bring varying degrees of reduction in algorithm performance, which consistently demonstrates the precision of algorithm design and the importance of tacit cooperation among different components. The obvious performance deterioration when only performing AMRL and when lacking the module switching component specifically shows the effectiveness and efficiency of our design of the index $I_{mul}$ and the customized differentiable ranking functions.

5 DISCUSSION

In this section, we discuss the advantages of CAMRL over existing MTRL algorithms, some clarifications of CAMRL, and direct ways to improve CAMRL.

5.1 Advantages Over Existing MTRL Algorithms

When there exist many tasks, it is difficult to find common traits owned by all tasks, and thus, the distillation-like algorithms may perform poorly. On the contrary, except for distilling common traits of all tasks by a large network, our $B$ matrix probes the transfer relationship between every two tasks and enables us to perform asymmetric transfer between tasks so that CAMRL will not be troubled by this issue.

For multi-task learning with a modular paradigm, negative transfer can be serious since the task relationship in the early period is incorrect and the bad influence may accumulate. Moreover, it may take a long time to approximately capture the correct task relationship and by the time the relationship is well learned, while the effect caused by negative transfer might have already been irreversible. On the contrary, by initializing $B$ as a unit matrix (no transfer at the beginning), adding constraints to $B$, and switching between various training modes, CAMRL can not only mitigate negative transfer, but also learn a good $B$ at a relatively early stage, allowing the training on difficult tasks to be positively affected by other tasks.

We summarize the advantages of CAMRL as five-fold. First, CAMRL is flexible: (i) CAMRL can freely switch between parallel single-task training and curriculum-based AMTL modes according to the indicator related to learning progress. (ii) The sub-components of CAMRL have variants and can be paired with different RL baseline algorithms and training modes. CAMRL does not impose restrictions on the network architecture and learning mode. For example, $W$ could be parameters of the whole network. When there is a large number of tasks, we could share the principle part of the whole network and let $W$ be only a small subset of the network in order to save RAM. Second, CAMRL is promising in mitigating negative transfer. For one thing, by considering the 1-norm constraint on $B$ and applying Frank-Wolfe to meet the constraint, CAMRL can reduce negative transfers to some extent. For another, by customizing and utilizing indicators about the learning progress and the performance of mutual tests across tasks, CAMRL can alleviate negative transfer. Third, the loss function of CAMRL can take information from multiple aspects to improve the efficiency of RL tasks’ training. Specifically, the loss function in Eq. (4) considers the positivity and sparsity of the transfer matrix, the training difficulty for different tasks, the performance of mutual tests, and the similarity between every two tasks. Thanks to our customized ranking loss which enables to absorb partial ranking information, CAMRL could take full advantage of various existing prior knowledge and training factors, regardless of their amount. Fourth, CAMRL eliminates the need for laborious hyper-parameter analysis. Specifically, CAMRL allows the hyper-parameters in the composite loss to dynamically auto-adjust and regards the auto-adjusting process as a multi-task learning problem. An uncertainty-based multi-task learning method is utilized to update the hyper-parameters automatically. Fifth, CAMRL can be auto-adapted to a new task without affecting original tasks by changing $B$ to $B' = [B, (0, \ldots, 0); (0, \ldots, 0), 1]$.

5.2 Some Clarifications

Computation Amount of Mutual Evaluation. If the mutual evaluation between tasks requires a big amount of computation, we could reduce the frequency of mutual evaluation between tasks and reuse the evaluation results between each evaluation. Besides, in each epoch, we could just randomly select a few tasks for mutual evaluation and only conduct partial ranking with regard to the ranking loss.
Complexity of the Loss Term. Although our customized loss contains multiple terms which might lead to strenuous hyper-parameter tuning, we adapt the automatic hyper-parameter adjustment technology in [16] to handle the above issue. Moreover, in the future, it might be valuable to discover other factors that make a more significant difference to CAMRL and simplify the loss term by replacing the existing factors in the loss with the most remarkable factors.

Performances of the Trade-Off Mechanism Among Differentiable Loss Functions. Although we adopt differentiable loss functions in order to keep the ranking alignment between $B_{i,t}(i \in [T])$ and $p_{i,t}(\mathcal{L}(u_t))$, there are multiple terms in the loss and thus we have to make a compromise between different terms. As a result, it is difficult to achieve perfect alignment concerning each ranking loss, and to mitigate negative transfer we mainly hope that $B_{i,t}$ can be small when $p_{i,t}$ and $\mathcal{L}(u_t)$ are small enough. Actually, as shown in Fig. 3, when $PR_{i,t}$ is small enough, say no larger than 2, $B_{i,t} - I_{x,T}^d$ is also obviously smaller than other elements, which meets our expectation for relieving serious negative transfer.

Details of Integrating MRCL With the Baselines. Since the tasks we select from Gym-minigrid are of the biggest size or complexity, directly overcoming these tasks would be very difficult. Therefore, to lower the barrier of tasks’ learning, we follow the mastering rate based online curriculum learning (MRCL) algorithm [30] to perform curriculum learning for each selected task. Specifically, for each selected task $i$, we take task $i$ and other easier tasks in Gym-minigrid with the same property as task $i$ as the curriculum of task $i$. During the training phase of each task $i$, we adopt MRCL to learn the curriculum of task $i$ as quickly as possible, with the learning algorithm set to be our selected baseline. At each training step, MRCL adaptively adjusts the next task in the curriculum of task $i$ for training according to the dynamic attention for each task, under the assumption that the good next tasks are the ones which are learnable but not learned yet. Thanks to the delicate attention mechanism of MRCL, the learning algorithm could overcome the selected tasks in Gym-minigrid little by little instead of being stuck somewhere for a long time.

Configuration Method of Hyper-Parameters. (i) For hyper-parameters $a, b$, and $c$, we adjust them according to the magnitude of the variables in each item of the index $I_{mul}$; with the hope that the composite magnitude of each item in $I_{mul}$ does not differ too far. (ii) For the hyper-parameter $d$, we plot various ranking functions with different $d$ and select $d$ with a relatively smooth ranking function as well as being as small as possible, since a small $d$ could contribute to the convergence speed of vanilla Frank-Wolfe during the optimization of $h_t^d$. (iii) For hyper-parameters $\lambda_t(i \in \{0, 1, 2, 3, 4\})$, we automatically adjust them according to an uncertainty-based multi-task learning method in [16].

5.3 Potential Improvements

Below we list several directions for CAMRL to be improved, so as to illuminate further incorporation of curriculum-based AMTL into RL.

- Consider more auto-tuning approaches of hyper-parameters $\mu_t(j = 1, 2)$ and $\lambda_t(j = 1, 2, 3, 4)$, as well as allowing $\mu_t$ to be the time-varying function of the learning progress and other factors.
- Enrich the loss by integrating the intermediate difficulty, diversity, surprise and energy [25], etc.
- Design techniques to select the optimal subset of all the networks’ parameters as $W$.
- Transform $B$ into a non-linear function and update it with theoretical guidance.
- Apply the sliding window, the exponential discounted factor, the appropriate entropy, or other tricks to mitigate the non-stationarity during the AMTL training.

6 Conclusion and Future Work

In this paper, we proposed a novel multi-task reinforcement learning algorithm, called CAMRL (curriculum-based asymmetric multi-task reinforcement learning). CAMRL switches the training mode between parallel single-task and curriculum-based AMTL. CAMRL employs a composite loss function to reduce negative transfer in multi-task learning. Apart from regularizing tasks’ outgoing transfer and network weights’ similarity, we introduced three differentiable ranking functions into the loss to incorporate various prior knowledge flexibly. An alternating optimization with Frank-Wolfe has also been utilized to optimize the loss, and an uncertainty-based automatic adjustment mechanism of hyper-parameters has been adopted to eliminate laborious hyper-parameter analysis. We have conducted extensive experiments to confirm the effectiveness of CAMRL and have analyzed its flexibility from various perspectives. In the future, we plan to study CAMRL with more theoretical insights and consider its potential improvements listed in the Discussion Section, such as incorporating more prior knowledge into the loss, designing the non-linear version of the transfer matrix, and overcoming the non-stationarity during the AMTL training.

REFERENCES

[1] Constop-PyTorch: A library for constrained optimization built on PyTorch, 2020. [Online]. Available: https://github.com/GeoFlip/Constop-Pytorch
[2] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda, “Purposive behavior acquisition for a real robot by vision-based reinforcement learning,” Mach. Learn., vol. 23, no. 2/3, pp. 279–303, 1996.
[3] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, “Emergent complexity via multi-agent competition,” 2017, arXiv:171003748.
[4] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: An evaluation platform for general agents,” J. Artif. Intell. Res., vol. 47, pp. 253–279, 2013.
[5] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in Proc. 26th Annul. Int. Conf. Mach. Learn., 2009, pp. 41–48.
[6] E. Brunskill and L. Li, “Sample complexity of multi-task reinforcement learning,” 2013, arXiv:1309.6821.
[7] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, “GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 794–803.
[8] M. Chevalier-Boisvert et al., “BabyAI: A platform to study the sample efficiency of grounded language learning,” in Proc. Int. Conf. Learn. Representations, 2018.
[9] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, “Learning modular neural network policies for multi-task and multi-robot transfer,” in Proc. IEEE Int. Conf. Robot. Automat., 2017, pp. 2169–2176.
[10] A. Elgharabawy, “Preference neural network,” 2019, arXiv:1904.02345.
[11] J. L. Elman, “Learning and development in neural networks: The importance of starting small,” Cognition, vol. 48, no. 1, pp. 71–99, 1993.

[12] L. Espeholt et al., “IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 1407–1416.

[13] R. M. Freund and P. Grigas, “New analysis and results for the Frank–Wolfe method,” Math. Program., vol. 155, pp. 199–230, 2016.

[14] P. Gong, C. Zhang, Z. Lu, J. Huang, and J. Ye, “A general iterative shrinkage and thresholding algorithm for non-convex regularized optimization problems,” in Proc. Int. Conf. Mach. Learn., 2013, pp. 37–45.

[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in Proc. 35th Int. Conf. Mach. Learn., 2018, pp. 1856–1865.

[16] A. Kendall, Y. Gal, and R. Cipolla, “Multi-task learning using uncertainty to weigh losses for scene geometry and semantics,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2018, pp. 7482–7491.

[17] V. R. Konda and J. N. Tsitsiklis, “Actor-critic algorithms,” in Proc. Ann. Int. Conf. Inf. Syst. Proces., 2000, pp. 1008–1014.

[18] S. Lacoste-Julien, “Convergence rate of Frank-Wolfe for non-convex objectives,” 2016, arXiv:1607.00345.

[19] G. Lee, E. Yang, and S. Hwang, “Asymmetric multi-task learning based on task relatedness and loss,” in Proc. Int. Conf. Mach. Learn., 2016, pp. 230–238.

[20] H. B. Lee, E. Yang, and S. J. Hwang, “Deep asymmetric multi-task feature learning,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 2956–2964.

[21] S. Liu, S. James, A. J. Davison, and E. Johns, “Auto-lambda: Disentangling dynamic task relationships,” 2022, arXiv:2202.03091.

[22] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” 2017, arXiv:1706.06083.

[23] Y. E. Nesterov, “Complexity bounds for primal-dual methods minimizing the model of objective function,” Math. Program., vol. 171, no. 1/2, pp. 311–330, 2018.

[24] A. Pentina, V. Sharmanska, and C. H. Lampert, “Curriculum learning of multiple tasks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2015, pp. 5492–5500.

[25] R. Portelas, C. Colas, L. Weng, K. Hofmann, and P. Oudeyer, “Automatic curriculum learning for deep RL: A short survey,” in Proc. 29th Int. Joint Conf. Artif. Intell., 2020, pp. 4819–4825.

[26] C. Shenton, “Atari reinforcement learning leaderboard,” 2018.[Online]. Available: https://github.com/cshenton/atari-leaderboard.

[27] A. Taylor, I. Dusparic, M. Gueriu, and S. Clarke, “Parallel transfer learning in multi-agent systems: What, when and how to transfer?,” in Proc. Int. Joint Conf. Neural Netw., 2019, pp. 1–8.

[28] Y. Teh et al., “Distral: Robust multitask reinforcement learning,” in Adv. Neural Inf. Process. Syst., 2019, pp. 4496–4506.

[29] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, “You only learn one representation: Unified network for multiple tasks,” 2021, arXiv:2105.04206.

[30] L. Willems, S. Lahlou, and Y. Bengio, “Mastering rate based curriculum learning,” 2020, arXiv:2008.06456.

[31] Y. Wu and Y. Tian, “Training agent for first-person shooter game with actor-critic curriculum learning,” in Proc. 5th Int. Conf. Learn. Representations, 2017.

[32] R. Yang, H. Xu, Y. Wu, and X. Wang, “Multi-task reinforcement learning with soft modularization,” in Proc. 34th Int. Conf. Neural Inf. Process. Syst., 2020, Art. no. 400.

[33] D. Ye et al., “Towards playing full moba games with deep reinforcement learning,” in Proc. Adv. Neural Inf. Process. Syst., 2020, pp. 621–632.

[34] D. Ye et al., “Mastering complex control in moba games with deep reinforcement learning,” Proc. AAAI Conf. Artif. Intell., vol. 34, pp. 6672–6679, 2020.

[35] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, “Gradient surgery for multi-task learning,” in Proc. 34th Int. Conf. Neural Inf. Process. Syst., 2020, Art. no. 489.

[36] T. Yu et al., “Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning,” in Proc. Conf. Robot Learn., 2019, pp. 1–18.

[37] T. Yu et al., “Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning,” in Proc. Conf. Robot Learn., 2020, pp. 1094–1100.