Applied Intelligence (2020) 50:3301–3317
https://doi.org/10.1007/s10489-020-01702-7

Combining a gradient-based method and an evolution strategy for multi-objective reinforcement learning

Diqi Chen[1] · **Yizhou Wang**[2] · **Wen Gao**[3]

Published online: 1 June 2020

Abstract

Multi-objective reinforcement learning (MORL) algorithms aim to approximate the Pareto frontier uniformly in multi-objective decision making problems. In the scenario of deep reinforcement learning (RL), gradient-based methods are often adopted to learn deep policies/value functions due to the fast convergence speed, while pure gradient-based methods can not guarantee a uniformly approximated Pareto frontier. On the other side, evolution strategies straightly manipulate in the solution space to achieve a well-distributed Pareto frontier, but applying evolution strategies to optimize deep networks is still a challenging topic. To leverage the advantages of both kinds of methods, we propose a two-stage MORL framework combining a gradient-based method and an evolution strategy. First, an efficient multi-policy soft actor-critic algorithm is proposed to learn multiple policies collaboratively. The lower layers of all policy networks are shared. The first-stage learning can be regarded as representation learning. Secondly, the multi-objective covariance matrix adaptation evolution strategy (MO-CMA-ES) is applied to fine-tune policy-independent parameters to approach a dense and uniform estimation of the Pareto frontier. Experimental results on three benchmarks (Deep Sea Treasure, Adaptive Streaming, and Super Mario Bros) show the superiority of the proposed method.

Keywords Multi-objective reinforcement learning · Multi-policy reinforcement learning · Pareto frontier · Sampling efficiency

1 Introduction

Deep reinforcement learning (RL) algorithms have been successfully applied in many challenging decision making problems, such as video games [32, 33, 60], the game of Go [48, 50] and robotics [19, 20, 27, 49]. In these scenarios, only one objective is optimized. Nevertheless, many real-world decision making problems consider more than one objective. Network routing takes energy, latency, and channel capacity into account [35]. Medical treatment needs to release symptoms and minimize side effects [28, 29]. Economic systems are analyzed from both economic and ecological perspectives [57]. Furthermore, through adding extra reward signals that encode domain knowledge, reward shaping methods are adopted to encourage exploration of the agents [11, 36].

✉ Diqi Chen
cdq@pku.edu.cn

Extended author information available on the last page of the article.

The objectives of multi-objective reinforcement learning (MORL) are often conflicting, which means that maximizing one objective will normally lead to the minimization of the others [37]. Therefore, rather than learn only one optimal solution, MORL aims to achieve a set of mutually incomparable solutions. In this set, each solution is not dominated by all the other solutions. This non-dominated set is named as the Pareto frontier. And the non-dominated solution is called the Pareto optimal solution. However, approaching all Pareto optimal solutions is untrivial in complicated scenarios. As a compromise, MORL algorithms are expected to approximate the Pareto frontier as uniformly as possible [43].

Recently, gradient-based methods are popular in deep single-policy RL because of the capability to learn deep neural networks. Specifically, deep RL algorithms [19, 20, 27, 32, 33, 48–50, 60] adopt deep neural networks to approximate policies or value functions, which are optimized by gradient-based methods with fast convergence speed. To adapt single-objective RL methods to multi-objective RL problems, a naive approach is to run a single-objective single-policy deep RL algorithm independently

and repeatedly guided by scalarized rewards. However, pure gradient-based methods can not guarantee a uniformly approximated Pareto frontier. On the other side, evolution strategies are good at achieving a well-distributed shape of Pareto frontier, since these algorithms manipulate in the solution space straightly. But applying evolution strategies to optimize deep neural networks is still a challenging topic. To leverage the advantages of both kinds of methods, we propose a two-stage MORL framework combining a gradient-based method and an evolution strategy. This work builds on and gives a more detailed description and discussion of our preliminary work [13]. The new contributions include extending the work to a reward shaping task, adding a more exhaustive discussion and applying more ablation studies to investigate and explain this framework.

At the first stage, the soft actor-critic (SAC) algorithm [19, 20] is extended to a multi-policy soft actor-critic (MPSAC) algorithm. Rather than naive independent repetition on a single-objective algorithm, the proposed model collaboratively learns a group of policy networks. Each policy targets on a specific scalarized objective. The extension is based on the assumption that exploration efficiency can be improved by collaboratively learning multiple policies with different targets. In this work, the linear weighted scalarization method is adopted. During training, the model maintains a multi-channel replay buffer. Each channel is related to a weight vector for scalarization. The collaboration is conducted through replay buffer sharing. Furthermore, to reduce the redundancy of model representation and computation, all policy networks share the same low-level parameters. The first-stage learning can be regarded as representation learning, which shows the potential for better performance from some previous works [9, 10, 17].

The proposed MPSAC algorithm is also inspired by the bootstrapped DQN [38], which maintains more than one deep networks for Q-value estimation. The Q-networks in bootstrapped DQN are optimized for the same objective. To improve exploration efficiency, the randomness of model learning is promoted by randomized initialization and randomized training samples. In our method, policy networks pursue different scalarized objectives. Hence, the model randomness is naturally sustained to achieve high exploration efficiency.

At the second stage, the multi-objective covariance matrix adaptation evolution strategy (MO-CMA-ES) is applied to fine-tune the policy-independent parameters. The policy-independent parameters are vectorized as the chromosomes of individuals. The individuals of the first generation are initialized from the policies learned at the first stage. In each generation, every individual generates one offspring according to its covariance matrix. The new offsprings are evaluated and marked by a fitness score vector. Then all individuals are ranked and half elitist individuals are selected as the parents of the next generation. The ranks are determined by the non-dominated ranks at first. Then, in each non-dominated set, the crowding distance is utilized as the second ranking criterion to approximate the Pareto frontier uniformly. Multi-objective methods have been applied to evolutionary algorithms to improve performance [3, 6], in which the multiple objectives are considered for a better random search methodology. The motivation is that different objectives may work in different scenarios, then pursuing multiple objectives can enrich the searching process. In our scenario, the algorithm is designed straightly for the multi-objective environment, so the ideas are similar but there is a more explicit goal for our method, namely, to learn multiple policies to approximate the Pareto frontier with high efficiency.

The proposed framework can be applied in both Markovian and non-Markovian environments. Specifically, in the non-Markovian situation, recurrent neural networks (RNN) are adopted as the policy networks. The recurrent connections are learned at the first learning stage and fixed at the second stage. Previous works [47, 52] adopt evolution strategies and neuroevolution to train deep neural networks, but training RNNs are not considered in their frameworks. Safe mutations [26] is proposed to learn RNN policies, while this method needs to compute gradients during the neuroevolution algorithm.

The principal contribution of this paper is as follows.

1. A multi-policy gradient-based deep reinforcement learning algorithm with high sampling efficiency. This algorithm simultaneously learns multiple policies which collaborate through buffer sharing.
2. A multi-objective reinforcement learning framework approaching a well-distributed Pareto frontier. This framework leverages the advantages of gradient-based methods and evolutionary methods and alleviates the disadvantages of both types of methods.
3. The proposed framework learns compact model representation with lower computational complexity.
4. The proposed framework can be applied in the non-Markovian environment to train deep recurrent neural networks efficiently.

Experimental results on three benchmarks (Deep Sea Treasure, Adaptive Streaming and Super Mario Bros) show the superiority of our proposed method.

2 Related work

Multi-objective reinforcement learning methods can be divided into single-policy methods and multi-policy

methods based on the number of the learned policies in a single run.

Single-policy methods learn only one policy at each time. It needs to be repeated many times to approximate the Pareto frontier. Commonly, scalarization methods are adopted to transfer multiple objectives into a single one. Then classical single-policy RL algorithms can be applied without any modification. The simplest scalarization is the linear weighted scalarization [37, 57, 58]. The limitation of linear scalarization is demonstrated in [59]. The learned policies can not converge to the solutions on the concave region of the Pareto frontier. The Chebyshev scalarization method [62] and the thresholded scalarization method [37, 58] are proposed to alleviate this limitation. Instead of simple repetition, Parisi et al. [43] proposed Pareto path following algorithm optimizing directly on the Pareto frontier.

Multi-policy methods learn more than one policy in a single run. Van et al. [61] extended Q-learning to a multi-policy method using hypervolume metric as an action selection strategy. Beyond discrete solutions, some methods approach continuous approximation. Barrett et al. [8, 12, 28, 29] introduced an approach which learns the set of optimal policies for all weights through involving linear weights to represent value functions. Parisi et al. [42] proposed a method, which utilizes a function to define a manifold in the space of policy parameters, then approximates the Pareto frontier continuously by performing a single gradient ascent on the parameters of the function.

Furthermore, some studies considered MORL problems from other perspectives. Mannor and Shimkin [30] consider MORL problems as a constrained optimization problem. In the constrained problem, rather than optimize a single objective, the method drives the agent to approach a target set. Wiering et al. [64] proposed a model-based MORL method through learning a multi-objective model of the environment. Tajmajer [56] leveraged multiple DQNs through forming a hierarchy of control then combines outputs of several DQNs into a single action.

3 Preliminaries

Markov Decision Process (MDP) In a single-objective RL problem [54], an agent learns from interacting with the environment to achieve a goal. The problem can be modeled as a Markov Decision Process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$. At each time step t, the agent receives the current state $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}$ according to a stochastic policy $\pi(a_t|s_t)$. Then the agent observes a reward $r_t = R(s_t, a_t) \in \mathcal{R}$ and transitions to a new state s_{t+1}. The transition is based on the transition probability $p(s_{t+1}|s_t, a_t)$ which is determined by the environment. The goal is to learn a policy to maximize an expectation over the discounted accumulated reward $\mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t r_t]$, where $\gamma \in [0, 1]$ is a discounted factor. $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ denote the state space, the action space and the reward space, respectively. Following a specific policy π, $\rho_\pi(s, a) = \mathbb{E}_\pi[\frac{1}{T}\sum_{t=1}^{T} \mathbb{1}(s_t = s, a_t = a)]$ denotes the state-action marginal distribution, which measures the probability of state-action pairs being visited in a finite-length episode. In this work, the considered state spaces are continuous and the action spaces are discrete.

Multi-Objective Markov Decision Process (MOMDP) In an MOMDP, at each time step, the agent receives a vector reward $\boldsymbol{r}_t = \boldsymbol{R}(s_t, a_t) \in \mathcal{R}^M$, where M is the number of the objectives. Hence, each policy π is associated with an expectation over the discounted accumulated reward vector $\mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t \boldsymbol{r}_t]$. The goal is to learn a set of non-dominated solutions $\Pi^\star = \{\pi^\star\}$, each of which is not dominated by any other solutions. This non-dominated set is named as the Pareto frontier.

Soft Actor-Critic (SAC) algorithm SAC algorithm [19, 20] is utilized as our baseline single-policy RL algorithm. To make this paper as self-contained as possible, SAC is described elaborately. SAC is a policy iteration method designed on a maximum entropy RL framework. The objective function is:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))], \tag{1}$$

where $\mathcal{H}$ denotes the entropy of the policy and α denotes the temperature parameter. During learning, SAC updates the soft Q-function and the soft policy iteratively. The soft Q-function is defined as $Q(s_t, a_t)$, which is approximated by a neural network, called Q-network. Under this framework, the Bellman update of the Q-network is processed by minimizing the square error of the estimated Q-value and the target value $\bar{Q}(s_t, a_t)$:

$$\begin{aligned} J_Q &= [Q(s_t, a_t) - \bar{Q}(s_t, a_t)]^2, \\ \bar{Q}(s_t, a_t) &= [r(s_t, a_t) + \gamma\mathbb{E}_{(s_{t+1},a_{t+1})\sim\rho_\pi}[Q(s_{t+1}, a_{t+1}) \\ &\quad -\alpha \log \pi(a_{t+1}|s_{t+1})]]. \end{aligned} \tag{2}$$

Then the policy network is updated towards the exponential of the new soft Q-function:

$$\begin{aligned} \pi_{new} &= \arg\min_{\pi\in\Pi} J_\pi \\ &= \arg\min_{\pi\in\Pi} \mathrm{D}_{\mathrm{KL}}\left(\pi(\cdot|s_t) \,\middle\|\, \frac{\exp(\frac{1}{\alpha}Q^{\pi_{\mathrm{old}}}(s_t, \cdot))}{Z^{\pi_{\mathrm{old}}}(s_t)}\right), \end{aligned} \tag{3}$$

where J_π denotes the objective function of the policy network, D_{KL} denotes the Kullback-Leibler divergence and $Z^{\pi_{\mathrm{old}}}(s_t)$ represents the partition function.

SAC is an off-policy algorithm which maintains a replay buffer during learning. The model learning is an iterative process and each iteration consists of a sampling stage and a learning stage. At the sampling stage, the agent interacts with the environment following the current policy and the experience is stored in the replay buffer. At the learning stage, the method updates the Q-network and the policy network. In the practical algorithm, SAC mainly adopts two techniques to make learning more stable and more efficient. Firstly, two soft Q-functions are adopted to mitigate positive bias, which is suggested in double Q-learning [60]. Specifically, two Q-networks are learned simultaneously, and the minimum value of two estimates are used to compute the target value for the Bellman update. Secondly, target networks are utilized to calculate the target value, which is suggested in [34]. Since SAC maintains two Q-networks, SAC also maintains two target networks, which are fixed during the Bellman update and updated slowly at the end of each iteration.

Evolutionary algorithms and multi-objective evolution strategy Evolutionary algorithms are a class of black-box optimization method, which imitates the evolutionary process of the species. Evolutionary algorithms have broad application scenarios including robotics [52], information retrieval [7], video games [45, 47], feature selection [1, 4, 6], text document clustering [2, 3, 5], *etc*. CMA-ES [21] is a real-valued optimization method relying on covariance matrix adaptation. It is known to work well for solution spaces of up to a few thousand parameters [16, 18, 47, 52]. MO-CMA-ES [22, 24, 55, 63] is a multi-objective extention of CMA-ES. As claimed in [47], the advantages using evolution strategies for RL are: 1) little influence from the sparsely distributed rewards and long time horizons; 2) no gradients computation; 3) avoidance of approximating the value function.

4 Method

The proposed method learns a set of policies $\Pi^\star = \{\pi_i^\star\}_{i=1}^N$ to approximate the Pareto frontier uniformly. In this section, we describe the details of the proposed two-stage algorithm, which includes the multi-policy soft actor-critic algorithm and MO-CMA-ES. Figure 1 shows an overview of the proposed method.

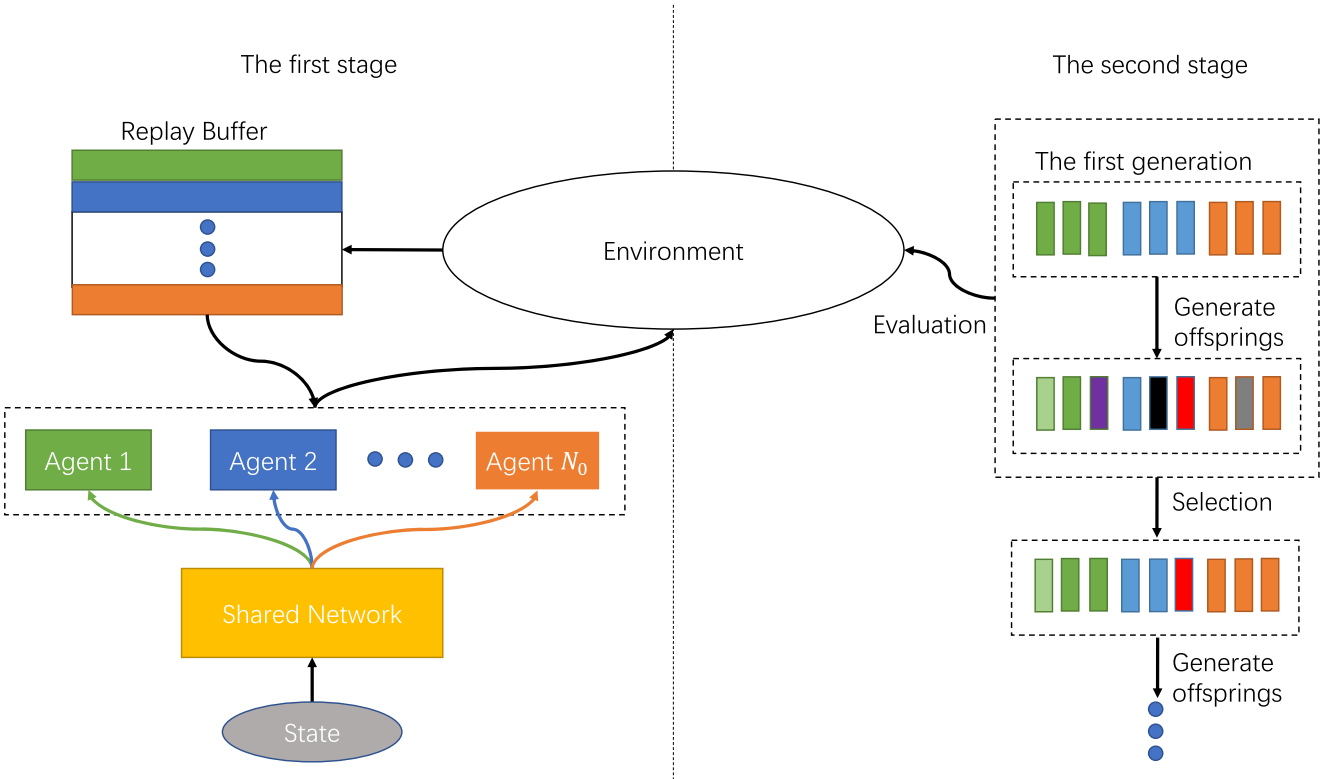

Fig. 1 Overview of the proposed framework. The first stage is a multi-policy soft actor-critic algorithm which collaboratively learns multiple policies with different preferences on objectives. The collaboration is conducted by buffer sharing. The second stage is a multi-objective evolution strategy to achieve a smooth approximation of the Pareto frontier. The first generation is initialized from the policy-independent parameters learned at the first stage

4.1 Multi-policy soft actor-critic algorithm

Algorithm The single-policy SAC is extended to a multi-policy algorithm, which learns more than one policy in the same time. Each policy is related to a specific preference of objectives. The preference is represented by specific weights. During learning, A multi-channel replay buffer is maintained. Each channel is associated with a specific policy. Collaborative learning is conducted through buffer sharing to improve sampling efficiency.

Specifically, Algorithm 1 shows the pseudocode, in which N_0 denotes the number of learned policies. The algorithm maintains N_0 groups of Q-networks and policy networks. The double Q-learning technique and target network technique are also adopted in this algorithm to stabilize learning. Therefore, in the i-th group, there are two Q-networks $\theta_{i,1}, \theta_{i,2}$, two target Q-networks $\bar{\theta}_{i,1}, \bar{\theta}_{i,2}$ and one policy network ϕ_i. The target networks are initialized randomly. And each group is associated with a specific weight vector $\boldsymbol{w}_i$. In each epoch, the algorithm includes two key procedures. Firstly, the agents sample data by interacting with the environment. The experience tuple $(s_t, a_t, \boldsymbol{r}(s_t, a_t), s_{t+1}, i)$ of agent i is stored in the i-th channel of the replay buffer. Secondly, training data is sampled from the replay buffer to update Q-networks and policy networks. Each group of Q-networks and policy networks is optimized by the single-policy SAC on the rewards scalarized by linear weights $\boldsymbol{w}_i$. The objective functions J_Q and J_π are defined in Preliminaries. Thus the target networks are updated consequently. As suggested in [15], the A2C scheme is adopted to learn the proposed algorithm. Specifically, the agents interact with the environment asynchronously, while the networks are optimized synchronously in a central agent. In this setting, the central agent needs to broadcast actions and receive rewards and the next states. See Algorithm 1 for a more formal description of the algorithm.

We discuss the linear scalarization method, shared low-level layers and replay buffer sharing in detail in the remaining part of this section.

Algorithm 1 Multi-policy soft actor-critic algorithm.

Input: Initialize N_0 Q-networks with $\{\theta_{i,1}, \theta_{i,2}\}_{i=1}^{N_0}$, N_0 target Q-networks with $\{\bar{\theta}_{i,1}, \bar{\theta}_{i,2}\}_{i=1}^{N_0}$ and N_0 policy networks with $\{\phi_i\}_{i=1}^{N_0}$.
Initialize an empty multi-channel replay buffer: $\mathcal{D} = \{\mathcal{D}_i\}$, where $\mathcal{D}_i \leftarrow \emptyset$ for $i = 1, 2, \cdots, N_0$.
Initialize the step sizes $\lambda_Q, \lambda_\pi, \tau$ for learning Q-networks, policy networks and target Q-networks, respectively.
Set the scalarization weights: $\{\boldsymbol{w}_i\}_{i=1}^{N_0}$.
for each epoch **do**
  for each environment step **do**
    for $i = 1, 2, \cdots, N_0$ **do** asynchronously
      Sample action from the policy: $a_t \sim \pi_{\phi_i}(a_t|s_t)$
      Get transition from the environment: $s_{t+1} \sim p(\cdot|s_t, a_t)$
      Store the experience in the replay buffer: $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{(s_t, a_t, \boldsymbol{r}(s_t, a_t), s_{t+1}, i)\}$
    end for
  end for
  for each gradient step **do**
    Sample experience from the replay buffer $\mathcal{X}_i \subseteq \mathcal{D}_i$ for $i = 1, 2, \cdots, N_0$.
    for $i = 1, 2, \cdots, N_0$ **do** synchronously
      Construct training data: $\widetilde{\mathcal{X}}_i = \bigcup_{j=1}^{N_0} \mathcal{X}_j$ or $\widetilde{\mathcal{X}}_i = \mathcal{X}_i$ or $\widetilde{\mathcal{X}}_i \leftarrow \mathcal{X}_i \bigcup_{j \in \Omega_i} \mathcal{X}_j$. Ω_i denotes the set of adjacent policies.
      Weighted scalarization: $\widetilde{\mathcal{X}}_i \leftarrow \{(s_t, a_t, \boldsymbol{w}_i^T \boldsymbol{r}(s_t, a_t), s_{t+1})\}$, where the channel index is neglected.
      Update the Q-function parameters: $\theta_{i,j} \leftarrow \theta_{i,j} - \lambda_Q \hat{\nabla}_{\theta_{i,j}} J_Q(\theta_{i,j})$ for $j \in \{1, 2\}$
      Update policy weights: $\phi_i \leftarrow \phi_i - \lambda_\pi \hat{\nabla}_{\phi_i} J_\pi(\phi_i)$
      Update target network weights: $\bar{\theta}_{i,j} \leftarrow \tau\theta_{i,j} + (1-\tau)\bar{\theta}_{i,j}$ for $j \in \{1, 2\}$
    end for
  end for
end for
Output: Parameters of learned policy networks: $\{\phi_i\}_{i=1}^{N_0}$

Linear scalarization method Considering a specific policy, the linear scalarization method transfers the reward vector $\boldsymbol{r}_t$ to a scalar: $r_t = \boldsymbol{w}^T \boldsymbol{r}_t$, where $\boldsymbol{w}$ denotes the weights. Since the number of objectives is M, $\boldsymbol{w}$ is a M-dim vector. Each element of the weight vector should be equal to or larger than zero. At least one element is larger than zero. How to assign the weights reveals the preference/importance of the related objectives. Furthermore, the strategy of assigning weights is different from one problem to another. Thus, considering two objectives, the method empirically fixes the weight of the first objective as 1 and varifies the weight of the second objective in a predefined range uniformly. Even though there are limitations in the linear scalarization method when the Pareto frontier is concave [59]. Specifically, the methods with linear scalarization method can not achieve the Pareto optimal solutions in the concave region. This problem is handled by the evolution strategy at the second stage.

Shared low-level layers To reduce computational complexity, the policy networks share low-level network parameters. From the perspective of representation learning, joint optimization learns a more compact and robust representation, thus the performance is improved. The learned representation also guarantees the performance at the second stage, in which the low-level parameters are fixed. In the non-Markovian environment, recurrent neural networks (RNN) are adopted as policy networks. The recurrent connections are utilized to gather the information of previous time steps. At the second stage, the recurrent part of RNN is fixed. Note that we adopt asynchronous exploration and synchronous learning, so there is no communication bottleneck for sharing low-level layers. Since the shared low-level structures are not compulsive for MPSAC, specified symbols of shared parameters and policy-independent parameters are not demonstrated in Algorithm 1.

Replay buffer sharing Collaborative learning is implemented by sharing the replay buffer. We believe that each policy learns a specific exploration mode since each policy aims to optimize a specific weighted goal. Hence closer scalarization weights may lead to closer modes. The collaboration is two folds: firstly, policies with close modes follow similar/overlap exploration pattern to cooperate on data sampling; secondly, if a policy trained by the sampled data from another policy working in different mode, the value function is learned more critically, thus its exploration efficiency is improved. The multi-channel replay buffer consists of all the historical experience tuples. Each tuple is represented by $\{(s_t, a_t, \boldsymbol{r}(s_t, a_t), s_{t+1}, i)\}$, which includes the current state s_t, action a_t, reward vector $\boldsymbol{r}(s_t, a_t)$, the next state s_{t+1} and the index of associated policy i. The index can also be explained as the channel index. By storing the vector rewards rather than the scalarized rewards, each policy can utilize data sampled from other policies. According to which part of the replay buffer a policy can access, we investigate three buffer sharing strategies. In the global sharing strategy, a policy accesses the whole buffer. In the no sharing strategy, each policy only accesses its buffer. And in the neighbor sharing strategy, each policy can utilize its buffer and the buffer of the adjacent policies. The adjacent set is defined as the set of policies with closest scalarization weights. Through ablation experiments, the global sharing strategy is regarded as the default strategy since this strategy achieves the best result.

SAC is not the only alternative RL algorithm for our proposed framework. Other single-objective off-policy RL algorithms such as Deep Q-Learning [33] and DDPG [27] can be extended in the same way.

4.2 Multi-Objective Covariance Matrix Adaptation Evolution Strategy (MO-CMA-ES)

The implementation of MO-CMA-ES is adapted from the Shark machine learning library [23], which is mainly based on [24, 63]. In this work, we utilize MO-CMA-ES to refine the rough approximation of the Pareto frontier from the first stage.

Algorithm 2 shows the general procedures (four main steps) of MO-CMA-ES. The method maintains N individuals. Note that it is not compulsive to constrain that N must be equal to N_0. Each individual a_i consists of a chromosome (search point) $\bar{\phi}_i$, a global step size σ_i and a covariance matrix C_i. The chromosome $\bar{\phi}_i$ is a n-dimensional vector initialized by vectorizing the policy-independent parameters of a randomly selected learned policy at the first stage. σ_i is initialized as 0.1 and C_i is initialized as a n-dimensional identity matrix. The initialized individual a_i is evaluated by simulation and labeled by a fitness vector $\boldsymbol{y}_i$. Each element of $\boldsymbol{y}_i$ represents the accumulated reward of the related objective. In every generation, each individual generates one offspring by sampling from the Gaussian distribution: $\bar{\phi}_i \sim \mathcal{N}(\bar{\phi}_i, \sigma_i C_i)$, in which the mean and the covariance matrix are $\bar{\phi}_i$ and $\sigma_i C_i$. New offsprings are evaluated by simulation and labeled with fitness vectors. Then all individuals are ranked and the N elitists are selected as the parents of the next generation. At last, the covariance matrix of each selected individual is updated consequently. Please refer to [24, 63] for more details. The algorithm repeats this procedure until the stopping criterion is met.

Algorithm 2 Multi-objective convariance matrix adaptation evolution strategy.

1: **Input:** Initialize the first generation with N individuals. Each individual a_i consists of a search point $\bar{\phi}_i$, a global step size σ_i and a covariance matrix C_i.
2: Evaluate the first generation, each individual a_i is labeled with a fitness vector $\boldsymbol{y}_i$.
3: **for** each generation **do**
4: Generate one offspring for each individual: $\bar{\phi}_i^{(g+1)} \sim \mathcal{N}(\bar{\phi}_i^{(g)}, \sigma_i^{(g)} C_i^{(g)})$.
5: Evaluate N offsprings by simulation, each individual $a_i^{(g)}$ is labeled with a fitness vector $\boldsymbol{y}_i^{(g)}$.
6: Sort $2N$ individuals (parents and offsprings), then select first N individuals.
7: Update step size and covariance matrix for each individual.
8: **end for**
9: **Output:** Optimized parameters: $\{\phi_i\}_{i=1}^N$

Selection step The details of the selection step are demonstrated. Two ranking criteria are considered: the non-dominance level and the crowding distance, as suggested by [14]. The non-dominance level is determined by recursively exclude non-dominant individuals from the solution set. Formally, let A be a population and a, a' be two individuals. The non-dominant set of A is defined as:

$$\text{ndom}(A) = \{a \in A | \nexists a' \in A : a \prec a'\}, \tag{4}$$

where $\prec$ denotes the Pareto-dominance relation. Let $A_0 = A$, $A_l = A_{l-1} \backslash \text{ndom}(A_{l-1})$, for $l \geq 1$, where "\" denotes a removing operation. Thus $\text{rank}(a, A) = i + 1$, iff $a \in \text{ndom}(A_i)$. When the Pareto frontier is concave, the optimal solution in the concave region still has a high non-dominance level, then the limitation of linear scalarization is handled.

Through the first criterion, the individuals are ranked to many non-dominant sets. The crowding distance is used as the second ranking criterion to rank the points in a non-dominant set. For M objectives, the crowding distance of point a in the non-dominant set A' is defined by:

$$c(a, A') = \sum_{m=1}^{M} c_m(a, A') / (f_m^{\max} - f_m^{\min}), \tag{5}$$

where $f_m^{\max}$ and $f_m^{\min}$ are the maximum and the minimum values of the m-th objective respectively. $c_m(a, A')$ is positive infinite when a is a boundary point. Otherwise, $c_m(a, A')$ is the distance between two closest points of a w.r.t. the m-th objective. Therefore, the boundary points and the less crowded points are preferred, so that the approximated Pareto frontier is more spread and uniform.

5 Experiments

In this section, the comparative methods and the evaluation protocol are introduced. Then we show the experimental results on three benchmark problems, which shows the superiority of the proposed method. And the ablation studies are described for deeper analysis.

5.1 Comparative methods

The proposed methods are compared with four methods, the hypervolume-based (HB) [61] algorithm, the radial algorithm (RA) [43], the Pareto following algorithm (PFA) [43] and the Deep Neuroevolution (DN) [52]. For a fair comparison, the same network structures are adopted for the proposed methods and the comparative methods in all problems. HB is an extension of Deep Q-learning [33]. This method maintains value function approximation for each objective and takes actions according to the largest hypervolume indicator. Hence, no scalarization method is applied in HB. As for RA and PFA, the linear scalarization method is adopted and SAC is also utilized for a fair comparison. RA optimizes more than one policy simultaneously. Each policy is assigned with a specific weight vector on objectives. In each iteration, through applying SAC, every individual policy is optimized independently by the gradients related to its assigned preference. From another idea, PFA is optimized directly on the Pareto frontier. Specifically, at first, an initial model is learned by SAC with initialized scalarized weights. Then the weights are tuned uniformly and gradually and the model is fine-tuned for each tuned preference iteratively. Naive stochastic gradient descent methods are utilized in RA and PFA, considering the computational complexity of deep neural networks. DN is an efficient genetic algorithm for optimizing deep neural networks to solve single-objective problems. In this work, DN is naturally extended to a multi-objective optimization algorithm by using the same ranking criteria and selection strategies as our method.

5.2 Evaluation protocol

The comparative experiments are conducted on three benchmark problems: Deep Sea Treasure, Adaptive Streaming and Super Mario Bros. The hypervolume indicator [66] is utilized as our evaluation protocol, which is suggested by [58] because of its sensitivity on the improvements

in any frontal characteristics (accuracy, extent, diversity). The experimental results are represented by the mean and standard deviation to show the overall performance and stability. The results are calculated from repeated independent runs. The number of repeats is 30 and 10 and 10 for three benchmark problems, respectively.

Furthermore, to consider the smoothness of the approximated Pareto frontier, the solutions are ranked so that one objective is monotonically increasing and the other objective is monotonically decreasing. Then the Euclidean distance between all pairs of neighbor solutions is calculated. By doing this, a distance array is achieved. The standard deviation of this distance array is utilized to measure the smoothness of the approximated Pareto frontier.

Different random seeds are used for training and testing. We also demonstrate the wall-clock time and the number of parameters for showing time and space complexity. The codes run on PyTorch 1.0 [44], Ubuntu 16.04 system with 44 Intel(R) Xeon(R) CPUs.

5.3 Problem 1: deep sea treasure

Problem description This is an episodic task. A submarine searches for treasures in a 10×11 grid. The treasures with different values are in different locations. At each episode, the submarine starts from the top left corner, searches by four actions- left, right, up and down. The end condition is reaching a treasure or moving 1,000 steps. The reward is a 2-dim vector. The first element is negative time and the second element is the reached treasure value. Therefore, the goal is to maximize the treasure value and minimize the time penalty. This task is created to highlight the limitations of linear weighted scalarization [59], since the Pareto frontier of this task is globally concave.

This task is tested in two different ways. Firstly, the learned policies are tested deterministically. The agents always take actions with maximum probability. In this setting, there are ten Pareto optimal policies. Each optimal policy leads to one treasure location. Secondly, the policies are tested stochastically. During testing, the actions are sampled according to the learned policy. In this setting, the results of each run are averaged from 20 simulations.

Consequently, the proposed algorithm adopts different fitness evaluation mechanisms at the evolution strategy stage to handle these two testing settings. For training deterministic policies, the evaluation is conducted in only one episode. During learning stochastic policies, the fitness score vector is averaged from the results of 20 episodes.

Implementation details The input state is a 61-dim binary vector. Each element of the vector is related to a specific available position. All elements are zero except the element representing the current position of the submarine. Both the policy network and Q-network consist of two linear layers with the ReLU activation function. The hidden layer has 128 hidden units. A softmax layer is on the top of the policy network. At the first stage, the model simultaneously learns five policies by 25 agents. The linear weights of learned policies are $(1.0, 0.1)$, $(1.0, 0.12)$, $(1.0, 0.14)$, $(1.0, 0.16)$ and $(1.0, 0.18)$, respectively. The model is learned for 400 epochs, each of which includes 100 sampling iterations and 100 training iterations. Each mini batch includes 100 samples from each agent. The learning rate is 0.001. At the second stage, the number of individuals in each generation is 10. To calculate the hypervolume indicator, the reference point is $(-30, 0)$.

Experimental results of deterministic testing HB only maximizes hypervolume indicator for a single policy, so the solutions learned by HB are poorly diversified. In this problem, HB often converges to the policy which approaches the highest treasure value. Therefore, HB achieves the lowest hypervolume indicator within all comparative methods. As shown in Table 1, when testing deterministic policies, policy gradient methods with linear scalarization (RA and PFA) are not able to approach all Pareto optimal solutions. In this problem, both RA and PFA achieve near two out of ten Pareto optimal solutions, which causes low hypervolume indicator. Furthermore, as a genetic algorithm, DN achieves close performance as our method but with less stability. Give the credit to MO-CMA-ES at the second stage, our methods can approach all optimal policies and higher hypervolume indicator stably.

Experimental results of stochastic testing HB is not tested in this scenario because it is not trivial to transfer HB into a stochastic policy. As shown in Table 1, compared with PFA, RA achieves higher average hypervolume indicator and lower standard deviation, which shows that RA is more effective and more stable than PFA in this task. Nevertheless, PFA needs less training time since it operates on the Pareto frontier with model reusing. DN performs better than RA and PFA with higher stability in this task. And our method outperforms these comparative methods with a considerable margin. A two-sample T-test (with a 95% confidence) is conducted for a statistical significance analysis, which shows that the aforementioned comparative results are all statistically significant. Furthermore, the smoothness of RA, PFA, DN and our method are 21.72, 19.54, 14.35 and 11.65 respectively, which indicates that our method approaches better-distributed frontier.

Work loads As shown in Table 1, considering the number of iterations, the training wall-clock time and the number of parameters, our algorithm has lower time and space complexity. Especially, to store optimal solutions of

Table 1 Experimental results of Deep Sea Treasure

	HI (Deterministic)	HI (Stochastic)	Iterations	Wall-Clock Time	Parameters
HB	$1364.00 \pm 0.00(1.00 \pm 0.00)$	–	4,000	$6.68 \times 10^4 s$	8.97×10^4
RA	$1406.33 \pm 54.41(2.17 \pm 0.37)$	1857.75 ± 156.19	4,000	$7 \times 10^4 s$	8.45×10^4
PFA	$1346.47 \pm 125.58(2.03 \pm 0.18)$	1713.71 ± 274.73	1,400	$2.45 \times 10^4 s$	8.45×10^4
DN	$1637.40 \pm 444.35(9.30 \pm 2.25)$	–	30,000	$3 \times 10^3 s$	8.45×10^4
DN	–	1997.56 ± 114.41	15,000	$2.13 \times 10^4 s$	8.45×10^4
Ours	$\mathbf{1756.13 \pm 28.20(9.53 \pm 0.62)}$	–	$400 + 100$	$3 \times 10^3 s$	1.31×10^4
Ours	–	$\mathbf{2134.81 \pm 88.80}$	$400 + 1500$	$1.93 \times 10^4 s$	1.31×10^4

HI denotes the hypervolume indicator. The number in the bracket of deterministic results indicates the number of approached Pareto optimal policies. The number of iterations of our algorithm includes two parts since the method has two different stages, thus "+" is used to separate these two parts

deterministic policies, our algorithm uses much fewer parameters to represent more optimal solutions than baseline methods.

5.4 Problem 2: adaptive streaming

Problem description This problem is derived from an anonymous technical report,[1] which proposed a rate-distortion optimization framework for Adaptive Streaming [31, 51, 53, 65] In this problem, the server segments each video into a set of 4-second consecutive video chunks. Each chunk is encoded into different representations. Each representation is characterized by a perceptual quality and a bitrate. The perceptual quality is estimated from the encoded video segment and assumed viewing devices. During downloading, the controller determines which representation to download for the next chunk. Then the environment downloads the chunk, updates the buffer occupancy, tracks the throughput and returns the new states to the agent. Except for the perceptual quality, the stalling effect and adaptation effect are also considered in a unified Quality-of-Experience (QoE) score. In the rate-distortion optimization framework, the first objective is to maximize the overall QoE and the second one is to minimize the average bitrate consumption. Hence, the reward function at the first stage is: $\text{Reward} = \text{QoE} - \lambda\text{Bitrate}$, where λ is the scalarization weight specified for each policy. Since the environment in this problem is non-Markovian, RNNs are used to handle temporal information. Consequently, during learning MPSAC models, the storing and retrieving of the replay buffer treat a complete sequence of experience as an operation unit.

Implementation details The dataset includes 5 source videos and 300 network traces. The videos are encoded by the H.264 video encoder as 13 representations. We only consider Phone as the viewing device. The same videos are used during training, validation and testing. Network traces are equally split into a training set, a validation set and a testing set. Both the policy network and Q-network are constructed by two GRU layers and one linear layer. All hidden layers consist of 128 neurons. The activation function is ReLU. A softmax layer is at the top of the policy network. At the first stage, the model learns 1,000 epochs, each of which includes 100 sampling iterations and 20 training iterations. The model simultaneously learns five policies by 25 agents. The scalarization weights are $(1.0, 0.0)$, $(1.0, 0.0025)$, $(1.0, 0.0050)$, $(1.0, 0.0075)$ and $(1.0, 0.01)$. Each batch includes 100 samples from each agent. The learning rate is 0.0003. At the second stage, the number of individuals is 50. Since the lowest QoE score is 0 and the highest bitrate is 16800, the reference point $(0, -16800)$ and normalizing factor $(10, 1680)$ are utilized in the objective space for calculating hypervolume indicator.

Experimental results Table 2 shows the experimental results of Adaptive Streaming. HB achieves low hypervolume indicator because of the solutions are not spread enough. RA outperforms PFA considering both the mean and the standard deviation of the hypervolume indicator, which is similar to the results of Deep Sea Treasure. The instability of PFA may be caused by its learning procedure which tunes models incrementally. During learning, the uncertainty and inaccuracy could be propagated and enlarged along with learning. DN performs better than RA and PFA in Deep Sea Treasure but worse than RA and PFA in this task, which reveals that it is still not efficient enough to optimize RNNs by applying a genetic algorithm straightly. Leveraging the advantages of gradient-based methods and evolution strategies, the proposed algorithm is capable to train deep RNNs well and achieves the highest hypervolume indicator and stability. Due to the low-level

[1] Anonymous et al., Adaptive Streaming: From Bitrate Maximization to Rate-Distortion Optimization

Table 2 Experimental results of Adaptive Streaming

	HI	Iterations	Wall-Clock Time	Parameters
HB	61.252 ± 0.134	50,000	$3.35 \times 10^5 s$	9.06×10^6
RA	73.140 ± 0.058	50,000	$1.25 \times 10^6 s$	8.97×10^6
PFA	73.115 ± 0.073	16,000	$4 \times 10^5 s$	8.97×10^6
DN	72.642 ± 1.146	500	$3.2 \times 10^5 s$	8.97×10^6
Ours	$\mathbf{73.460 \pm 0.048}$	$1,000 + 200$	$2.66 \times 10^5 s$	2.62×10^5

HI denotes hypervolume indicator. The number of iterations of our algorithm includes two parts since the method has two different stages, thus "+" is used to separate these two parts

parameters sharing, our algorithm has lower time and space complexity comparing with RA and PFA as well. Specifically, the RA method needs the longest training time. PFA method is more efficient on time complexity while it is still redundant on model representation. Our algorithm is efficient in both ways. We also conduct a statistical significance analysis by a two-sample T-test (with a 95% confidence). The data for hypothesis testing is collected from the hypervolume indicators by independent repetition. Table 3 shows that our method is statistically better than all the compared models.

Furthermore, Fig. 2 shows the estimated Pareto frontiers of RA, PFA and our method. In the solution space, there are some missing regions for RA and PFA which may cause inconvenience and poor performance during practical application, while our method achieves smoother frontier than RA and PFA. As for the quantitive evidence, the smooth measure of RA, PFA and our method is 0.042, 0.035 and 0.012, respectively. Considering the region of the highest QoE, our method uses fewer bitrates and approaches higher QoE than the other two methods.

5.5 Problem 3: super mario bros

Problem description This task is implemented by [25] which is an OpenAI Gym environment. It is designed on the Nintendo Entertainment System (NES) using the nes-py emulator. In this task, Mario explores a two-dimensional map by seven alternative actions, which include 'doing nothing', 'right', 'right+A', 'right+B', 'right+A+B', 'A' and 'left'. These actions are simple combinations of basic actions corresponded to the buttons on the NES controller. The states are the current game-play frames. Mario has three lives in each epoch. The goal is to make Mario run further away and collect more coins. To achieve this, we set a 2-dim reward vector. Implemented in [25], the first reward is the sum of three elements: the difference of agent's horizontal coordinates between states, the difference in the game clock between frames and a death penalty that penalizes the agent for dying in a state. This setting punishes standing still and death, and encourages going right. The second reward is the collected coins. For evaluation, only the horizontal position and collected coins are concerned. This problem setting is a typical example of reward shaping, thus one objective can be regarded as the main objective and the other objective can be regarded as the auxiliary one.

Table 3 Statistical significance matrix of Adaptive Streaming

Methods	HB	DN	PFA	RA	Ours
HB	–	0	0	0	0
DN	1	–			0
PFA	1	1	–		0
RA	1	1	1	–	0
Ours	1	1	1	1	–

The comparisons are based on the hypothesis testing. "1" means that the row algorithm outperforms the column algorithm statistically; "0" means that the row algorithm performs worse than the column algorithm statistically; "–" means that two algorithms are indistinguishable statistically

Implementation details The comparative algorithms are tested on the first stage of the first world of Super Mario Bros. In each time step, the input is the current game-play frame represented by a color image with RGB channels. The initial size of the input image is $240 \times 256 \times 3$. To reduce the time and space complexity, the image is subsampled to size $60 \times 64 \times 3$ by bilinear interpolation and the value of each pixel is normalized into range $[0, 1]$. Both the policy network and Q-network are constructed by stacking four convolutional layers, one GRU layer and one fully connected layer. All convolutional layers consist of 32 convolutional kernels with stride 2 and padding 1. The GRU layer has 512 units. The activation function is ReLU. At the first stage, the model learns 5,000 epochs, each of which includes 100 sampling iterations and 20 training iterations. The model simultaneously learns five policies by 15 agents. The scalarization weights are $(1.0, 0.2)$, $(1.0, 0.4)$, $(1.0, 0.6)$, $(1.0, 0.8)$ and $(1.0, 1.0)$. Each training batch includes 100 samples from each agent. The learning rate is 0.0003. At the second stage, the number of individuals is 15. This setting is determined empirically by considering

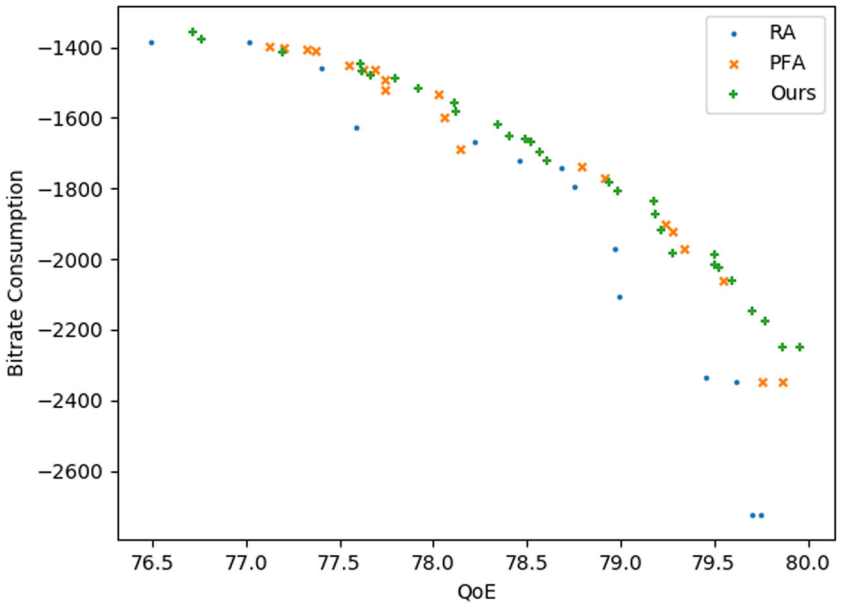

Fig. 2 Approximation of the Pareto frontier in Adaptive Streaming

the range of horizontal position and the number of collected coins.

Experimental results Table 4 shows the experimental results of Super Mario Bros. Since this task is mainly designed for testing reward shaping, the highest horizontal position and the number of collected coins are shown in Table 4, which is different from the previous two problems. By this metric, HB is comparable to RA and DN, which reveals that HB can achieve the solutions on or close to the Pareto frontier even though HB can not achieve diversified solutions. PFA achieves the highest horizontal position but lowest collected coins. The reason may be that PFA is initialized to maximize the horizontal position then gradually finetuned on the Pareto frontier, but PFA is stuck on the previous goals so that it is not efficient to collect coins. Our method achieves a slightly smaller horizontal position than PFA (660.84 versus 671.95). Concerning the collected coins, our method outperforms the other methods with a considerable margin, which is benefit from the high exploration efficiency of multi-policy collaboration. And our algorithm has lower time and space complexity.

5.6 Ablation study

Several ablation studies are conducted on Adaptive Streaming to demonstrate the contribution of the proposed components.

Which data sharing strategy is better? To reveal the sampling efficiency of MPSAC, we investigate three different data sharing strategies: global sharing, no sharing and neighbor sharing. These three methods are named as MPSAC.A, MPSAC.O and MPSAC.N respectively, which means that each policy attains all samples, own samples and neighbor samples. For comparison, the result of SAC is achieved from five independent repetitions. Each run is related to a specific preference. These results of different algorithms are achieved from exploring the same number of iterations for each agent. Table 5 shows the results of different data sharing strategies at the first learning stage. All three strategies outperform the baseline method with less computational redundancy. MPSAC.O outperforms SAC, which reveals the advantage of sharing low-level parameters. It means that sharing low-level

Table 4 Experimental results of Super Mario Bros

	Horizontal Position	Collected Coins	Iterations	Wall-Clock Time	Parameters
HB	641.25	1571.65	75,000	$9.38 \times 10^5 s$	2.42×10^7
RA	642.85	1571.15	75,000	$1.88 \times 10^6 s$	2.41×10^7
PFA	**671.95**	1178.70	20,000	$5 \times 10^5 s$	2.41×10^7
DN	639.20	1573.20	100	$2.6 \times 10^5 s$	2.41×10^7
Ours	660.84	**2518.74**	$5000 + 100$	$1.9 \times 10^5 s$	1.66×10^6

The number of iterations of our algorithm includes two parts since the method has two different stages, thus "+" is used to separate these two parts

Table 5 Ablation study on three exploration strategies

	HI	Iterations	Wall-clock time
SAC	72.50 ± 0.18	5,000	$1.25 \times 10^5 s$
MPSAC.A	$\mathbf{72.97 \pm 0.11}$	1,000	2.4×10^4
MPSAC.O	72.93 ± 0.16	1,000	2.35×10^4
MPSAC.N	72.85 ± 0.10	1,000	2.6×10^4

Single-policy SAC is executed repeatedly for five times with different preferences on objectives. MPSAC maintains five policies at the same time

features not only reduces the time and space complexity but also enhances the representation learning for more robust policies. MPSAC.A and MPSAC.N exceed MPSAC.O, which shows the data sharing improves sampling efficiency. MPSAC.A outperforms MPSAC.N, which reveals that all sharing is a better data sharing strategy. Furthermore, Fig. 3 shows the training curve of SAC and MPSAC.A. Our method approaches higher performance and higher stability. And our method needs fewer training samples to achieve the same hypervolume indicator, which reveals the exploration efficiency introduced by collaborative learning.

Is representation learning important? To answer this question, the model learning at the first stage is modulated by investigating two hyper-parameters, which are the number of learning epochs and the number of learning policies. These two hyper-parameters are correlated with the robustness of the representation learning, hence we expect that the performance will increase if these parameters increase.

Firstly, we fix the mentioned setting, namely, learning 5 policies by 25 agents. Then the models initialized from different learning epochs (300, 500, 700, 900) are adopted at the second stage. The comparative results in Table 6 show that the model approaches better results when the number of training epochs increases. Secondly, we fix the number of learning epochs but change the number of learned policies to 1, 5, 10, 15, 20, and 25, respectively. The comparative results in Table 7 show that the final performance improves in general by adding the number of learned policies at the first stage.

Furthermore, a model without pretraining low-level networks is compared in Table 7. Represented by setting the number of pretraining agents as 0, this model is implemented by straightly training the policy-independent parameters with randomized low-level parameters. This model achieves a much lower hypervolume indicator than other methods with pretraining. We also increase the training agents from 25 to 50. MPSAC(50,50) denotes the MPSAC algorithm learning 50 policies with 50 agents at the same time. By adopting the MO-CMA-ES algorithm at the second learning stage, the performance improves from 73.286 to 73.663, which shows a considerable performance gain of the second-stage learning.

The experimental results are satisfied with our expectations. These four ablation studies show the importance of representation learning at the first stage.

Ablation study on sampling efficiency To investigate and explain the collaborative learning of our MPSAC algorithm, an ablation study is elaborately designed through assigning the available training samples for each policy. The method

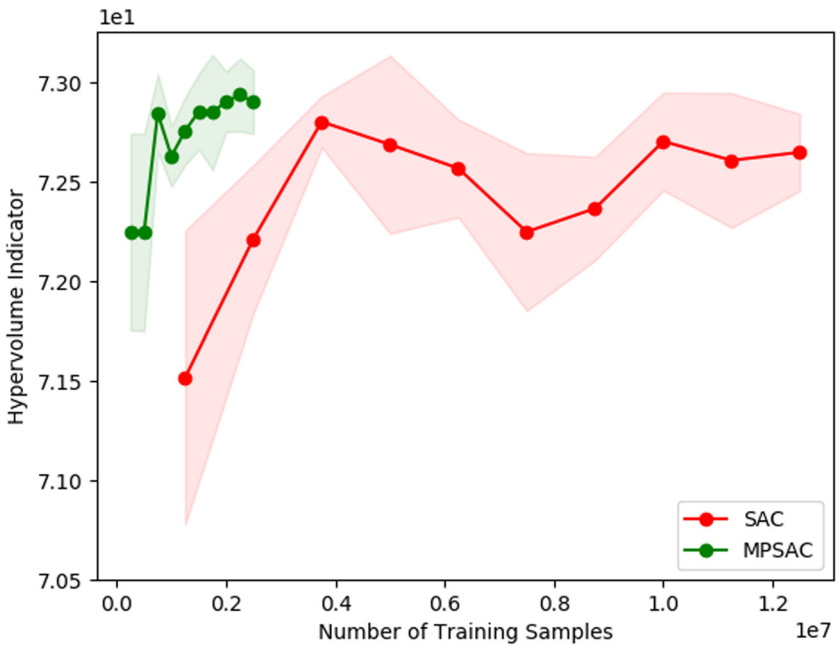

Fig. 3 Training curves showing sampling efficiency of MPSAC algorithm compared with SAC

Table 6 Ablation study on the number of training epoches at the first stage to show the importance of representation learning. MPSAC(k) represents a first-stage model which learns k epoches

Number of Epoches	HI
MPSAC(300)+ES	73.189
MPSAC(500)+ES	73.350
MPSAC(700)+ES	73.290
MPSAC(900)+ES	**73.405**

in this comparative experiment simultaneously learns 12 policies, which are divided into 4 groups. For simplicity, these groups are named as group A, B, C and D. The main purpose of this ablation study is to train policies with scalarization weight $(1, 0.001)$ in different problem settings, namely, different training samples. In group A, the scalarization weights of three policies are all $(1, 0.001)$. The three policies in group B are learned with weights $(1, 0.001)$, $(1, 0.0005)$ and $(1, 0.002)$, while the weights for group C are $(1, 0.001)$, $(1, 0.0002)$ and $(1, 0.005)$. Hence, the policies in these three groups pursue the same goal, similar goals and diverse goals, respectively. Table 8 shows the average QoE, bitrate and reward of the first policy with weights $(1, 0.001)$ in these groups. The results show that both policies for similar goals and diverse goals achieve better results than the policies for the same goal, which demonstrates the benefits of collaborative learning. In group D, policy A′/B′/C′ is learned from its own samples and the samples of the first policy of group A/B/C. The results show that samples generated by the first policy of group B and group C are more efficient for learning. In summary, these ablation tests prove that the collaboration is two folds: firstly, policies optimized for similar goals cooperate with higher sampling efficiency; secondly, the exploration efficiency of an individual policy is improved through training on the samples generated by another policy working in diverse mode.

Table 7 Ablation study on the number of learning policies at the first stage

Models	HI
MPSAC(0,25)+ES(50)	63.998
MPSAC(1,25)+ES(50)	73.269
MPSAC(5,25)+ES(50)	73.460
MPSAC(10,25)+ES(50)	73.453
MPSAC(15,25)+ES(50)	73.463
MPSAC(20,25)+ES(50)	73.435
MPSAC(25,25)+ES(50)	**73.489**
MPSAC(50,50)	73.286
MPSAC(50,50)+ES(50)	**73.663**

MPSAC(m, n) represents a first-stage model which learns m policies using n agents by MPSAC. ES(m) denotes a second-stage model learning m policies by MO-CMA-ES

Table 8 Ablation study on sampling efficiency with elaborate experimental design

Policy	QoE	Rate	Reward
Group A	79.501	2151.398	77.350
Group B	79.441	2060.386	77.381
Group C	79.498	2093.226	**77.404**
Policy A′	79.514	2222.751	77.291
Policy B′	79.509	2161.001	**77.348**
Policy C′	79.461	2139.077	77.322

6 Discussion

Why not just apply MO-CMA-ES to learn deep policies? An alternative way to learn the deep policies in this work is to apply MO-CMA-ES to the whole neural networks and neglect the first pretraining stage. Specifically, the chromosome of each individual is constructed through vectorizing and stacking the parameters of the whole neural network. However, the critical issue is that the algorithm needs too much runtime memory, since the space complexity of MO-CMA-ES is $\Theta(n^2)$. For example, in Adaptive Streaming problem, the dimension of the chromosome is increased from 1,677 to 261,642 by stacking all parameters together. By doing this, the covariance matrix of each individual needs more than 200 gigabytes of memory, which is impractical.

The similarity to the bootstrapped DQN The proposed MPSAC algorithm is similar to the bootstrapped DQN [38], which is a deep version of the bootstrapped Thompson sampling [40, 46]. Inspired from [41], to improve the sampling efficiency, the bootstrapped DQN utilizes randomized value functions to approximate the posterior sampling, rather than modeling an intractable exact posterior estimate. This method is adopted in [38, 39, 41] as well. Both bootstrapped DQN and our method maintain more than one deep networks for Q-value estimation. These networks share the same network structure. The difference is that the DQNs in [38] are optimized for the same objective. Randomized initialization and independent training samples are utilized to impel the randomness of model learning to improve exploration efficiency. In our method, the policy networks pursue differently scalarized objectives. Therefore, our method naturally sustains the model randomness to improve sampling efficiency, since different policies pursue different goals.

Limitation The limitation of this method is discussed below. The computational complexity of the evolution strategy is reduced through the parallelism and the avoidance of gradients computation. This effect is significant especially when the policy networks and Q-networks are RNNs. While in some problem settings, evaluation by simulation has high computational complexity. For instance, if more videos and traces are tested in Adaptive Streaming, the evaluation step in the evolution strategy is more time-consuming. Then the proposed algorithm may be relatively less efficient than a mini-batch gradient-based algorithm. Therefore, it is paramount to explore the "mini-batch" evaluation strategy which takes both efficiency and performance into consideration. This issue should be addressed in future research. Furthermore, the current method needs to manually decide the shared low-level layers, which may not be the optimal solution. Hence, determine the shared parameters automatically should be a potential research direction.

Other possible methodologies There are some other possible methodologies for further investigation. For example, it is paramount to design an evolution strategy that can efficiently optimize the recurrent strategy. The current performance may be hindered by the sensitivity of recurrent connections. Hence, designing a more cautious optimization strategy for the recurrent neural network may alleviate this issue. For example, the parameters can be divided into recurrent parameters and non-recurrent parameters, and each step of optimization only changes one group of parameters.

7 Conclusion

In this paper, we propose a two-stage algorithm to address the multi-objective reinforcement learning problem. The first stage is a multi-policy soft actor-critic algorithm which collaboratively learns multiple policies with different preferences on objectives. The second stage is a multi-objective evolution strategy algorithm to achieve a smooth approximation of the Pareto frontier. Our method is efficient in data exploration and model representation.

The proposed model may be improved in many ways. First, more collaboration strategies can be investigated to improve sampling efficiency. Second, more efficient evaluation methods can be explored to accelerate model learning at the second stage. Third, how to integrate gradient-based methods and evolutionary algorithms more efficiently is worth further studies.

Acknowledgements The authors would like to express our thanks for the support from the following research grants: 2018AAA0102004, NSFC-61625201, NSFC-61527804.

References

1. Abualigah LM, Khader AT (2017) Unsupervised text feature selection technique based on hybrid particle swarm optimization algorithm with genetic operators for the text clustering. J Supercomput 73(11):4773–4795
2. Abualigah LM, Khader AT, Hanandeh ES (2018) A combination of objective functions and hybrid krill herd algorithm for text document clustering analysis. Eng Appl Artif Intell 73:111–125
3. Abualigah LM, Khader AT, Hanandeh ES (2018) Hybrid clustering analysis using improved krill herd algorithm. Appl Intell 48(11):4047–4071
4. Abualigah LM, Khader AT, Hanandeh ES (2018) A new feature selection method to improve the document clustering using particle swarm optimization algorithm. J Comput Sci 25:456–466
5. Abualigah LM, Khader AT, Hanandeh ES, Gandomi AH (2017) A novel hybridization strategy for krill herd algorithm applied to clustering techniques. Appl Soft Comput 60:423–435
6. Abualigah LMQ (2019) Feature selection and enhanced krill herd algorithm for text document clustering. Springer, Berlin
7. Abualigah LMQ, Hanandeh ES (2015) Applying genetic algorithms to information retrieval using vector space model. International Journal of Computer Science, Engineering and Applications 5(1):19
8. Barrett L, Narayanan S (2008) Learning all optimal policies with multiple criteria. In: International conference on machine learning. ACM, pp 41–47
9. Bengio Y, Courville A, Vincent P (2013) Representation learning: a review and new perspectives. IEEE Trans Pattern Anal Mach Intell 35(8):1798–1828
10. de Bruin T, Kober J, Tuyls K, Babuška R (2018) Integrating state representation learning into deep reinforcement learning. IEEE Robotics and Automation Letters 3(3):1394–1401
11. Brys T, Harutyunyan A, Vrancx P, Taylor ME, Kudenko D, Nowé A (2014) Multi-objectivization of reinforcement learning problems by reward shaping. In: International joint conference on neural networks. IEEE, pp 2315–2322
12. Castelletti A, Pianosi F, Restelli M (2012) Tree-based fitted Q-iteration for multi-objective Markov decision problems. In: International joint conference on neural networks. IEEE, pp 1–8
13. Chen D, Wang Y, Gao W (2020) A two-stage multi-objective deep reinforcement learning framework. In: European conference on artificial intelligence (ECAI)
14. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
15. Dhariwal P, Hesse C, Klimov O, Nichol A, Plappert M, Radford A, Schulman J, Sidor S, Wu Y, Zhokhov P (2017) OpenAI baselines. https://github.com/openai/baselines
16. Fernando C, Banarse D, Blundell C, Zwols Y, Ha D, Rusu AA, Pritzel A, Wierstra D (2017) Pathnet: evolution channels gradient descent in super neural networks. arXiv:1701.08734
17. Gao P, Zhang Q, Wang F, Xiao L, Fujita H, Zhang Y (2020) Learning reinforced attentional representation for end-to-end visual tracking. Inf Sci 517:52–67
18. Ha D, Schmidhuber J (2018) Recurrent world models facilitate policy evolution. In: Advances in neural information processing systems, pp 2450–2462
19. Haarnoja T, Zhou A, Abbeel P, Levine S (2018) Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv:1801.01290

20. Haarnoja T, Zhou A, Hartikainen K, Tucker G, Ha S, Tan J, Kumar V, Zhu H, Gupta A, Abbeel P et al (2018) Soft actor-critic algorithms and applications. arXiv:1812.05905
21. Hansen N (2016) The CMA evolution strategy: a tutorial. arXiv:1604.00772
22. Igel C, Hansen N, Roth S (2007) Covariance matrix adaptation for multi-objective optimization. Evol Comput 15(1):1–28
23. Igel C, Heidrich-Meisner V, Glasmachers T (2008) Shark. J Mach Learn Res 9(Jun):993–996
24. Igel C, Suttorp T, Hansen N (2007) Steady-state selection and efficient covariance matrix update in the multi-objective CMA-ES. In: International conference on evolutionary multi-criterion optimization. Springer, Berlin, pp 171–185
25. Kauten C (2018) Super mario bros for OpenAI Gym. GitHub. https://github.com/Kautenja/gym-super-mario-bros
26. Lehman J, Chen J, Clune J, Stanley KO (2018) Safe mutations for deep and recurrent neural networks through output gradients. In: Proceedings of the genetic and evolutionary computation conference. ACM, pp 117–124
27. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2015) Continuous control with deep reinforcement learning. arXiv:1509.02971
28. Lizotte DJ, Bowling M, Murphy SA (2012) Linear fitted-Q iteration with multiple reward functions. J Mach Learn Res 13(Nov):3253–3295
29. Lizotte DJ, Bowling MH, Murphy SA (2010) Efficient reinforcement learning with multiple reward functions for randomized controlled trial analysis. In: International conference on machine learning. Citeseer, pp 695–702
30. Mannor S, Shimkin N (2004) A geometric approach to multi-criterion reinforcement learning. J Mach Learn Res 5(Apr):325–360
31. Mao H, Netravali R, Alizadeh M (2017) Neural adaptive video streaming with pensieve. In: Proceedings of the conference of the ACM special interest group on data communication. ACM, pp 197–210
32. Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K (2016) Asynchronous methods for deep reinforcement learning. In: International conference on machine learning, pp 1928–1937
33. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. arXiv:1312.5602
34. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529
35. Natarajan S, Tadepalli P (2005) Dynamic preferences in multi-criteria reinforcement learning. In: International conference on machine learning. ACM, pp 601–608
36. Ng AY, Harada D, Russell S (1999) Policy invariance under reward transformations: theory and application to reward shaping. In: International conference on machine learning, vol 99, pp 278–287
37. Nguyen TT (2018) A multi-objective deep reinforcement learning framework. arXiv:1803.02965
38. Osband I, Blundell C, Pritzel A, Van Roy B (2016) Deep exploration via bootstrapped DQN. In: Advances in neural information processing systems, pp 4026–4034
39. Osband I, Russo D, Van Roy B (2013) (More) efficient reinforcement learning via posterior sampling. In: Advances in neural information processing systems, pp 3003–3011
40. Osband I, Van Roy B (2015) Bootstrapped thompson sampling and deep exploration. arXiv:1507.00300
41. Osband I, Van Roy B, Wen Z (2014) Generalization and exploration via randomized value functions. arXiv:1402.0635
42. Parisi S, Pirotta M, Restelli M (2016) Multi-objective reinforcement learning through continuous pareto manifold approximation. J Artif Intell Res 57:187–227
43. Parisi S, Pirotta M, Smacchia N, Bascetta L, Restelli M (2014) Policy gradient approaches for multi-objective sequential decision making. In: International joint conference on neural networks. IEEE, pp 2323–2330
44. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L et al (2019) Pytorch: an imperative style, high-performance deep learning library. In: Advances in neural information processing systems, pp 8024–8035
45. Risi S, Togelius J (2015) Neuroevolution in games: state of the art and open challenges. IEEE Transactions on Computational Intelligence and AI in Games 9(1):25–41
46. Russo DJ, Van Roy B, Kazerouni A, Osband I, Wen Z et al (2018) A tutorial on thompson sampling. Foundations and Trends®, in Machine Learning 11(1):1–96
47. Salimans T, Ho J, Chen X, Sidor S, Sutskever I (2017) Evolution strategies as a scalable alternative to reinforcement learning. arXiv:1703.03864
48. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M et al (2016) Mastering the game of Go with deep neural networks and tree search. Nature 529(7587):484
49. Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic policy gradient algorithms. In: International conference on machine learning
50. Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A et al (2017) Mastering the game of Go without human knowledge. Nature 550(7676):354
51. Spiteri K, Urgaonkar R, Sitaraman RK (2016) BOLA: near-optimal bitrate adaptation for online videos. In: IEEE international conference on computer communications. IEEE, pp 1–9
52. Such FP, Madhavan V, Conti E, Lehman J, Stanley KO, Clune J (2017) Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv:1712.06567
53. Sullivan GJ, Wiegand T et al (1998) Rate-distortion optimization for video compression. IEEE Signal Proc Mag 15(6):74–90
54. Sutton RS, Barto AG et al (1998) Introduction to reinforcement learning, vol 135. MIT Press, Cambridge
55. Suttorp T, Hansen N, Igel C (2009) Efficient covariance matrix update for variable metric evolution strategies. Mach Learn 75(2):167–197
56. Tajmajer T (2017) Multi-objective deep Q-learning with subsumption architecture. arXiv:1704.06676
57. Tesauro G, Das R, Chan H, Kephart J, Levine D, Rawson F, Lefurgy C (2008) Managing power consumption and performance of computing systems using reinforcement learning. In: Advances in neural information processing systems, pp 1497–1504
58. Vamplew P, Dazeley R, Berry A, Issabekov R, Dekker E (2011) Empirical evaluation methods for multiobjective reinforcement learning algorithms. Mach Learn 84(1-2):51–80
59. Vamplew P, Yearwood J, Dazeley R, Berry A (2008) On the limitations of scalarisation for multi-objective reinforcement learning of Pareto fronts. In: Australasian joint conference on artificial intelligence. Springer, pp 372–378

60. Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double Q-learning. In: AAAI conference on artificial intelligence
61. Van Moffaert K, Drugan MM, Nowé A (2013) Hypervolume-based multi-objective reinforcement learning. In: International conference on evolutionary multi-criterion optimization. Springer, pp 352–366
62. Van Moffaert K, Drugan MM, Nowé A (2013) Scalarized multi-objective reinforcement learning: novel design techniques. In: 2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL). IEEE, pp 191–199
63. Voß T, Hansen N, Igel C (2010) Improved step size adaptation for the MO-CMA-ES. In: Annual conference on genetic and evolutionary computation. ACM, pp 487–494
64. Wiering MA, Withagen M, Drugan MM (2014) Model-based multi-objective reinforcement learning. In: 2014 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL). IEEE, pp 1–6
65. Yin X, Jindal A, Sekar V, Sinopoli B (2015) A control-theoretic approach for dynamic adaptive video streaming over HTTP. In: ACM SIGCOMM computer communication review, vol 45. ACM, pp 325–338
66. Zitzler E, Thiele L (1999) Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Trans Evol Comput 3(4):257–271

Diqi Chen received the B.E. degree from Peking University, Beijing, China, in 2014. He is currently pursuing the Ph.D. degree with the Institute of Computing Technology, Chinese Academy of Sciences and University of Chinese Academy of Sciences, Beijing, China. His research interests include perceptual image processing, computer vision and reinforcement learning

Yizhou Wang received the Ph.D. degree in computer science from the University of California at Los Angeles, Los Angeles, CA, USA, in 2005. He is currently a Professor of the Computer Science Department, Peking University, Beijing, China. He was a Research Staff of the Palo Alto Research Center (Xerox-PARC) from 2005 to 2008. His research interests include computer vision, statistical modeling and learning. Dr. Yizhou Wang is a BoYa Professor of Computer Science Department and the vice director of the Center on Frontiers of Computing Studies at Peking University. He received his Bachelor's degree in Electrical Engineering from Tsinghua University in 1996, and his Ph.D. in Computer Science from University of California at Los Angeles (UCLA) in 2005. He joined Xerox Palo Alto Research Center (Xerox PARC) as a research staff from 2005 to 2007. His research interests include computational vision, statistical modeling and learning, medical image analysis

Wen Gao received his PhD degree from University of Tokyo, Japan. He is now Boya Chair Professor and the founding director of National Engineering Lab. for Video Technology (NELVT) at Peking University. Prof. Gao works in the areas of multimedia and computer vision, topics including video coding, video analysis, multimedia retrieval, face recognition, multimodal interfaces, and virtual reality. He published seven books, over 300 papers in refereed journals, and over 700 papers in selected international conferences. He served or serves on the editorial board for several journals, such as IEEE T-IP, IEEE T-CSVT, IEEE T-MM, IEEE T-AMD. He chaired a number of prestigious international conferences on multimedia and video signal processing, such as IEEE ICME 2007, ACM Multimedia 2009, IEEE ISCAS 2013, and also served on the advisory and technical committees of numerous professional organizations. He is a fellow of IEEE, a fellow of ACM, and a member of Chinese Academy of Engineering

Affiliations

Diqi Chen[1] · Yizhou Wang[2] · Wen Gao[3]

Yizhou Wang
Yizhou.Wang@pku.edu.cn

Wen Gao
wgao@pku.edu.cn

1 Key Lab of Intelligent Information Processing,
Institute of Computing Technology and University
of Chinese Academy of Sciences, Beijing, China

2 National Engineering Lab for Video Technology,
Institute of Digital Media, Key Lab of Machine Perception (MoE),
School of Electronics Engineering and Computer Science (EECS),
Peking University, and Deepwise AI Lab, Beijing, China

3 National Engineering Lab for Video Technology,
Institute of Digital Media, Key Lab of Machine Perception (MoE),
School of Electronics Engineering and Computer Science (EECS),
Peking University, Beijing, China