Multi-Task Offline Reinforcement Learning with Heuristic Conservative Soft Actor-Critic

Project of AI3601 Reinforcement Learning, 2024 Spring, SJTU

Jiazi Bu*, Kailing Wang*, Xiangyuan Xue*†

School of Electronic Information and Electrical Engineering
*Equal Contribution   †Project Leader

Abstract

In this project, we propose a novel method named heuristic conservative soft actor-critic (HCSAC) based on conservative Q-learning (CQL) and soft actor-critic (SAC) to solve the offline multi-task reinforcement learning problem. We employ task embedding to allow the agent to learn shared representations across multiple tasks. We also follow conservative data sharing (CDS) to utilize heuristic relabeling to augment the offline dataset, so that the agent can benefit from the shared transitions while avoiding performance degradation. We conduct a series of experiments on the MuJoCo benchmark to verify the effectiveness of our method. The results show that our proposed methods in general bring superior performance and improve the robustness of the agent.

1 Introduction

Reinforcement learning (RL) describes the sequential decision-making process where an agent interacts with the environment to maximize the cumulative reward signal [20]. With the development of deep learning, deep reinforcement learning (DRL) has emerged as a powerful paradigm that combines deep neural networks with traditional reinforcement learning algorithms, making it possible to learn complex behavior representations from high-dimensional sensory inputs [13]. Such paradigm empowers the agent to learn effective behavior policies in various real-world scenarios such as robotics control [8], game playing [19], and autonomous driving [7].

In the traditonal setting, the agent learns in an online manner, which requires the agent to frequently interact with the environment to gather samples. However, such interaction can be extremely expensive in some real-world applications. On one hand, collecting real-world samples can hardly be parallelized, which makes the process remarkably slow. On the other hand, the agent may suffer from catastrophic failures during the learning process, which may not be affordable in safety-critical applications. To address these issues, researchers have proposed an alternative offline manner, where the agent learns from a fixed dataset without interacting with the environment [12] as shown in Figure 1. Admittedly, the offline reinforcement learning paradigm is attractive, but it also faces several challenges. The bootstrap method unavoidably introduces distribution shifts, which makes the traditional algorithms fail to learn a satisfactory policy. Researchers have proposed various methods to tackle this problem, such as implicit Q-learning [9], conservative Q-learning [11], and TD3-BC [3]. The core idea of these methods is to suppress the overestimation of the Q-value function, which is widely recognized as the main reason for the failure of offline reinforcement learning.

When it comes to scaling up the reinforcement learning algorithms, researchers find that learning multiple tasks simultaneously can effectively improve the sample efficiency and generalization ability of the agent as shown in Figure 2. Multi-task reinforcement learning (MTRL) thus becomes a popular research direction in recent years [22]. Researchers have proposed various methods for the
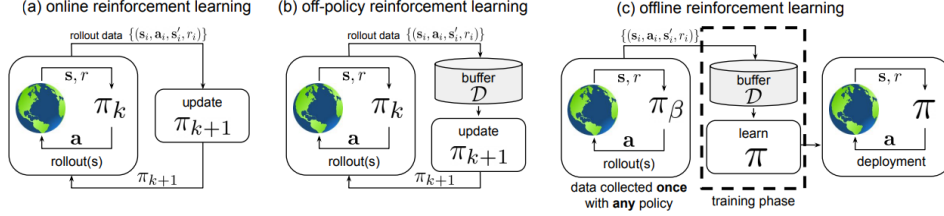
Figure 1: An illustration of online reinforcement learning, off-policy reinforcement learning, and offline reinforcement learning. Offline reinforcement learning follows an open-loop control paradigm, where the agent learns from a fixed dataset without interacting with the environment.

multi-task setting, including shared representation learning [1], knowledge transfer and distillation [23], and progressive neural networks [16]. However, most of these methods are designed for the online setting, so they will suffer from the same issues when applied to the offline setting. Some novel methods have been proposed to combine the offline and multi-task learning paradigms, such as conservative data sharing [24] and scaled Q-learning [10], but how to achieve reliable offline multi-task reinforcement learning still remains an open problem till today.
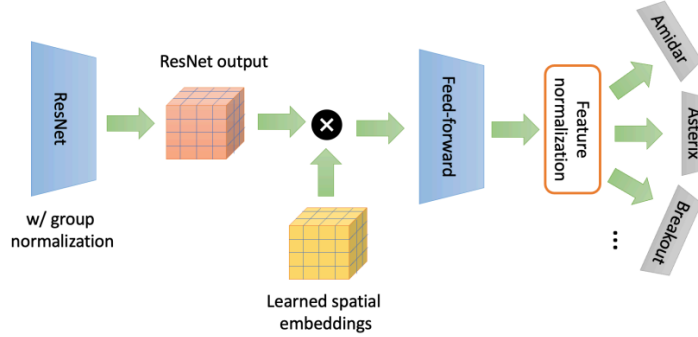

Figure 2: An example of multi-task reinforcement learning. Multiple policy heads share the same backbone as the feature extractor to learn useful shared representations, so that the agent can benefit from the extensive experience and knowledge transfer across multiple tasks.

In this project, we aim to explore the offline multi-task reinforcement learning paradigm. Our study pivots around improving the sample efficiency and generalization ability of the agent by fully exploiting the potential of multi-task learning. We propose a novel algorithm framework named heuristic conservative soft actor-critic (HCSAC) to solve the offline multi-task reinforcement learning problem. The framework is built upon the conservative Q-learning [11] and soft actor-critic [4] algorithms to suppress distribution shifts in the offline setting. We also introduce task embeddings to allow the agent to learn shared representations across multiple tasks. In addition, we follow conservative data sharing [24] and propose a heuristic strategy to relabel the dataset, so that the agent can benefit from the augmented transitions while avoiding performance degradation. We conduct a series of experiments on the MuJoCo benchmark [21] to evaluate the performance of our method. Through these efforts, we expect to provide some insights into the characteristics and efficacies of the proposed method, and to demonstrate the potential of offline multi-task reinforcement learning in real-world applications.

2 Method

2.1 Preliminaries

The goal of reinforcement learning is to learn a policy that maximizes the expected cumulative discounted reward in a Markov decision process (MDP) [20] defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s'|s,a)$ is the transition probability, $R(s,a)$ is the reward function, and $\gamma \in (0,1)$ is the discount factor. The agent interacts with the environment

by following a policy $\pi(a|s)$, which maps states to actions. The state value function $V^\pi(s)$ and the action-value function $Q^\pi(s, a)$ are defined as the expected cumulative discounted reward under policy π starting from state s and taking action a. The optimal value functions $V^*(s)$ and $Q^*(s, a)$ are defined as the maximum value functions over all policies, which satisfy the Bellman equation

$$V^*(s) = \max_\pi \mathbb{E}_{a \sim \pi(\cdot|s)} [Q^\pi(s, a)]$$

$$Q^*(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a'} Q^*(s', a') \right] \tag{1}$$

In the offline setting, the environment is not accessible when training, so the agent can only learn from a fixed dataset $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^N$ collected by a behavior policy $\pi_\beta(a|s)$, where s_i is the state, a_i is the action, r_i is the reward, s_i' is the next state, and N is the number of transitions. In the multi-task setting, the dataset $\mathcal{D} = \cup_{i=1}^M \mathcal{D}_i$ consists of transitions from multiple tasks, where $\mathcal{D}_i$ is the dataset of task i and M is the number of tasks. The goal is to learn a policy that maximizes the expected cumulative discounted reward across all the tasks.

2.2 Soft Actor-Critic

Soft actor-critic (SAC) is an actor-critic algorithm that maximizes the entropy-regularized expected cumulative discounted reward [4]. Different from the traditional actor-critic algorithms such as trust region policy optimization (TRPO) [17] and proximal policy optimization (PPO) [18], SAC follows an off-policy learning paradigm, which significantly improves the sample efficiency and can be better applied to the offline setting. Besides the cumulative reward, SAC also tries to maximize the entropy of the policy, which encourages exploration and prevents premature convergence to suboptimal policies. The objective function of SAC can be specified as

$$J(\pi) = \sum_{t=0}^\infty \mathbb{E}_{(s_t, a_t) \sim \rho_\pi} [R(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))] \tag{2}$$

where ρ_π is the state-action visitation distribution induced by the policy π, $\mathcal{H}(\pi(\cdot|s_t))$ is the entropy of the policy, and α is the temperature parameter that determines the relative importance of the entropy term. Then the policy evaluation step should be modified to match the entropy-regularized objective. The soft Q-value can be computed by iteratively applying a modified Bellman operator

$$\mathcal{T}^\pi Q(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\mathbb{E}_{a' \sim \pi(\cdot|s')} [Q(s', a') - \alpha \log \pi(a'|s')] \right] \tag{3}$$

Theoretical analysis shows that the soft Q-value will finally converge with enough iterations. At the same time, the policy improvement step should also be modified. SAC assumes that the policy distribution is proportional to the exponential of the soft Q-value, so the policy can be updated by minimizing the Kulback-Leibler (KL) divergence between the current policy and the normalized exponential of the soft Q-value, which can be formulated as

$$\pi'(\cdot|s) = \arg\min_\mu D_{\text{KL}} \left(\mu(\cdot|s) \left\| \frac{\exp(Q^\pi(s, \cdot))}{Z^\pi(s)} \right. \right) \tag{4}$$

where $Z^\pi(s)$ is the partition function that normalizes the exponential of the soft Q-value. Theoretical analysis shows that the policy is guaranteed to improve, so it will converge to the optimal policy.

In the actual implementation, several tricks are introduced to realize the expected algorithm. Firstly, SAC uses a Gaussian policy parameterized by a mean and a standard deviation, so the reparameterization trick is required to back propagate the gradient correctly. Secondly, SAC uses two Q-networks to estimate the soft Q-value and takes the minimum of the two results to avoid the overestimation of the Q-value. SAC also introduces two target Q-networks as in deep deterministic policy gradient (DDPG) [14] to update the Q-networks softly and stabilize the training process. Finally, we introduce automating entropy adjustment for maximum entropy [5] to adaptively choose the temperature parameter α during training. To be specific, it converts the entropy regularization term into a constraint and tries to solve the following constrained optimization problem

$$\max_\pi \sum_{t=0}^\infty \mathbb{E}_{(s_t, a_t) \sim \rho_\pi} [R(s_t, a_t)]$$

$$\text{s.t. } \mathbb{E}_{(s_t, a_t) \sim \rho_\pi} [-\log \pi(a_t|s_t)] \geq \mathcal{H}_0, \forall t \tag{5}$$

where $\mathcal{H}$ is the target entropy. This constrained optimization problem can be converted into its dual form and solved by the Lagrange multiplier method. The result can be formulated as

$$\alpha_t^* = \arg\min_{a_t} \mathbb{E}_{a_t \sim \pi(\cdot|s_t)} [-\alpha_t \log \pi(a_t|s_t) - \alpha_t \mathcal{H}_0] \tag{6}$$

3

where $\bar{\mathcal{H}}$ is a predefined minimum policy entropy threshold, which is empirically set to the negative of the action space dimension. Now the temperature parameter α can be automatically adjusted during the training process by gradient descent method. This trick is proved to be quite effective in practice and has been widely used in the SAC algorithm.

2.3 Conservative Q-Learning

Although researchers have proposed many methods for offline reinforcement learning, conservative Q-learning (CQL) [11] is still the most stable and effective method in practice. The key idea of CQL is to discourage the out-of-distribution Q-values while encouraging the Q-values in the dataset, so that the overestimation of the Q-values can be suppressed to some extent. The original form of CQL can be formulated as a family of optimization problems

$$\min_Q \max_\mu \alpha \left(\mathbb{E}_{s \sim \mathcal{D}, a \sim \mu(\cdot|s)} \left[Q(s,a) \right] - \mathbb{E}_{s \sim \mathcal{D}, a \sim \pi_\beta(\cdot|s)} \left[Q(s,a) \right] \right)$$
$$+ \frac{1}{2} \mathbb{E}_{(s,a,s') \sim \mathcal{D}} \left[\left(Q(s,a) - \mathcal{B}^{\pi_k} Q^k(s,a) \right)^2 \right] + \mathcal{R}(\mu) \tag{7}$$

where α is the temperature parameter that determines the relative importance of the conservative term, $\mathcal{B}^{\pi_k}$ is the Bellman operator under the behavior policy π_k, and $\mathcal{R}(\mu)$ is a general regularizer. If we choose $\mathcal{R}(\mu)$ to be the KL divergence between the current policy and the uniform distribution, the optimization problem can be rearranged into the following form

$$\min_Q \alpha \mathbb{E}_{s \sim \mathcal{D}} \left[\log \sum_a \exp(Q(s,a)) - \mathbb{E}_{a \sim \pi_\beta(\cdot|s)} \left[Q(s,a) \right] \right]$$
$$+ \frac{1}{2} \mathbb{E}_{(s,a,s') \sim \mathcal{D}} \left[\left(Q(s,a) - \mathcal{B}^{\pi_k} Q^k(s,a) \right)^2 \right] \tag{8}$$

In continuous action spaces, the log-sum-exp operation should be approximated by importance sampling, where we can rewrite the term as

$$\log \sum_a \exp(Q(s,a)) = \log \left(\frac{1}{2} \sum_a \exp(Q(s,a)) + \frac{1}{2} \sum_a \exp(Q(s,a)) \right)$$
$$= \log \left(\frac{1}{2} \mathbb{E}_{a \sim U(\mathcal{A})} \left[\frac{\exp(Q(s,a))}{U(a)} \right] + \frac{1}{2} \mathbb{E}_{a \sim \pi(\cdot|s)} \left[\frac{\exp(Q(s,a))}{\pi(a|s)} \right] \right) \tag{9}$$

where $U(\mathcal{A})$ is the uniform distribution over the action space. Then we can sample N actions each from the uniform distribution and the policy distribution to effectively estimate this term. Note that CQL only makes some modifications to the objective function of the Q-network, so it can be easily combined with other actor-critic algorithms. In practice, CQL is usually combined with SAC to solve the offline reinforcement learning problems with continuous action spaces.

2.4 Task Embedding

Since we expect the agent to learn multiple tasks simultaneously, we need a machanism to explicitly inform the agent of which task it is currently facing. The simplest way is to separately train a policy for each task, but this method is not efficient enough and cannot fully exploit the shared representations among tasks. Scaled Q-learning [10] uses multiple policy heads for each task which share the same backbone as the feature extractor, so that the agent can benefit from the shared feature representations across tasks. Conservative data sharing [24] appends a one-hot vector to the state space to indicate the task, so that the agent can distinguish the tasks and learn specific policies.

Inspired by the embeddings used in transformers, we propose a similar method to specify the task information. As shown in Figure 3, we assign a unique identity k to each task. The identity will be fed into an embedding layer to generate the task embedding e, which will be added onto the state s as the task-aware state z. The task-aware state z will be fed together with the action a into the critic network to output the estimated Q-value $\hat{q}$. It will also be fed alone into the actor network to output the predicted action $\hat{a}$. Note that the task embedding e can be learned during the training process, which ensures that the performance of the agent will not degrade in the multi-task setting.
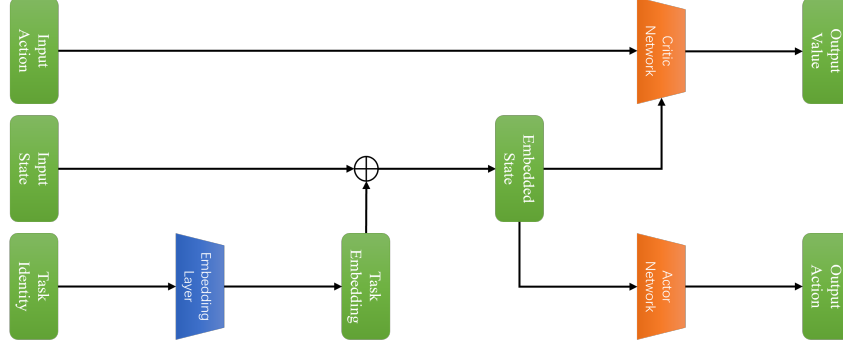
Figure 3: An illustration of the task embedding. The task identity k is fed into an embedding layer to generate the task embedding e, which will be added onto the state s as the task-aware state z. The task-aware state z will be fed together with the action a into the critic network to output the estimated Q-value $\hat{q}$. It will also be fed alone into the actor network to output the predicted action $\hat{a}$.

2.5 Heuristic Relabeling

Previous works [24] have verified that one of the main reason for the failure in the multi-task setting is distribution shifts. If the agent utilizes the transitions from all the tasks without making any distinction, those out-of-distribution transitions will significantly hinder the learning process. Inspired by the idea of conservative data sharing [24], one feasible solution is to share the transitions conservatively. Luckily, conservative Q-learning has already provided the conservative Q-values as an indicator to measure the distribution shifts, so we define the relabeling threshold $\tau(\mathcal{D}_i)$ as

$$\tau(\mathcal{D}_i) = \{Q_i(s,a)|(s,a) \in \mathcal{D}_i\}_{k\%} \tag{10}$$

where $Q_i(s,a)$ is the Q-value of the transition (s,a) under the policy of task i, and $\{\cdot\}_{k\%}$ denotes the $k\%$ quantile of the set. Then for any transition $(s,a,r,s') \in \mathcal{D}_j$, we can relabel j as i only if the conservative Q-value of the transition (s,a) under the policy of task i is no smaller than the relabeling threshold of $\mathcal{D}_i$, which can be formulated as

$$\Delta(s,a) = Q_i(s,a) - \tau(\mathcal{D}_i) \geq 0 \tag{11}$$

Then we can extend the dataset $\mathcal{D}$ by adding the transitions that satisfy the relabeling condition, which can benefit the learning process of the agent. Note that the Q-network is changing during the training process, so we only relabel the dataset once in a while to ensure the stability of training. The overall pipeline of the heuristic relabeling algorithm is summarized in Algorithm 1.

Algorithm 1 Heuristic Relabeling

Require: The multi-task offline dataset $\mathcal{D} = \cup_{i=1}^{M} \mathcal{D}_i$.
 1: Initialize the active dataset $\mathcal{D}^* \leftarrow \mathcal{D}$.
 2: **for** $k \leftarrow 1, 2, \cdots, L$ **do**
 3:     Train the agent using samples from $\mathcal{D}^*$ to obtain the latest critic network Q.
 4:     **for** $i \leftarrow 1, 2, \cdots, M$ **do**
 5:         Initialize the active dataset $\mathcal{D}_i^* \leftarrow \mathcal{D}_i$.
 6:         Compute the relabeling threshold $\tau(\mathcal{D}_i) \leftarrow \{Q(s,a)|(s,a) \in \mathcal{D}_i\}_{k\%}$.
 7:         **for** $j \leftarrow 1, 2, \cdots, M$ where $j \neq i$ **do**
 8:             Extend dataset by relabeling $\mathcal{D}_i^* \leftarrow \mathcal{D}_i^* \cup \{(s,a,r,s') \in \mathcal{D}_j | Q_i(s,a) \geq \tau(\mathcal{D}_i)\}$.
 9:         **end for**
10:     **end for**
11:     Merge the active dataset $\mathcal{D}^* \leftarrow \cup_{i=1}^{M} \mathcal{D}_i^*$.
12: **end for**

We can see that the time complexity of a single heuristic relabeling is $O(M^2 N)$, where M is the number of tasks and N is the number of transitions in each task. But the heuristic relabeling process will not take too much time because the dataset only updates every hundreds of epochs.

3 Experiments

3.1 Experiment Setup

We conduct all the experiments based on the customized MuJoCo Walker2D environment [21]. The walker is a two-dimensional two-legged figure that consist of seven main body parts: a single torso at the top, two thighs in the middle below the torso, two legs in the bottom below the thighs, and two feet attached to the legs on which the entire body rests. The goal is to walk or run in the in the forward direction by applying torques on the six hinges connecting the seven body parts. The state space and the action space contain 24 and 6 dimensions respectively. The two tasks have exactly the same state space, action space and transition probability. The only difference is the reward function.

The dataset consists of offline samples from two tasks (i.e. `walk` and `run`), each including two data quality levels (i.e. `medium` and `medium-replay`). They are collected from the replay buffer when training an online TD3 agent. In this project, the agent can only learn from the offline dataset when training while the evaluation is conducted online [2]. We expect the multi-task agent to learn from all the samples in the dataset and achieve satisfactory performance in both tasks.

According to Section 2, we implement three algorithms step by step: conservative soft actor-critic (CSAC), multi-task conservative soft actor-critic (MTCSAC), and heuristic conservative soft actor-critic (HCSAC). CSAC is the classic conservative Q-learning (CQL) algorithm based on the soft actor-critic (SAC) framework. MTCSAC adds the task embedding based on CSAC, which can realize multi-task learning. HCSAC further introduces the heuristic relabeling method to extend the offline dataset, which may further improve the performance of MTCSAC. All the critic and actor networks are implemented as multi-layer perceptrons (MLPs) [15] with 3 hidden layers and 256 hidden units. The output action is mapped to the range $(-1, 1)$ by a $\tanh$ function.

The experiments are divided into two groups (i.e. `medium` and `medium-replay`) according to the dataset used for training. In each group, CSAC is trained on the two tasks (i.e. `walk` and `run`) separately, while MTCSAC and HCSAC are trained on the union of the datasets for the two tasks. All the agents are trained for 1000 epochs with a batch size of 256. Online evaluation is conducted every 10 epochs. Heuristic relabeling is conducted every 100 epochs for HCSAC. We use the Adam algorithm [6] to optimize the neural networks, where the learning rate is set to 3×10^{-4} for both the critic and actor networks. The soft update rate is set to 5×10^{-3} to update the target networks. The discount factor is set to 0.99. The weight of the CQL loss term is set to 5.0. We sample 10 actions each from the uniform distribution and the policy distribution to estimate the log-sum-exp term in the CQL loss. The quantile is set to 5% in the heuristic relabeling algorithm for HCSAC.
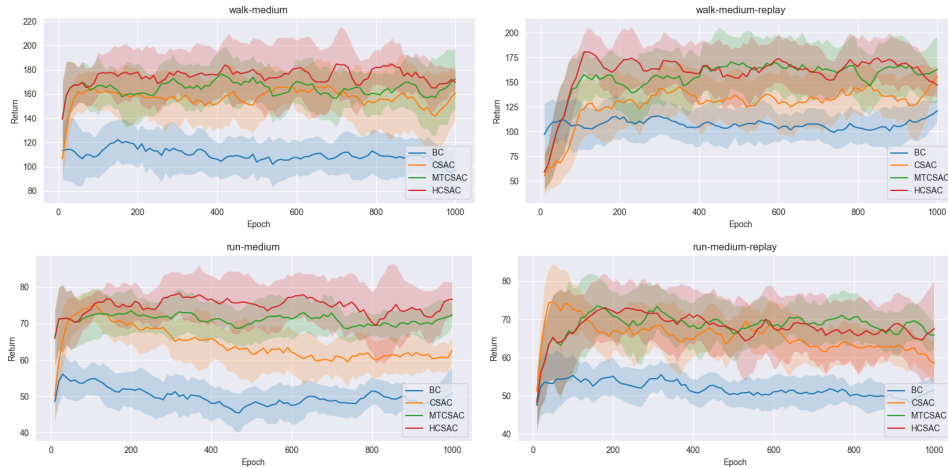
3.2 Performance Evaluation

Figure 4: Training curves of the agents on the Walker2D environment. The top row shows the results of the `walk` task and the bottom row shows the results of the `run` task. The left column shows the results of the `medium` dataset and the right column shows the results of the `medium-replay` dataset.

6

Figure 4 shows the training curves of the agents on the Walker2D environment. We can see that MTCSAC (green curve) and HCSAC (red curve) significantly outperform CSAC (orange curve) in all the tasks, which demonstrates that the multi-task setting benefits the training process so that the agent can learn useful shared representations from different tasks. HCSAC performs slightly better than MTCSAC in majority of the training process, especially for the `medium` quality dataset, which indicates that the heuristic relabeling method can effectively augment the offline dataset so that the agent can learn from more valuable transitions. We can expect that HCSAC will achieve even better performance on the dataset with higher quality. The averages return in the `run` task are much lower than those in the `walk` task because the former is more challenging and requires precise control, which is consistent with the sampled trajectories in the offline dataset.

Table 1: Performance evaluation of the agents on the Walker2D environment. The experiments are divided into two groups according to the dataset used for training, each containing two tasks. Every single experiment is conducted with 5 different random seeds and the returns are reported in the form of mean and standard deviation. The best results are highlighted in bold.

Dataset	Task	BC	CSAC	MTCSAC	HCSAC
`medium`	`walk`	113.96 ± 34.86	163.10 ± 21.00	170.22 ± 19.61	$\mathbf{173.41 \pm 11.72}$
	`run`	54.71 ± 7.88	$\mathbf{77.23 \pm 7.11}$	72.46 ± 6.25	76.06 ± 5.61
	`average`	84.34 ± 21.39	120.17 ± 14.56	121.34 ± 12.93	$\mathbf{124.74 \pm 8.67}$
`replay`	`walk`	136.78 ± 20.96	131.73 ± 39.88	$\mathbf{184.32 \pm 20.23}$	158.17 ± 25.72
	`run`	54.91 ± 3.39	$\mathbf{76.65 \pm 4.36}$	66.62 ± 11.22	69.44 ± 9.14
	`average`	95.85 ± 18.18	104.19 ± 22.26	$\mathbf{125.47 \pm 15.72}$	113.81 ± 17.43

Table 1 shows the quantitative evaluation of the agents on the Walker2D environment. We provide behavioral cloning (BC) as the weak baseline. CSAC is selected as the strong baseline without multi-task learning. According to the given quantitative results, MTCSAC and HCSAC achieve higher average returns with lower standard deviations compared to CSAC in most settings, which indicates that our proposed multi-task learning methods including task embedding and heuristic relabeling in general bring superior performance and improve the robustness of the agent.

4 Conclusion

In this project, we propose a novel method named heuristic conservative soft actor-critic (HCSAC) to solve the offline multi-task reinforcement learning problem. The method is developed based on the classic algorithms of conservative Q-learning (CQL) and soft actor-critic (SAC) to suppress distribution shifts in the offline setting. We further propose task embedding to allow the agent to learn shared representations across multiple tasks. In addition, we follow conservative data sharing (CDS) to propose heuristic relabeling to augment the offline dataset, so that the agent can benefit from the shared transitions while avoiding performance degradation. We conduct a series of experiments on the MuJoCo benchmark to verify the effectiveness of our method. The results show that our proposed methods in general bring superior performance and improve the robustness of the agent.

We should point out that it is not sufficient to evaluate our methods only on the MuJoCo benchmark. On one hand, the offline transitions are only shared across two tasks, where `walk` and `run` are both low-level locomotion skills, so it is unnecessary to share any part of the policy. In real-world scenarios, we actually expect the agent can learn from a large number of tasks to acquire those essential skills. On the other hand, MuJoCo is an environment with low-dimensional state spaces where the vectorized observations already contain all the information needed for robotic control, so the multi-task learning methods cannot be fully exploited. We assert that our methods will show more advantages in the high-dimensional vision-based environments, where the agent can utilize transitions from different tasks to learn better shared representations.

References

[1] Diana Borsa, Thore Graepel, and John Shawe-Taylor. Learning shared representations in multi-task reinforcement learning. *arXiv preprint arXiv:1603.02041*, 2016.

[2] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[3] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in neural information processing systems*, 34:20132–20145, 2021.

[4] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

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

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.

[8] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[9] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[10] Aviral Kumar, Rishabh Agarwal, Xinyang Geng, George Tucker, and Sergey Levine. Offline q-learning on diverse multi-task data both scales and generalizes. *arXiv preprint arXiv:2211.15144*, 2022.

[11] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[12] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[13] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[14] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[15] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[16] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

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

[18] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[19] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[20] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. *Robotica*, 17(2):229–235, 1999.

[21] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[22] Nelson Vithayathil Varghese and Qusay H Mahmoud. A survey of multi-task deep reinforcement learning. *Electronics*, 9(9):1363, 2020.

[23] Zhiyuan Xu, Kun Wu, Zhengping Che, Jian Tang, and Jieping Ye. Knowledge transfer in multi-task deep reinforcement learning for continuous control. *Advances in Neural Information Processing Systems*, 33:15146–15155, 2020.

[24] Tianhe Yu, Aviral Kumar, Yevgen Chebotar, Karol Hausman, Sergey Levine, and Chelsea Finn. Conservative data sharing for multi-task offline reinforcement learning. *Advances in Neural Information Processing Systems*, 34:11501–11516, 2021.

A Member Contribution

All the members contribute equally to this project, and the specific contributions are as follows:

Jiazi Bu. Investigate the related work about the provided topic, participate in the discussion of the framework design, and complete part of the report writing.

Kailing Wang. Run the experiments including model training and evaluation, analyze the results, tune the hyperparameters, and complete part of the report writing.

Xiangyuan Xue. Participate in the discussion of the framework design, implement the code of the proposed method, complete part of the report writing, and finally present the project.