# Multitask Actor-Critic Learning as Approach to Lifelong Learning

Anze Liu

UC Berkeley

Dec 14, 2022

## Abstract

In lifelong learning, an agent should continually learn multiple tasks while encountering new tasks over a period of time, which, to achieve effectively, requires the ability to 1) retain knowledge learned from different tasks and 2) selectively apply knowledge from previously learned tasks to the learning of new tasks. When tasks share similar characteristics, there is a potential for the agent to reuse learned information to achieve greater performance or learn more efficiently on unseen tasks, which helps to render the learning agent to be more applicable to the real-world.

Multitask learning enables agents to learn multiple tasks and construct a representation from shared characteristics to generalize to a larger variety of unseen tasks, which is potentially more efficient than learning each task independently in a linear order. However, a Multitask Learning approach to Lifelong Learning is challenged by Catastrophic Forgetting. Catastrophic Forgetting is the tendency of neural networks to forget the knowledge learned from previous tasks upon learning newly encountered tasks, which is a manifestation of the stability-plasticity dilemma. For a Lifelong Learning system to effectively learn multiple tasks sequentially, it is essential to maintain a balance between plasticity, the ability to integrate new knowledge, and stability, the quality of retaining previously concluded knowledge.

To maintain this balance of information from previous and future tasks, various factors should be considered, adjusted, and experimented. This project focuses on designing and implementing different advantage actor-critic based model architectures for multitask learning such that knowledge can be maintained and transferred between morphologically different agents/tasks. In addition, the multitask off-policy actor-critic algorithm is adapted to lifelong learning and the challenge of catastrophic forgetting is explored. Experiments are conducted with actor-critic variants to address multitask learning, for instance, multi-critic actor learning.

In summary, an agent may want to learn multiple tasks simultaneously because it accelerates the acquisition of all tasks being learned and it provides a better initialization for learning a new task. The ability to learn multiple tasks can be then used to approach a solution for continual learning.

# Multitask Off-Policy Actor Critic Reinforcement Learning

Multitask: Multitask learning is when an agent learns multiple tasks simultaneously with the objective of learning the different tasks more efficiently than learning the tasks individually in a linear way.

Actor Critic: actor critic algorithm utilizes an actor network, which learns the policy, and a critic network, which learns the Q-function, to enable the learning agent to achieve high return in the task environment.

Off-policy: Off-policy actor-critic is used in this project; taking an action according to the current policy, storing the obtained trajectories in a replay buffer, and sampling from the replay buffer during training to update the actor and critic network.

Reinforcement Learning: Lifelong learning is challenging to implement with deep neural networks because it aims to learn new knowledge from a sequence of tasks over a period of time instead of all at once. In contrast to an end-to-end training neural network where all parameters of the model are trained together simultaneously, a network where parameter tuning does not entirely overwrite existing parameters is more suitable for continual learning. Additionally, in Lifelong Learning, an agent makes decisions while learning tasks sequentially, which requires the ability to retain knowledge from different tasks and to transfer knowledge from previously learned tasks to the learning of new tasks. Therefore, this project, which aims to investigate the structure of the deep neural network in maintaining stability-plasticity balance in the context of the state-action based decision making, is relevant to Deep Reinforcement Learning.

Source of Data: MuJoCo (Multi-Joint dynamics with Contact), a simulation environment and physics engine, will be used as a simulator to generate data for this project's investigation in Lifelong Learning because the OpenAI Gym contains various highly optimized simulation environments for RL.

The goal of multitask reinforcement learning is to solve a distribution of multiple Markov Decision Processes (MPDs) to maximize the expected return. To generate the tasks to be used for this project's experiments, the HalfCheetah from MuJoCo simulator is modified to have different morphologies. The HalfCheetah is a 2-dimensional robot with 9 links and 8 joints, to which torque is applied to enable the cheetah to run forward. To achieve high return, the cheetah needs to run as fast as possible to cover a longer moving distance.

| **HalfCheetah** | **Feature** | **Torso Width** | **Thigh + Foot Width** | **Torso From-To** | **Torso Length** |
|---|---|---|---|---|---|
| v4 | Default | 0.046 | 0.046 | -.5 0 0 .5 0 0 | 10 |
| A | Small Torso | 0.02 | 0.046 | -.5 0 0 .5 0 0 | 10 |
| B | Large Torso | 0.08 | 0.046 | -.5 0 0 .5 0 0 | 10 |
| C | Small Leg | 0.046 | 0.02 | -.5 0 0 .5 0 0 | 10 |
| D | Large Leg | 0.046 | 0.08 | -.5 0 0 .5 0 0 | 10 |
| E | Long Torso | 0.046 | 0.046 | -.8 0 0 .8 0 0 | 16 |
| F | Short Torso | 0.046 | 0.046 | -.3 0 0 .3 0 0 | 6 |

Figure 1: A table displaying the modification to HalfCheetah's morphology. Actor-critic algorithm is performed on each of the tasks individually, which are used as the baselines for multitask learning.

# Actor Critic Algorithm

A challenge of policy gradients methods in reinforcement learning is the high variance of the gradient estimator:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

where $r(\tau)$ is the reward of a trajectory $\tau = \{s_i, a_i, r_i\}$. Higher reward increases the probability of action in trajectory, whereas lower reward decreases the probability of action. However, credit assignment to actions affecting future rewards is difficult in policy gradient methods, resulting in high variance. Instead of using the reward of an entire trajectory, a cumulative future reward from a state at a time step multiplied by a discount factor can be used to reduce variance and to discount the delayed effects. In addition, a baseline function in terms of the state can be subtracted from the discounted reward at the state such that the action resulting in higher return than the expected value is more likely to be chosen. Therefore, the difference between Q-function and value function, $Q^\pi(s_t, a_t) - V^\pi(s_t)$, indicates whether the action in this state should have higher probability or not; if the difference is large, the action is preferred in that state. Actor-critic algorithm combines policy gradients and Q-learning by training both the actor network, which is the policy, and the critic network, which is the Q-function. The actor indicates what action to take in a specific state while the critic provides feedback to the actor regarding how the actor should be updated. An advantage function is defined to be the difference between Q-function and value function, in other words, advantage is a measure of how an action is a state is better than what is expected.

In general, actor-critic uses a critic network to estimate the sum of rewards by computing the policy gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{it} \mid s_{it}) A^\pi(s_{it}, a_{it})$$

where the advantage is estimated to be

$$A^\pi(s_t, a_t) \approx r(s_t, a_t) + \gamma V_\phi^\pi(s_{t+1}) - V_\phi^\pi(s_t)$$

Value function is updated with the target values

$$y_t = r(s_t, a_t) + \gamma V^\pi(s_{t+1})$$

using the regression objective

$$\min \sum_{i,\, t} \left(V_\phi^\pi(s_{it}) - y_{it}\right)^2$$

## Experiment Setup

The following hyperparameters are used for experiments:

| | |
|---|---|
| episode length | 15 |
| discount factor for estimating advantage | 0.90 |
| number of iterations | 150 |
| number of hidden layers | 2 |
| size of hidden layer | 32 |
| batch size - steps collected per train iteration | 30000 |
| eval batch size - steps collected per eval iteration | 1500 |
| learning rate | 0.02 |
| number of target updates | 1 |
| number gradient steps per target update | 100 |

Figure 2: A table displaying the hyperparameters used for all experiments to maintain consistency and fairness.

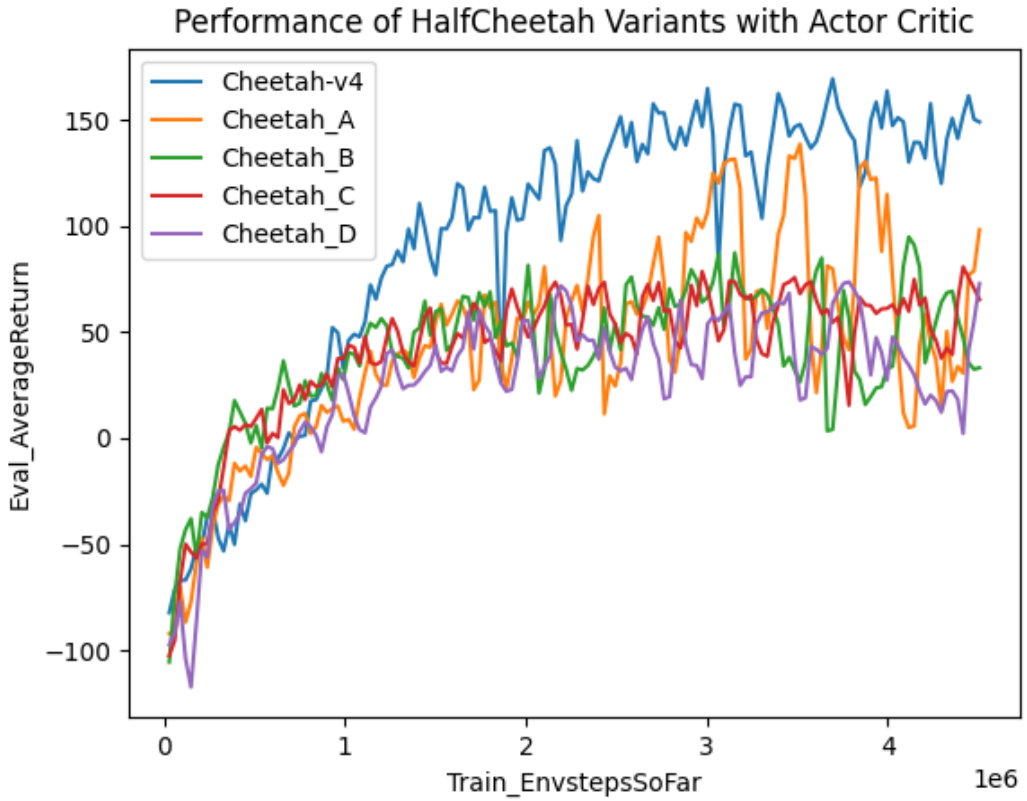



Figure 3: A graph showing the learning curve of performing Actor Critic on each of the HalfCheetah variants.

## Multitask Learning Algorithm

In Multitask Learning, a set of related tasks from an environment is performed by individual agents to be learned simultaneously.

A task is defined as $T_i = \{p_i(\mathbf{x}), p_i(\mathbf{y} \mid \mathbf{x}), L_i\}$, where $L_i$ is the loss of a task i and $p_i$ is the data generating distribution.

Instead of learning a single task model $f_\theta(\mathbf{y} \mid \mathbf{x})$, multitask learning trains a model $f_\theta(\mathbf{y} \mid \mathbf{x}, \mathbf{z_i})$ that depends on a task index $\mathbf{z_i}$. The subsequent questions are:

1) how to condition on the task index
2) what is the objective function
3) how to optimize the objective to train a model capable of performing multiple tasks.

Choosing how to perform task index conditioning is equivalent to choosing how and where to split model parameters into shared parameters and task specific parameters. Some common choices are concatenation-based conditioning, additive conditioning, multi-head architecture, and multiplicative conditioning. This project focuses on designing and implementing multi-head architecture based on the actor-critic algorithm in order to achieve multitask learning.

**Algorithm 1**: multitask actor-critic with a shared layer followed by task specific head

Initialization:
- Initialize a critic network with shared layer
- For each task $T_i$ in the given list of tasks:
  - Initialize a task specific head following the shared layer of critic
- Initialize an actor network with shared layer
- For each task $T_i$ in the given list of tasks:
  - Initialize a task specific head following the shared layer of actor

For each training iteration:
- For each task $T_i$ in the given list of tasks:
  - Take action $a \sim \pi_\theta(a \mid s)$, get trajectories $(s, a, s', r)$, store in task specific replay buffer R
- Sample a batch $\{s_i, a_i, r_i, s_i'\}$ from buffer for each task
- For each task $T_i$ in the given list of tasks:
  - Update value function $\hat{V}_\phi^\pi$ using target $y_{it} = r_{it} + \gamma \hat{V}_\phi^\pi(s_{it}')$ for each $s_{it}$
  - Evaluate $\hat{A}^\pi(s_{it}) \approx r(s_{it}, a_{it}) + \gamma V_\phi^\pi(s_i') - \hat{V}_\phi^\pi(s_i)$
  - Compute gradient $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{it} \mid s_{it}) A^\pi(s_{it}, a_{it})$
  - Update network parameters $\theta$ with the gradient and learning rate.

Algorithm 1 is used as the foundation for the algorithms and models designed and implemented in this project.

Sanity Check after implementing multitask model: Running a single HalfCheetah-v4 task on the actor critic Multitask Learning Model with the same seed should result in the same learning curve as running it on the original actor critic single task model.

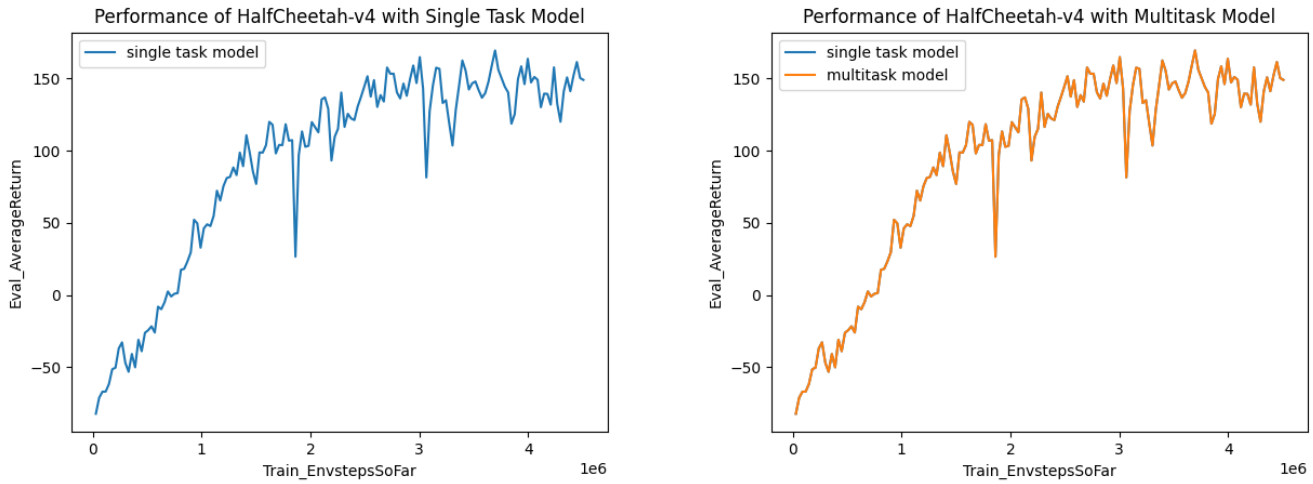



Figure 4: Graph showing the average return of HalfCheetah-v4 with the original single task actor critic model and with the multitask actor critic model under the same seed.

Challenges: Multitask Learning requires experimenting in a space of large numbers of parameters to search for an optimal way of sharing parameters among the different tasks, for example, the layers of the model, weights of each layer, and how different task losses contribute are weighted differently.

## Multitask Learning Architecture 1

### Algorithm and Model Architecture Description

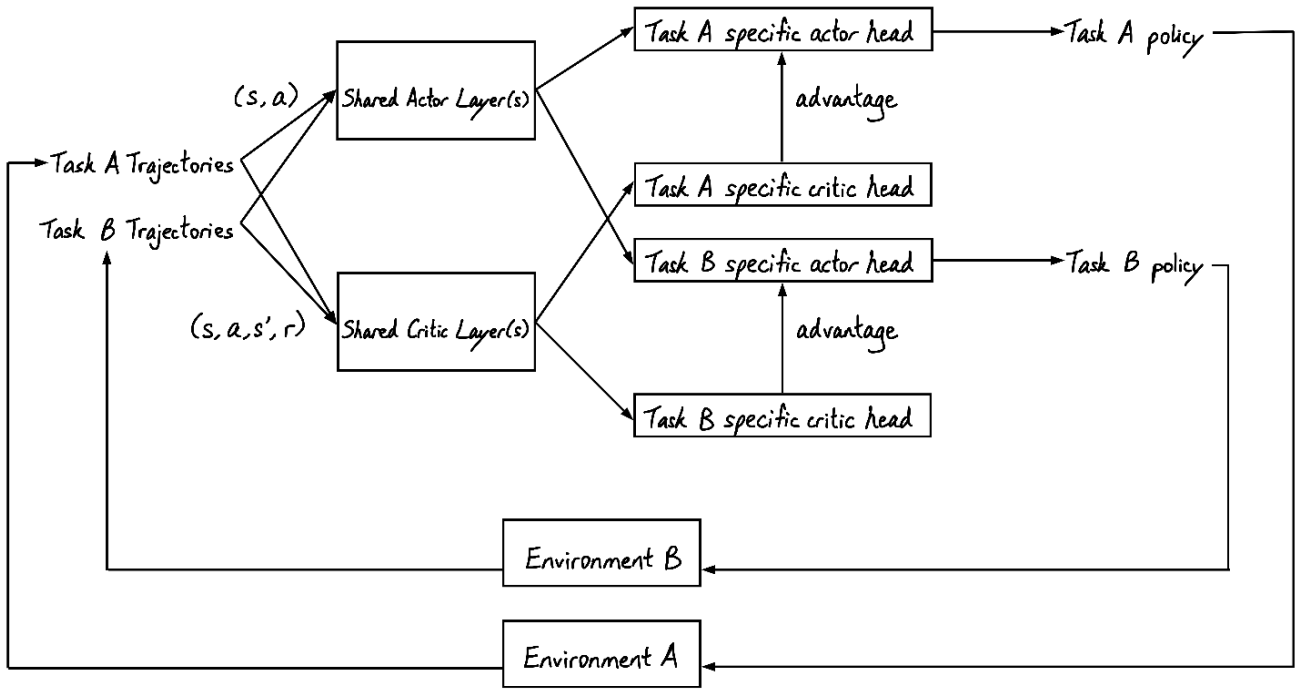



Figure 5: Diagram showing the algorithm of multitask learning with architecture 1.

In Architecture 1, each agent performing a task has access to a shared actor layer and critic layer, followed by a task specific actor head and a task specific critic head, respectively.

New parameters:

| | |
|---|---|
| multitask_learning | True if performing multitask actor critic learning |
| num_tasks | Number of tasks = number of workers for Q function and value function network |
| tasks | List of tasks to be learned simultaneously |

## Experimental Evaluation

Learning 2 Tasks: HalfCheetah-v4 and HalfCheetah_A, with the Multitask Learning Model

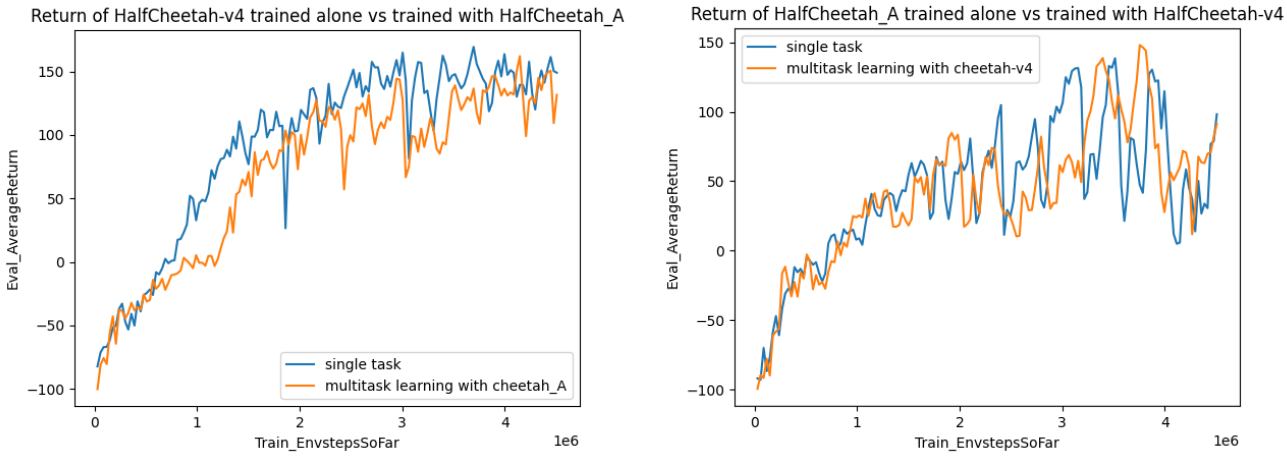
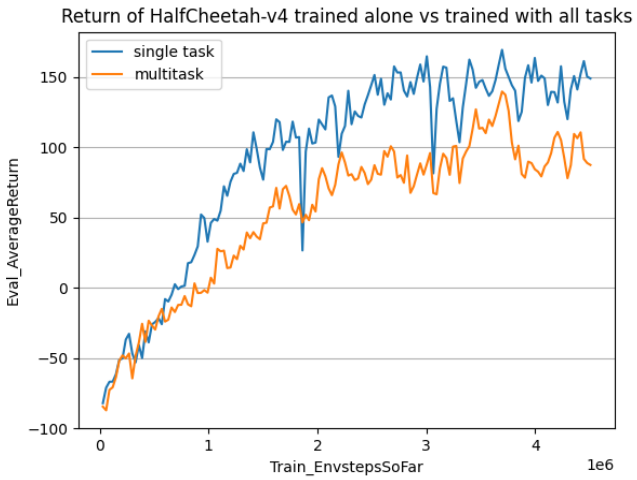



Figure 6: Graph showing the average return of HalfCheetah-v4 and HalfCheetah_A after being trained together using the multitask learning model architecture 1. The average return of HalfCheetah-v4 using architecture 1 does not achieve the average return baseline.

Learning All 7 Tasks: Running multitask learning with multi-head architecture on all 7 morphologically different tasks.



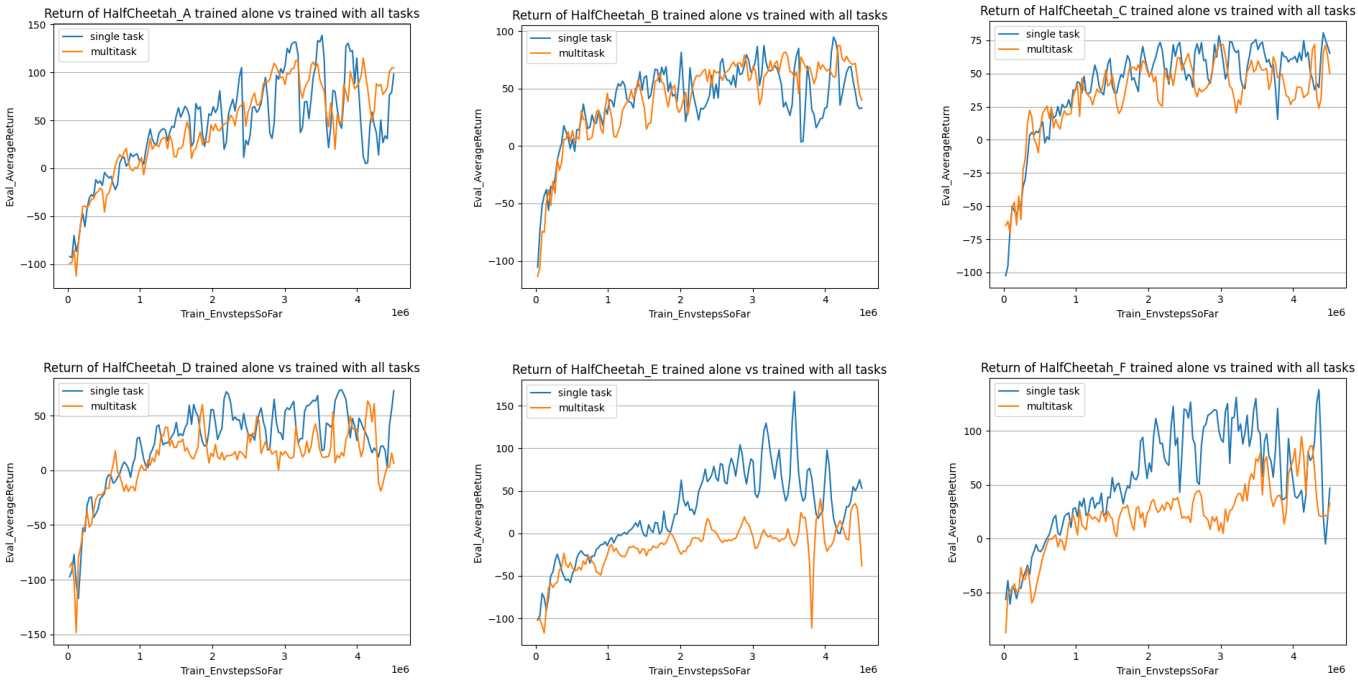



Figure 7: Graphs showing the average returns of the 7 HalfCheetah variants using multitask actor-critic algorithm compared with the returns from single task actor-critic.

## Multitask Learning Architecture 2

### Model Architecture Description

Sharing actor, critic with shared layer and task-specific heads

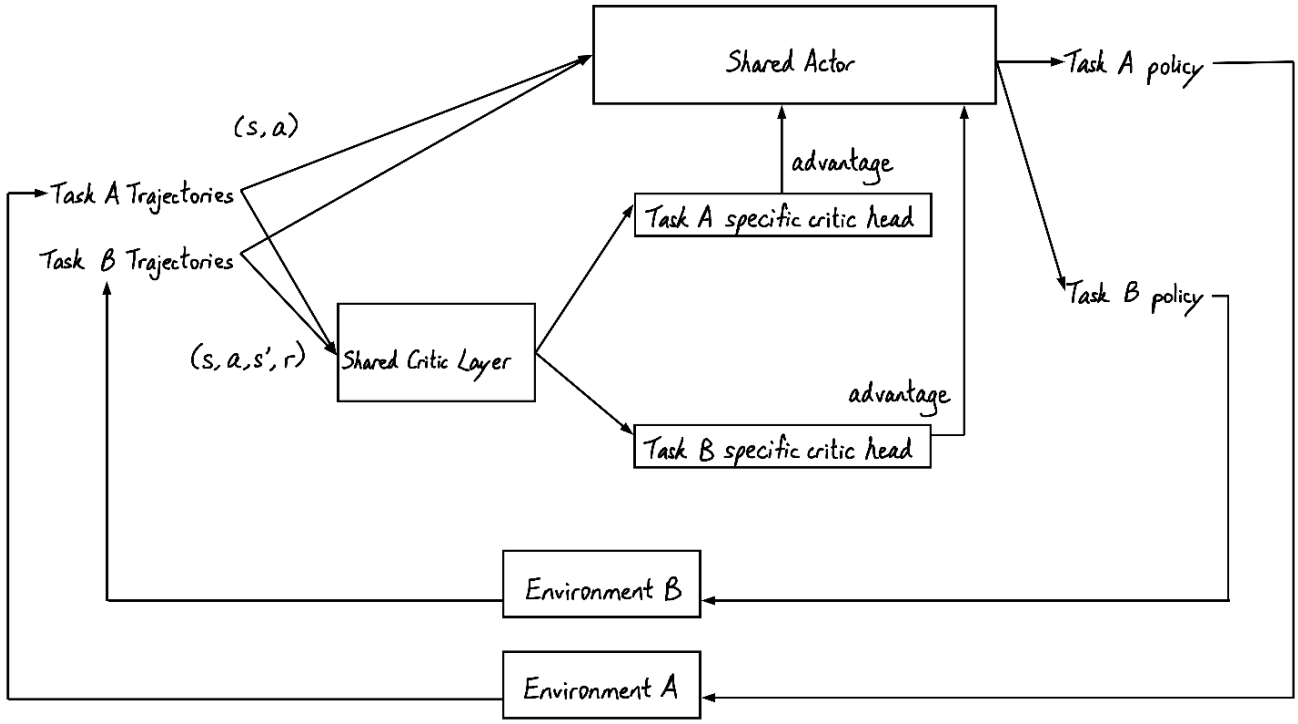



Figure 8: Diagram showing the algorithm of multitask learning with architecture 2.

Similar to Architecture 1, Architecture 2 uses a critic with a shared layer and task specific heads. Different from Architecture 1, Architecture 2 uses a single shared actor to learn the policies for all tasks.

## Experimental Evaluation

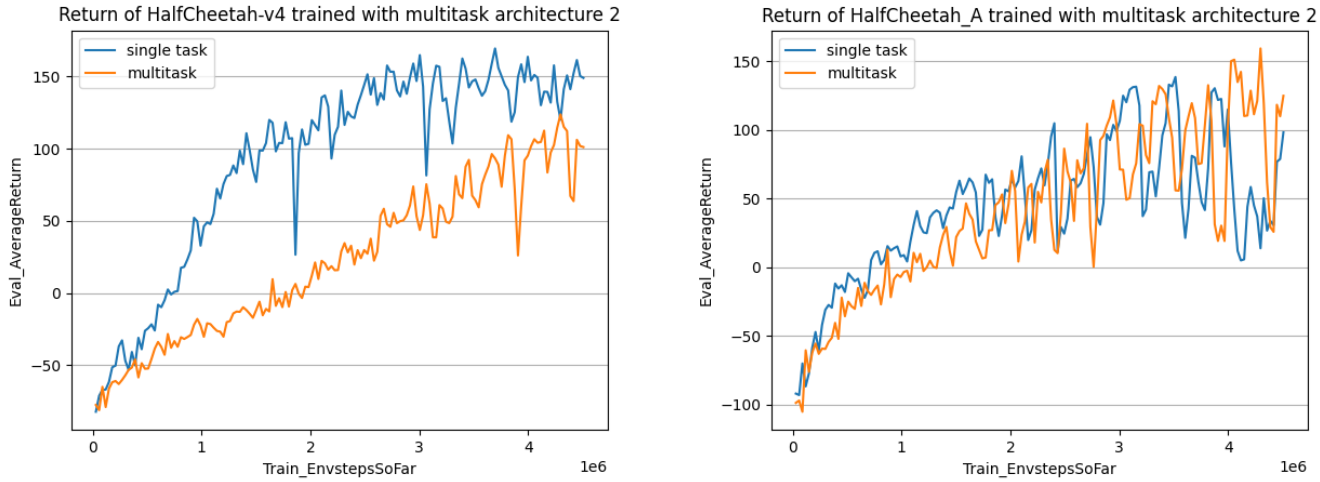



Figure 9: Graph showing the average return of HalfCheetah-v4 and HalfCheetah_A using multitask learning architecture 2. The return of HalfCheetah-v4 using architecture 2 does not reach the same level of return as using the single task actor critic algorithm. The single policy network is not capable of learning the policies for different tasks.

# Multitask Learning Architecture 3

## Model Architecture Description

Shared critic, shared layer + task specific head critic, task specific actor

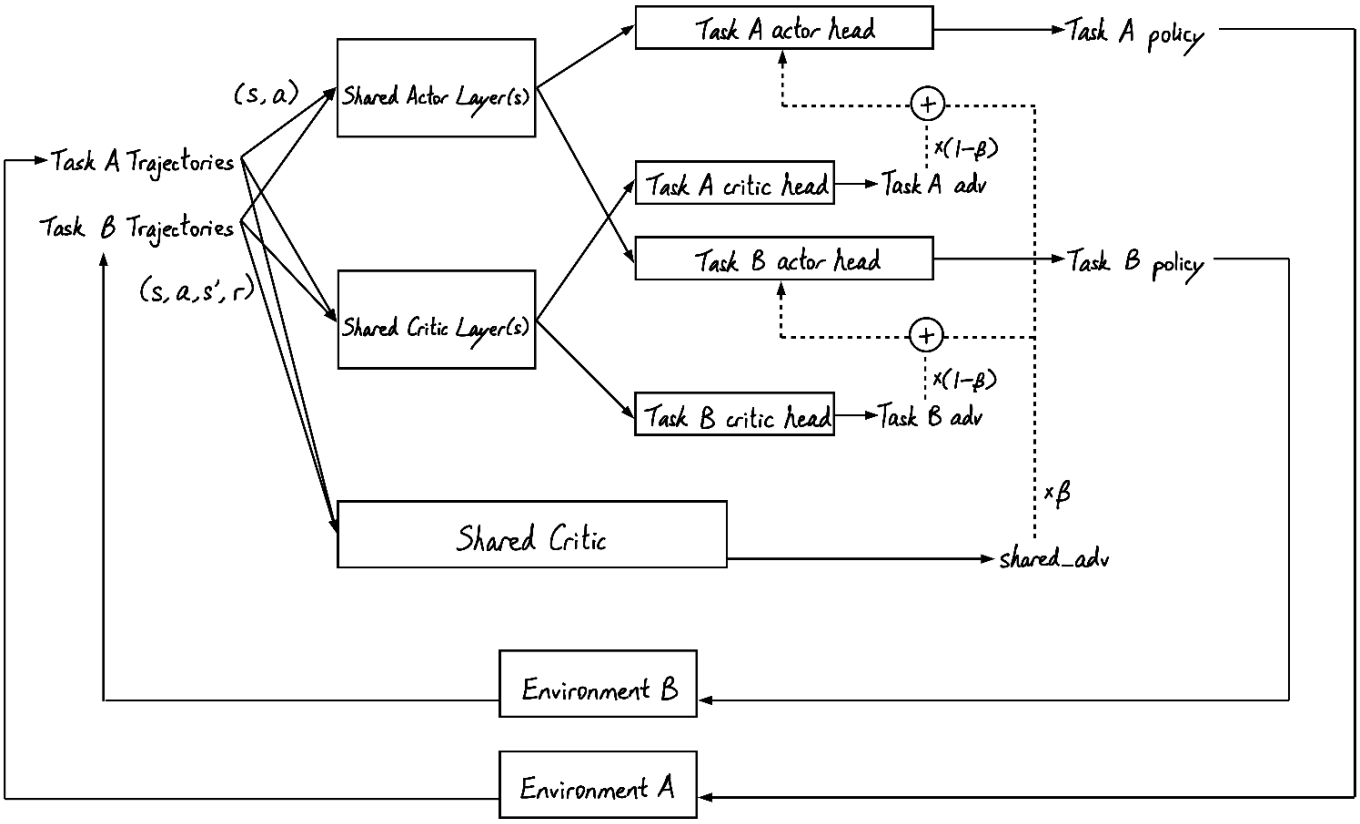



Figure 10: Diagram showing the algorithm of multitask learning with architecture 3.

The advantage is modified to balance between taking advantage of the shared critic and taking advantage of the task specific critic with a shared layer.

Advantage used to update the actor:

$$A^{\pi}(s_{it},\, a_{it}) \,=\, \beta A^{\pi}_{shared}(s_{it},\, a_{it}) \,+\, (1 \,-\, \beta) A^{\pi}_{task}(s_{it},\, a_{it})$$

where $A^{\pi}_{shared}(s_{it},\, a_{it})$ is obtained from the shared critic network and $A^{\pi}_{task}(s_{it},\, a_{it})$ is obtained from the task specific critic with one shared layer.

New parameters:

| | |
|---|---|
| beta | A ratio between 0 and 1 indicating the balance between learning from shared critic and learning from task specific critic |

## Experimental Evaluation

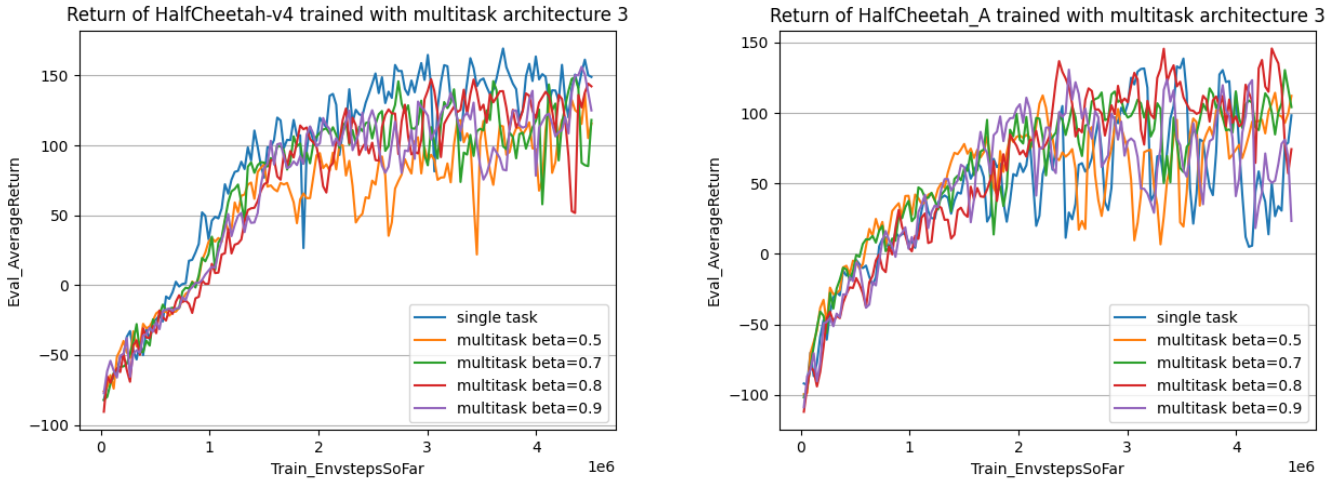



Figure 11: Graphs showing the average return of HalfCheetah-v4 and HalfCheetah_A with multitask learning architecture 3 under a parameter sweep for beta, the ratio between shared critic advantage and task specific critic advantage. Architecture 3 with beta value of 0.8 performs the best when learning tasks HalfCheetah-v4 and HalfCheetah_A.

## Multitask Learning Architecture 4

## Model Architecture Description

Architecture 4 utilizes a shared critic, task specific critic, and task specific actor.

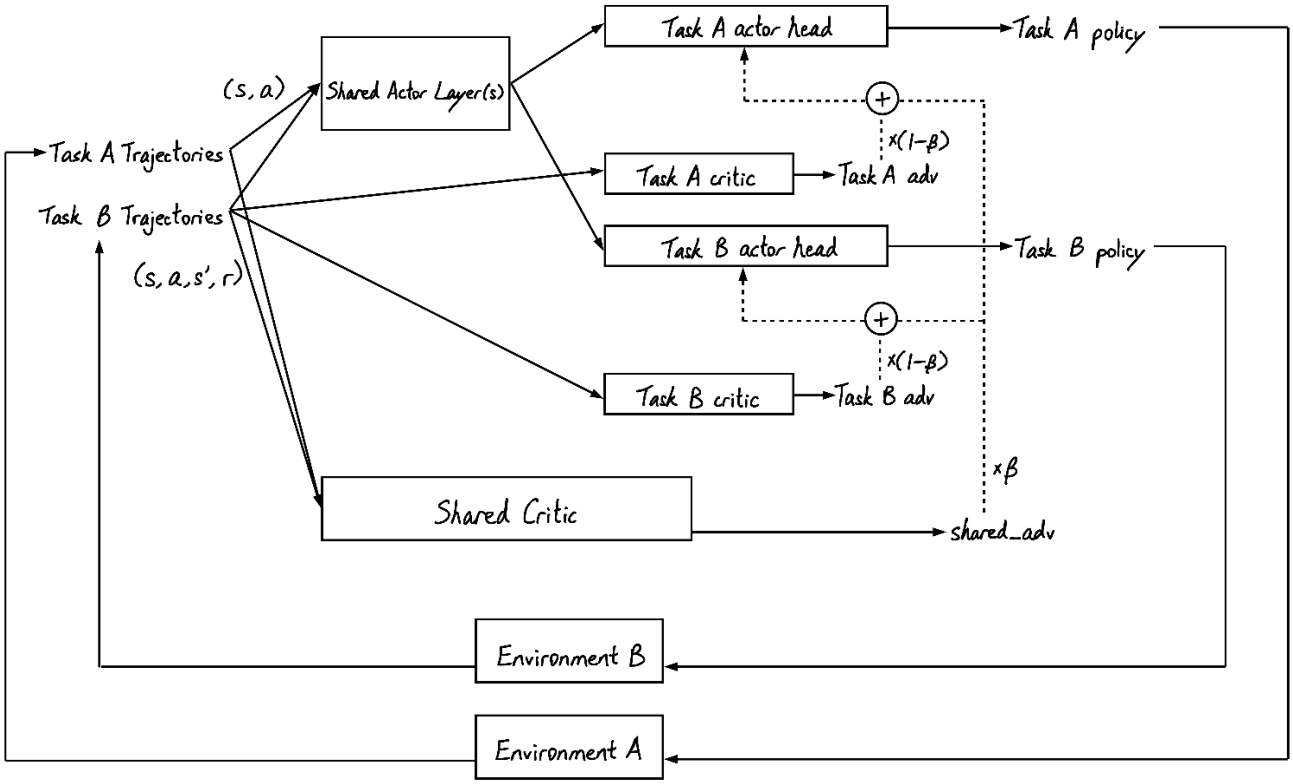



Figure 12: Diagram showing the algorithm of multitask learning with architecture 4.

Similar to Architecture 3, Architecture 4 also uses the modified advantage function:

$$A^{\pi}(s_{it},\, a_{it}) \;=\; \beta A^{\pi}_{shared}(s_{it},\, a_{it}) \;+\; (1 \;-\; \beta) A^{\pi}_{task}(s_{it},\, a_{it})$$

However, the task specific critic no longer has a shared layer as in the previous architectures.

## Experimental Evaluation

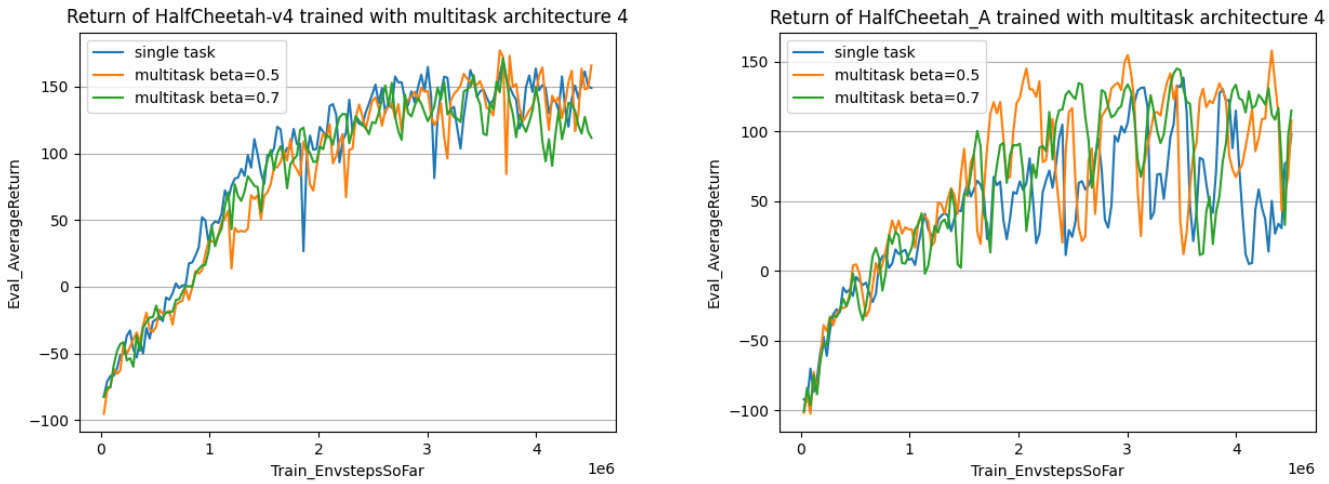



Figure 13: Graphs showing the average return of HalfCheetah-v4 and HalfCheetah_A using multitask learning architecture 4 and different beta values.

# Multitask Learning for Lifelong Learning

The best performance model from the previous section, Multitask Off-Policy Actor-Critic Reinforcement Learning, which is architecture 3 with beta value of 0.8, is adapted to lifelong learning. In the following implementation of continua learning, HalfCheetah-v4 is learned prior to encountering and learning HalfCheetah-A.

## Continual Learning with Multitask Actor-Critic Algorithm

The multitask off-policy actor critic algorithm is performed to learn HalfCheetah-v4. After n_iter_tasks iterations, the learning model encounters a new task, HalfCheetah_A. Joint training is then deployed, where the model is trained with both previous task data and new task data.

## Experimental Evaluation

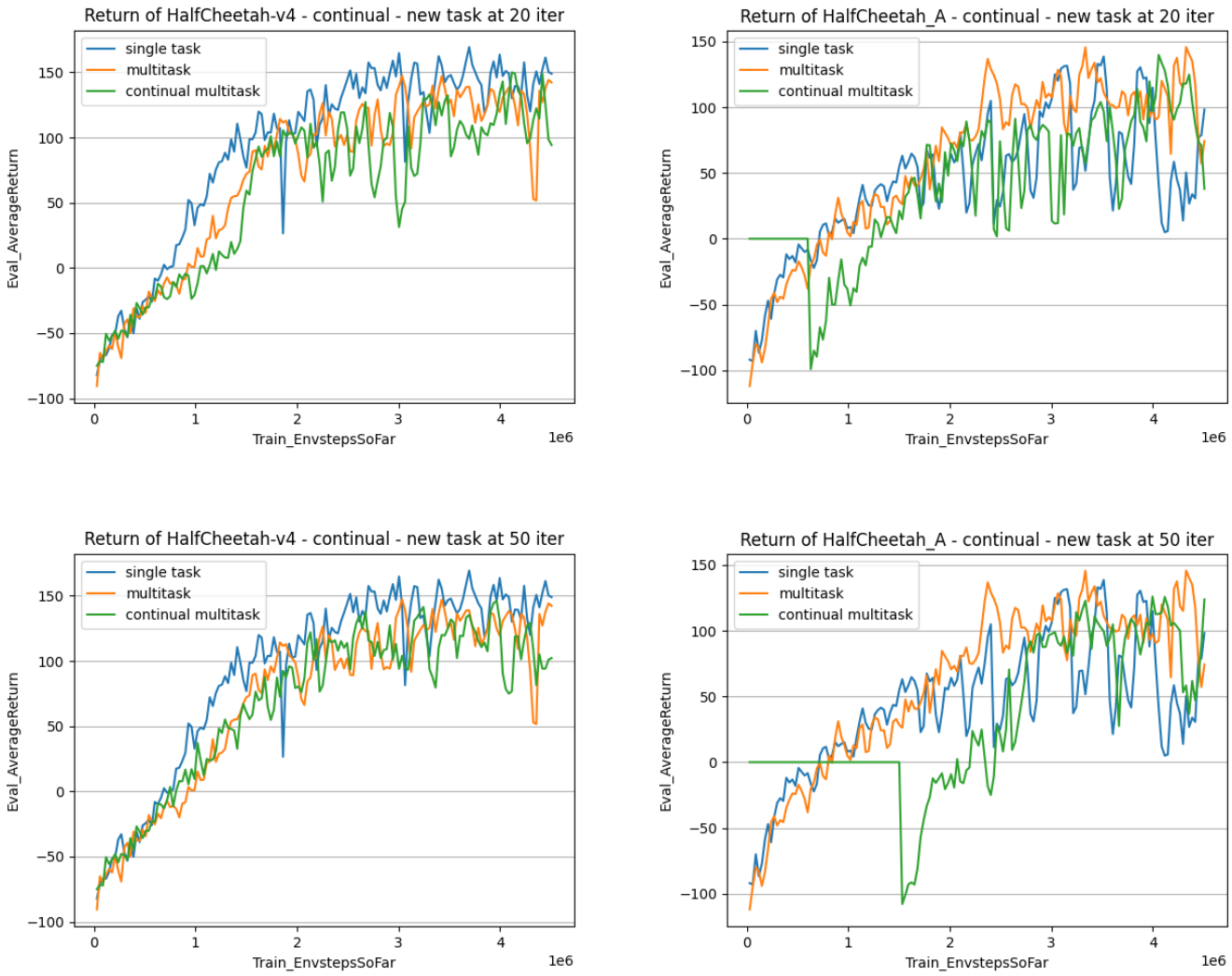



Figure 14: Plots showing the average return of HalfCheetah-v4 and HalfCheetah_A for continual learning: HalfCheetah-v4 is learned before encountering and learning HalfCheetah_A. The top two plots show the return of both environments when HalfCheetah_A comes into play at the 20th iteration, which is why the return of HalfCheetah_A from 0th iteration to 20th iteration is 0. The bottom two plots show the return of both environments when HalfCheetah_A comes into play at the 50th iteration, so its return is 0 from 0th iteration to 50th iteration in continual multitask model.

Catastrophic Forgetting: Neural networks experience distributional shift and catastrophic forgetting, also known as catastrophic inference, is one of the problems relevant to distributional shift. Catastrophic forgetting is the tendency of neural networks to lose information about previously learned tasks while learning new tasks, which is a phenomenon of stability-plasticity dilemma. A model of high stability will not be able to effectively learn new information from future training data, whereas a model of high plasticity suffers from large changes during parameter update and losing previously learned representations. One cause of catastrophic forgetting is parameter shift, when the gradient descent updates the neural networks, parameters drift to a region where the error of the previously learned tasks is high. Activation shift is the direct consequence of parameter shift. To mitigate catastrophic forgetting, this project designs how many parameters are shared and affected when a new task is learned.

## Conclusion

In conclusion, using an off-policy actor-critic algorithm as the foundation, different model architectures and advantage estimation techniques are designed and implemented to achieve multitask learning. Furthermore, the multitask learning algorithm is then being adapted to perform continual learning. Being able to employ multitask learning while maintaining the learned information and open to learning new information, mitigating catastrophic forgetting, could help to approach a solution for continual learning.

## References


Arora, H., Kumar, R., Krone, J., & Li, C. (2018, February 3). *Multi-task learning for continuous control*. arXiv.org. Retrieved December 14, 2022, from https://arxiv.org/abs/1802.01034

*IEEE Xplore Full-text PDF:* (n.d.). Retrieved December 15, 2022, from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7169508

Levine, S. (n.d.). *CS285 Lecture 22 Slides - Meta-Learning & Transfer Learning*. Meta-Learning & Transfer Learning. Retrieved December 14, 2022, from https://rail.eecs.berkeley.edu/deeprlcourse/

Levine, S. (n.d.). *CS 285 Assignment 3: Q-Learning and Actor-Critic Algorithms*.

Li, A., & Abbeel, P. (2020, May 29). *Algorithms for Multi-task reinforcement learning*. Algorithms for Multi-task Reinforcement Learning. Retrieved December 15, 2022, from https://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-110.pdf

Li, F.-F., Krishna, R., & Xu, D. (2020, June 4). *Reinforcement learning lecture 17 - Stanford University*. Lecture 17: Reinforcement Learning. Retrieved December 15, 2022, from http://cs231n.stanford.edu/slides/2020/lecture_17.pdf

Pmlr. (n.d.). *Proceedings of the 13th Asian conference on machine learning held in virtual on 17-19 November 2021 published as volume 157 by the Proceedings of Machine Learning Research on 28 November 2021. volume edited by: Vineeth N balasubramanian ivor tsang series editors: Neil D. Lawrence*. Proceedings of Machine Learning Research. Retrieved December 14, 2022, from https://proceedings.mlr.press/v157/