CMSC473/673 Final Report: DRL Transfer Gym

Charles Meehan **Prakhar Mishra** **Joshua Smith** **Chenqi Matthew Zhu**

Abstract

Deep reinforcement learning has grown increasingly useful for teaching agents to complete even difficult continuous robotic manipulation tasks. However, many of these reinforcement learning algorithms are not sample efficient due to requiring to learn a new task with no prior knowledge of how to act in the environment in order to solve the task at hand. However just like humans, reinforcement learning agents should be able to apply previous knowledge to new tasks. Transfer learning methods have been used to take knowledge learned from one task and apply it to learn a similar but different task. There has been many works in transfer learning applied to reinforcement learning agents, but there has not been a place to go in order to compare these various methods against one another, especially for continuous environments. Therefore, we created a DRL Transfer Gym that provides users with the ability to run transfer methods that we have included or run their own transfer algorithms on a robotic continuous manipulation environment. Further, a transfer learning metric plotting tool was created in order to provide a standard way of evaluating these methods against one another and against tasks learned with using transfer learning.

Introduction

A reinforcement learning agent attempts to maximize its performance by making decisions of what action it should take given feedback from the environment. There has been tremendous progress in the field of reinforcement learning (RL) recently due to the development and use of deep neural networks as function approximators. This combination of deep neural networks with reinforcement learning is called deep reinforcement learning (Zhu et al. 2020b). However even deep reinforcement learning methods struggle with partial observability, sparse feedback, and the high complexity of state and action spaces (Zhu et al. 2020b). One way to reduce the number of interactions with the environment is to provide the RL agent with some domain knowledge. One way to provide domain knowledge is through transfer learning which uses external expertise from other domains to benefit the learning progress of the target task (Zhu et al. 2020b).

Transfer learning methods for RL have been surveyed in (Taylor and Stone 2009; Lazaric 2012), but (Zhu et al. 2020b) focused on recent advances with transfer learning for deep reinforcement learning. As seen in Figure 1, Zhu et al. show many ways to transfer learned knowledge to a target domain; however, a suite of software tools where different transfer methods can be compared against one another and evaluated against a common set of transfer learning metrics is currently missing in this field. Our main objective with this project is to create a gym, named DRL transfer gym, where transfer learning methods for deep reinforcement learning can be evaluated similar to the other established reinforcement learning gyms and zoos such as OpenAI gym, PettingZoo (multi-agent gym), Stable Baselines3, RL Baselines3 Zoo, etc. (Brockman et al. 2016a; Terry et al. 2020; Raffin et al. 2021; Raffin 2020). Our hope with this tool is that users can focus on building better transfer learning algorithms and be able to clearly demonstrate the advantages of using their latest transfer method.

Literature Review

Transfer Learning in Deep Reinforcement Learning

The traditional approach to solving reinforcement learning problems faces challenges like scalability and is limited to low-dimensional problems due to the constraints on memory and computational complexity. Deep reinforcement learning combines the advantages of a deep neural network and the capability of function generalization with reinforcement learning, and upscales RL to high-dimensional problems that were previously deemed intractable (Arulkumaran et al. 2017). Transfer learning further improves efficiency by transferring models learned in one domain to fit problems in a related domain. It is extremely useful in situations where training data is difficult to obtain or resources for training a model are constrained (Weiss, Khoshgoftaar, and Wang 2016). Following the survey by (Zhu et al. 2020b) in the next subsections, we will discuss the latest transfer learning techniques used in deep reinforcement learning.

Reward Shaping Reward shaping is a useful technique that incorporates exterior knowledge into constructing the reward. It can be conveniently applied to RL by adding a reward shaping function to the native reward function with little additional modification to the learning architecture (Laud 2004). (Ng, Harada, and Russell 1999) proposed the idea of Potential-Based Reward Shaping (PBRS) where the shaping function $F : S \times A \times S \rightarrow \mathbb{R}$ is defined to be the difference

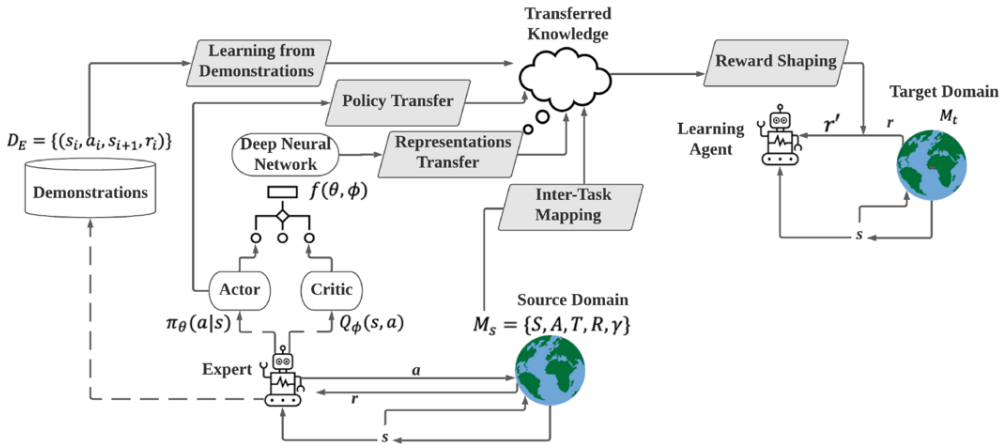

Figure 1: Various transfer learning methods, organized by the format of transferred knowledge from (Zhu et al. 2020b).

between two potential functions $\Phi : S \rightarrow \mathbb{R}$. The PBRS function guarantees consistency with optimal policy. (Devlin and Kudenko 2012) extended this idea of potential function from static to dynamic and proposed a Dynamic Potential function which means the shaping function is now dependent on both state and time. Building on that, (Harutyunyan et al. 2015) developed Dynamic Value Function Advice to include prior knowledge into the reward shaping function which enabled the transfer of an expert policy to be shaped as auxiliary rewards.

Learning From Demonstrations Imitation learning is a machine learning technique in which the agent learns to mimic expert behaviors from demonstrations. This technique is helpful in training humanoid robots to achieve generic human-level tasks such as self-driving vehicles and human-computer interaction. It is gaining popularity as the development of high-resolution sensors and high computational power processors enabled time-efficient mapping from sampled visual data to actions (Hussein et al. 2017). Such a task can be combined with reinforcement learning inspired by biological behaviors, e.g., humans learning how to ride a bicycle through teacher demonstration and increasing their skill through trial and error (Kober and Peters 2010). The reinforcement learning problem involves an agent with a policy which is a mapping from the state in which the agent resides to the action which the agent would execute. The agent would not know the effects of its action until after its execution which would generate a reward for the agent. In reinforcement learning, the agent maximizes its reward by finding an optimal policy played out over an extended period of time (Sutton, Barto, and others 1999). Bahl, Gupta, and Pathak trained a robot to complete house chores ranging from opening drawers to folding cloth using imitation learning based reinforcement learning. They proposed a framework of extracting a prior from human demonstration and improved such a prior through interaction with the environment with the aim to improve sample efficiency in imitation and reinforcement learning.

Policy Transfer Policy transfer adapts pretrained policies from single or multiple sources to a new task (Zhu et al. 2020b). Knowledge distillation is a well-known technique in supervised learning where a cumbersome model was first trained on a sizable dataset and a transfer set for training a small model is constructed as a probabilistic distribution over the targets from the cumbersome model (Hinton et al. 2015). This approach can be modeled in RL as Policy Distillation, where a student tries to minimize the divergence between its policy and the teacher's policy (Rusu et al. 2015). Implementations of this method include teacher distillation where multiple teacher policies are trained to be transferred to a single student policy. While student distillation takes the expectation during optimization over the trajectory of the student policy instead of the teacher's policy. Policy reuse is another policy transfer technique that directly develops the target policy from a saved pretrained policy (Zhu et al. 2020b). One of the initial policy reuse algorithms was Probabilistic Policy Reuse where the target agent during training would select actions from either a source trained policy with some diminishing probability, a epsilon-greedy based selection, or just a random action selection (Fernández and Veloso 2006). General Policy Improvement resolves the scalability issues existing in the previous policy reuse algorithm by showing that the optimal Q-value can be achieved through greedily choosing the action that results in the highest Q-value at each state (Barreto et al. 2016).

Representation Transfer Representation Transfer is similar to Policy Transfer but the feature representations from a pretrained RL model are transferred to target tasks instead of policy. This is achieved under the assumption that the state space and action space of the tasks can be broken down into orthogonal subspaces (Zhu et al. 2020b). One method is Progressive Neural Network which carries out representation transfer through reusing representation and learns lat-

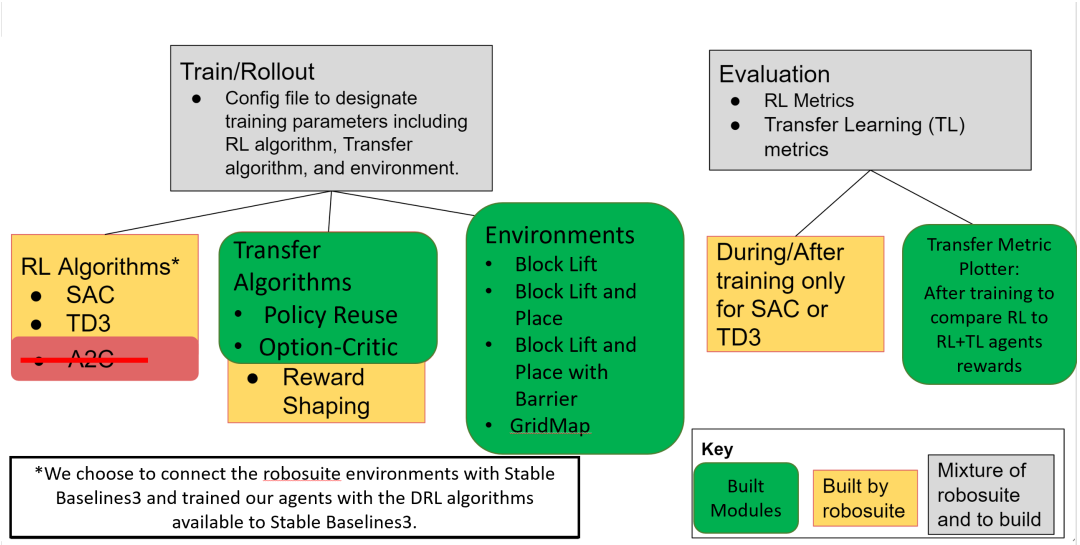

Figure 2: Completed or changed DRL Transfer Gym software modules.

eral connections between multiple tasks during training to extract useful features that would be extended to new tasks (Rusu et al. 2016). While it is possible to retain prior knowledge in this fashion, it comes at the cost of an enormous network. Both the pathway approach from (Harvey 2009) and the modular network approach from (Devin et al. 2017) attempt to solve this issue. The former fixed the size of the network which contains subsets of neurons with prior knowledge while the latter decomposes the network into task-specific and agent-specific modules, allowing the task-specific module to be applied to all the agents hence reducing duplicity in the network.

Inter-Task Mapping Inter-Task Mapping applies mapping functions from the source tasks domain M_s to the target tasks domain M_t. It assumes that there exists a one-to-one function between M_s and M_t. Earlier work from (Taylor, Stone, and Liu 2007) uses a given mapping function to enable fast transfer learning between two tasks with different action space and state space assuming that there are unique correspondence between the action and state spaces from the source domain and those from the target domain. While newer works attempts to learn a mapping function over the state space between the source and target domain (Gupta et al. 2017).

Option Learning Framework The Option Learning Framework was introduced in (Sutton, Precup, and Singh 1999). This framework introduced options into the action space which are temporally abstracted subtasks, which make a standard mdp a semi-mdp. Options can reduce the total amount of action decisions made in an episode, and therefore simplify credit assignment and accelerate learning (Sutton, Precup, and Singh 1999). One extension of option-learning to a standard RL algorithm is Option-Critic (Bacon, Harb, and Precup 2017) which extends standard advantage actor-critic (Mnih et al. 2016) with simultaneously learned options from experience on the environment. Specifically, (Bacon, Harb, and Precup 2017) showed that in a changing goal destination maze navigation task, the Option-Critic Algorithm was able to learn to adapt to changes in the goal destination faster than standard RL algorithms. Some of the learned options from one task might still be useful on a transfer task, and in this case those options do not need to be learned from scratch in the transfer task. Also, the reduction of credit assignment complexity by the Option Learning Framework might improve transfer learning by various metrics.

Project Goals

Our main goal is to provide a Python library where it is possible to conduct transfer learning in deep reinforcement learning experiments and evaluate those experiments against a set of standardized transfer learning metrics. Other goals of our DRL transfer gym project are:

- Provide a simple interface where users can run transfer learning algorithms for deep reinforcement learning trained agents.
- Provide at least three different transfer learning algorithms in our library.
- Evaluate the provided transfer learning algorithms against a set of standard transfer learning metrics.
- Provide a set of tools that will automate the evaluation process.
- Provide a set of instructions for running the provided transfer learning algorithms.
- Provide instructions on how to customize or add a customized learning environment.
- Demonstrate the usefulness of our evaluation suite with experiments upon a modified Option-Critic Algorithm.

Approach

In order to accomplish our project goals within this semester, we built off of the robosuite software platform (Zhu et al. 2020a) which provided a modular set of continuous space manipulation tasks that we used to train our reinforcement learning agents and apply our chosen transfer learning methods. The software modules that make up our transfer DRL gym are a mixture from robosuite and software built by the authors. The software modules that we built for this project and items that were changed are shown in Figure 2. A list of the software that our group will build to meet the goals of our project is below along with a discussion of work that we did not complete or completed differently.

Software Built by Our Group

- Implemented the probabilistic policy reuse algorithm (Fernández and Veloso 2006) which meant writing code to sample actions from a past learned policy with a decaying probability and sample actions from the current new policy following the epsilon-greedy method.
- Created a block lifting and placing environment where the reward function is similar to the block lifting reward function plus added rewards for placing the block on a specified target.
- Created a modified block lifting and placing environment with a barrier on the table to prevent the robot from simply pushing the block across the table to the target.
- Created a plotting tool that given log data from an agent trained without using transfer learning and an agent trained with using transfer learning will plot the average rewards metric from both agents in the same plot. Additional features to this tool are generating the transfer metrics on top of the two average rewards lines similar to lines seen in Figure 5. The transfer learning metrics are also saved to a csv file. Created unit tests for main functions of this tool.
- Created a separate training script that uses the GridMap environment.
- Created a testing script, which depending on command line args runs a training session for a specific environment and transfer method, and will log evaluation metrics/plots to tensorboard. Created unit tests for this functionality.
- Implemented a clean method to transfer a trained agent from one environment to the next while maintaining learned weights from previous task. This will depend on transfer algorithm being used.
- Designed a Transfer learning environment where goal point in the maze will change. Specifically, we created a continuous implementation of the FourRooms environment (Sutton, Precup, and Singh 1999). We designed the environment from scratch in numpy, implementing continuous alterations of the standard FourRooms reward, initialization, transfer, and termination functions.
- Modified an existing implementation of soft actor-critic (Haarnoja et al. 2018a) to be compatible with the agent interface in our evaluation suite because it was not implemented in robosuite or stablebaselines3.

Software and Direction Changed from Proposal

To utilize the existing implementations of RL algorithms as well as the capability to track training and evaluation simultaneously through Tensorboard (Abadi et al. 2016), we chose StableBaselines3 to implement our algorithms. As OpenAI Gym (Brockman et al. 2016b) environments are accepted by algorithms from StableBaselines3 and Robosuite provides wrapper functionality that converts its environment to that of a Gym environment, our environments can work with all algorithms implemented by StableBaselines3, but not without modifications to the original wrapper provided by Robosuite. We have to override its observation function to unify the returned value as float32 for compatibility issues. With the support of StableBaselines3, it became redundant for us to implement A2C which is already supported and was subsequently removed from our architecture.

The block lift environment that was created by robosuite was our first environment that we used to train our agents to pick up a block from the table and lift the block a certain height from the table. When we initially ran this environment with the parameters that robosuite used, our agent learned to pick up the block by resting its wrist on the table and twisting its end effector in the air as shown in Figure 3. We wanted to make sure that the agent learned to properly pick up the block without contacting the table so at first we created an environment with a termination condition and penalty based on if the gripper touched the table. This worked while using RLkit (a RL training library used by robosuite) to train the agent but did not work properly when using Stable Baselines3. Through numerous experiments, we discovered that a larger replay buffer helps the agent with faster training as it allows for more diverse state-action pairs to be sampled when updating the policy. Therefore, we increased the replay buffer for SAC and retrained the agent on the regular block lifting environment without termination conditions. Now, the agent was able to learn to properly pick up the can without touching the table.

We created more robotic manipulation environments than we had originally planned due to emergent behaviors that we needed to restrict. This occurred in training a RL agent from scratch (without transfer learning) in the block lift and place environment. For this environment, the agent was supposed to pick up a block placed on the table and place it on a green target that was also placed on the table. However, the agent learned an optimal policy where it simply pushed the block across the table to the green target. Therefore, we created a very similar environment (Block Lift and Place with Barrier) but with a barrier between the block and the green target in order to prevent the robot from simply pushing the block to the target.

Initially, we planned to be able to run the transfer learning metric plotting tool both during and after training, but it actually made more sense to have the plotting tool pull data from agents that have finished training. This gives us the ability to compare the finished rewards plot of an agent

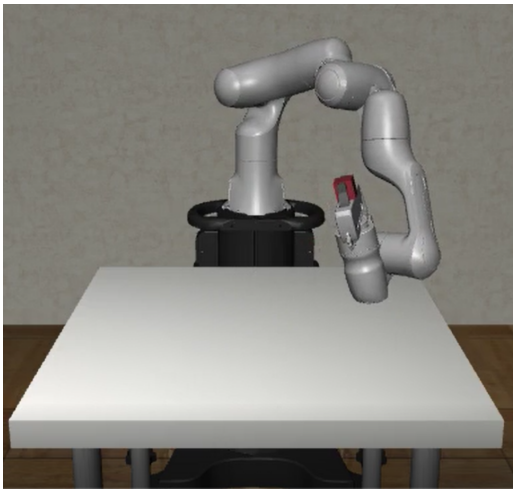

Figure 3: Robot agent cheating block lifting task by using arm on table to hold up block

with no transfer learning versus an agent trained with transfer learning.

For our Option-Critic implementation, we had to significantly modify our original plans. The Option-Critic implementation we were originally going to use did not support continuous action spaces. Instead, we took an existing SAC implementation (Huang et al. 2022), and later built the Option-Architecture around it. We used Soft Actor Critic as the base algorithm because it generally performs well on continuous environments. This was much more time-consuming than we had expected, so instead of implementing a mutual-information latent state, we demonstrated the usefulness of our evaluation suite with our Soft Option Critic algorithm.

Algorithm Information and Implementation

As mentioned in our project goals and approach, the main objective of our work is to be able to provide a software suite that can enable users to train DRL agents with or without transfer learning methods and be able to compare the results of trained agents without transfer learning against trained agents that used transfer learning. For transfer learning, there is typically an exchange of learned knowledge from a source task to an agent learning a new target task with the aim that the agent will be able to learn the target task faster and achieve a higher performance. As we stated in our goals, we have provided three different transfer learning methods as part of our DRL Transfer Gym. The transfer learning algorithms that we chose to replicate are discussed next.

Reward Shaping

The reinforcement learning algorithm used for our evaluation of reward shaping is Soft Actor-Critic (Haarnoja et al. 2018a) which was used by robosuite for their benchmark tests. The transfer method that we use for reward shaping is the simple case of static potential based reward shaping as introduced by (Ng, Harada, and Russell 1999). Learning a task in a continuous environment for a long horizon given sparse rewards (reward only given at the end of an episode) is very difficult, so shaping is used to provide the agents with rewards while taking actions during an episode. This provides more guidance to the agents of beneficial actions to take at each time step of an episode. This reward shaping is constructed by an expert which can be thought of as an expert transferring their knowledge of beneficial actions for the agent to take to successfully complete the task.

π-reuse $(\Pi_{past}, K, H, \psi, v)$.
Initialize $Q^{\Pi_{new}}(s,a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$
For $k = 0$ to $K-1$
  Set the initial state, s, randomly.
  Set $\psi_1 \leftarrow \psi$
  for $h = 1$ to H
    With a probability of ψ_h, $a = \Pi_{past}(s)$
    With a probability of $1 - \psi_h$, $a = \epsilon$-greedy$(\Pi_{new}(s))$
    Receive the next state s', and reward, $r_{k,h}$
    Update $Q^{\Pi_{new}}(s,a)$, and therefore, Π_{new}:
      $Q^{\Pi_{new}}(s,a) \leftarrow (1-\alpha)Q(s,a)^{\Pi_{new}} +$
        $\alpha[r + \gamma \max_{a'} Q^{\Pi_{new}}(s',a')]$
    Set $\psi_{h+1} \leftarrow \psi_h v$
    Set $s \leftarrow s'$
$W = \frac{1}{K}\sum_{k=0}^{K}\sum_{h=0}^{H}\gamma^h r_{k,h}$
Return W, $Q^{\Pi_{new}}(s,a)$ and Π_{new}

Figure 4: Probabilistic Policy Reuse algorithm (Fernández and Veloso 2006)

Policy Reuse

We use the probabilistic policy reuse method as our policy transfer learning method. This method used tabular Q-learning as its reinforcement learning method (Fernández and Veloso 2006). However, since we are interested in deep reinforcement learning, we train our policies using Soft Actor-Critic (Haarnoja et al. 2018b) and Twin Delayed DDPG (Fujimoto, van Hoof, and Meger 2018). We implemented policy reuse by following the pseudo code provided in the original paper as seen in Figure 4. Where with a decaying probability, the agent will pull actions from a past policy learned from the source task. With a probability of 1 minus the decaying probability, we had the agent select actions following typical Soft Actor-Critic algorithm which is similar to the original algorithm of following an epsilon greedy method.

Option-Critic

We use the Option-Critic Architecture (Bacon, Harb, and Precup 2017) for the demonstration of our evaluation suite. This method extends traditional actor-critic methods with option abstractions where each option has individually learned policies, value functions, and termination functions. We compare it to SAC (Mnih et al. 2016), a non-option algorithm which is otherwise identical(Haarnoja et al. 2018a)(Huang et al. 2022). Experiments in (Bacon, Harb, and Precup 2017) suggest that the Option-Critic Architecture may improve transfer learning sample efficiency, but this was only supported with a single transfer task experiment with a single metric. Our project should allow a more

rigorous evaluation of transfer learning improvement between algorithms, and therefore, a stronger claim of whether the Option Architecture improves Transfer Learning over non-Option methods.

Analysis of Our Results

We will first discuss the software and hardware platforms used to train and run our DRL Transfer Gym. Then, we will review the transfer learning metrics that we used for evaluating the transfer methods that we deployed. Lastly, we discuss the results of agents learning certain robotic tasks from scratch versus learning the same task using a transfer learning method.

Hardware and Software Platform

Our DRL transfer gym library was written in Python which matches other standard reinforcement learning gyms. As discussed, we used robosuite as our main platform to build continuous robotic manipulation environments and train reinforcement learning agents. These environments were also used for transfer learning as well. A 2D maze environment was initially used to test Option-Critic methods which then moved to the robosuite environments that were used by the other transfer methods.

Robosuite is a modular software framework for robotic learning in manipulation tasks which has a suite of environments for one arm and dual arm manipulation tasks (Zhu et al. 2020a). The manipulation tasks are simulated in MuJoCo which is a popular physics engine especially when conducting robotic reinforcement learning tasks (Todorov, Erez, and Tassa 2012). The robosuite software has been tested and run on Ubuntu 18.04, 20.04 and with WSL versions 1. We implemented Probabilistic Policy Reuse by extending the OffPolicyAlgorithm class existing in the framework of StableBaselines3. This polymorphic design enables compatibility with other Reinforcement Learning algorithms implemented by StableBaselines3, like Soft Actor-Critic (Haarnoja et al. 2018b) and Twin Delayed DDPG (Fujimoto, van Hoof, and Meger 2018), on which we conducted our experiments mostly. Theoretically, any other Off Policy Reinforcement Learning Algorithm that is StableBaselines3-compatible can extend our OffPolicyAlgorithmPolicyReuse class though it is not tested. For training, we took advantage of the UMI-ACS clusters that we have access to and performed various experiments. On average, it took 2 days to complete 2 million timesteps of training on Nexus clusters with Nvidia A4000 GPU. However, we noticed that this speed is bound by the time spent on obtaining observations from the Robosuite environment as the software runs on CPU only. For our Option-Critic experiments, we trained on a desktop with Ubuntu 20.04 and a Titan RTX GPU. This required 12 hours to train for 1 million timesteps on the robosuite environments with Soft-Option Critic. Similarly to the experiments we ran on the Nexus Clusters, we found that gathering observations on the CPU was the main bottleneck for training speed.

Transfer Learning Evaluation Metrics

The transfer learning evaluation metrics that we used in our transfer learning metric plotting tool are borrowed from (Zhu et al. 2020b; Taylor, Stone, and Liu 2007). The transfer metrics are:

- **Jumpstart Performance (jp)**: the initial performance (returns) of the agent as shown in Figure 5.
- **Asymptotic Performance (ap)**: the ultimate performance (returns) of the agent as shown in Figure 5.
- **Total Rewards (tr)**: the area under the learning curve of the agent shown as in Figure 5.
- **Time to Threshold (tt)**: the learning time or iterations needed by the target agent to reach a certain performance threshold as shown in Figure 5.

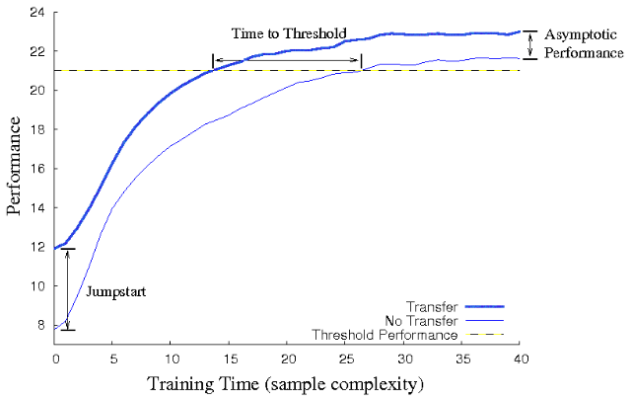

Figure 5: Jumpstart performance, asymptotic performance, time to threshold, and total reward (area under the learning curve) are shown on this example learning curve plot (Taylor and Stone 2009).

Evaluating Reward Shaping Method

Robosuite already has expert domain knowledge built to their environments and have run their benchmark tests including this knowledge in the form of reward shaping. Figure 6 shows a snapshot of the reward shaping for the block lifting task where a sloping function is used to give an increasing reward as the end effector gets closer to the block. There is also a grasping reward for when the gripper fingers make contact with the block. For our comparison, the target task will be the blocking lifting task with reward shaping which we compared against an agent that learned the block lifting task from scratch without using reward shaping. Our transfer metric plot of the performance of these two agents is shown in Figure 7 where the agent that learned the task from scratch is the no transfer method line in blue and the agent using reward shaping is the transfer method line in orange. Also, we show the smoothed data lines on top of their raw data, and the transfer metrics are shown by the red lines. Further, these metrics are printed in the terminal when our transfer metric plotter tool is used as well as output to a csv

```
# use a shaping reward
elif self.reward_shaping:

    # reaching reward
    cube_pos = self.sim.data.body_xpos[self.cube_body_id]
    gripper_site_pos = self.sim.data.site_xpos[self.robots[0].eef_site_id]
    dist = np.linalg.norm(gripper_site_pos - cube_pos)
    reaching_reward = 1 - np.tanh(10.0 * dist)
    reward += reaching_reward

    # grasping reward
    if self._check_grasp(gripper=self.robots[0].gripper, object_geoms=self.cube):
        reward += 0.25
```

Figure 6: Reward shaping used by robosuite for the block lifting task.

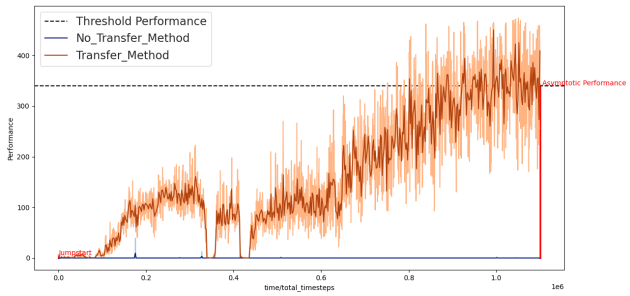

Figure 7: Transfer metric plot for reward shaping.

file. The table of the metrics for reward shaping are shown in Table 1.

As shown in Figure 7 and in Table 1, the transfer method of reward shaping provides a improvement to the learning agent at the beginning of training as seen by the jumpstart performance metric. Also, the transfer method reached the threshold performance, had a positive asymptotic performance metric, and had a higher total rewards metric. Therefore, we can say that using reward shaping improved the learning of the block lifting task, and further, it was necessary to use reward shaping to learn the task since from scratch did not learn a successful policy.

Evaluating Policy Reuse Method

The policy reuse method falls under the transfer learning category of policy transfer as discussed in our literature review. The source task will be the block lifting task provided by robosuite which has goals of grasping a block on a table and lifting it a specified distance from the table. The target task will be a block lifting and placing task which has goals of grasping a block on a table, lifting the block a specified distance off the table, and placing the block back on a certain target on the table as shown in Figure 8. The markov decision process difference between these two tasks will be in the reward since the block lifting and placing task has the added goal of placing the block back on a target on the table. We used the probabilistic policy reuse method (Fernández and Veloso 2006) as discussed in the algorithm implementation section. The transfer metric plot will compare an agent's average rewards from training the lift and place task from scratch versus the agent's average rewards from training the lift and place task using the policy reuse method. The transfer metric plot is shown in Figure 9, and the transfer metric data is included in Table 2.

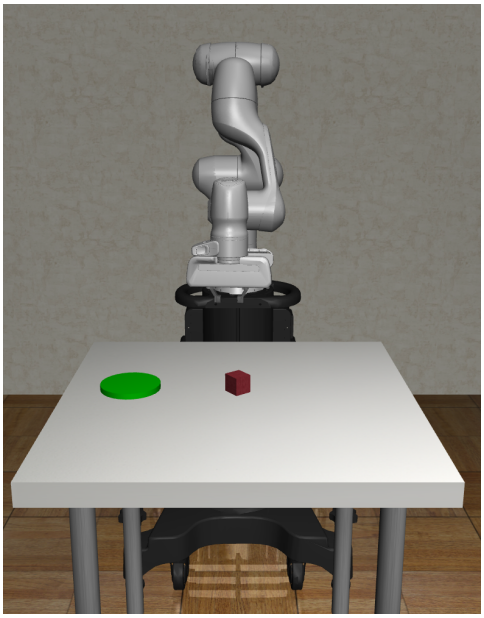

Figure 8: Lift and place task environment.

As seen in Figure 9 and in Table 2, the agent that learned from scratch (no transfer method) in blue actually outperforms the agent trained with policy reuse (transfer method) in orange. Even though both agents achieve performance levels above the threshold, the agent without using the transfer method has a higher ending performance which results in the asymptotic performance metric being negative. One thing to note is the fact that the jumpstart performance metric is still positive so the transfer method still gave a boost to the agent trained with policy reuse method. However, the agent that used no transfer method outperformed the agent with the transfer method because the agent pushed the block until it reached the target which violated the behaviour that we wanted to see. We wanted the arm to lift the block off the table and place it on the green target. The agent that was trained with policy reuse did copy the behaviour that it learned from the lift task by picking up and lifting the block from the table, but it struggled with the end goal of placing the block back on the table. Therefore, it did not receive on average as high of rewards as the agent that just pushed the block to the green target.

In order to address the emergent behaviour of the agent pushing the block across the table to the target, we created an environment with a barrier in between where the block was placed and where the green target was placed on the table as seen in Figure 10. We made the density and friction parameters of the blue barrier high in order to prevent the robot from pushing it over which we saw in training cases before we increased these physical parameters. The updated transfer metrics plot for the agent trained from scratch and an agent using policy reuse is shown in Figure 11 and the transfer metrics data is shown in Table 3.

Condition	Jumpstart Performance	Asymptotic Performance	Time to Threshold (Total Timesteps)	Total Rewards
No Transfer Method	NA	NA	Did Not Reach	347
Transfer Method	NA	NA	751500	1750963.5
One Metric Only	5.93	340.46	NA	NA

Table 1: Transfer metric data for reward shaping

Condition	Jumpstart Performance	Asymptotic Performance	Time to Threshold (Total Timesteps)	Total Rewards
No Transfer Method	NA	NA	370500	1663012.99
Transfer Method	NA	NA	379500	1085344.76
One Metric Only	20.43	-303.54	NA	NA

Table 2: Transfer metric data for policy reuse.

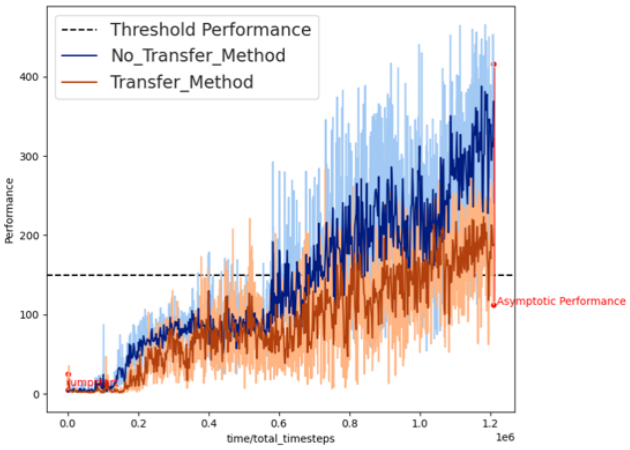

Figure 9: Transfer metric plot for policy reuse.

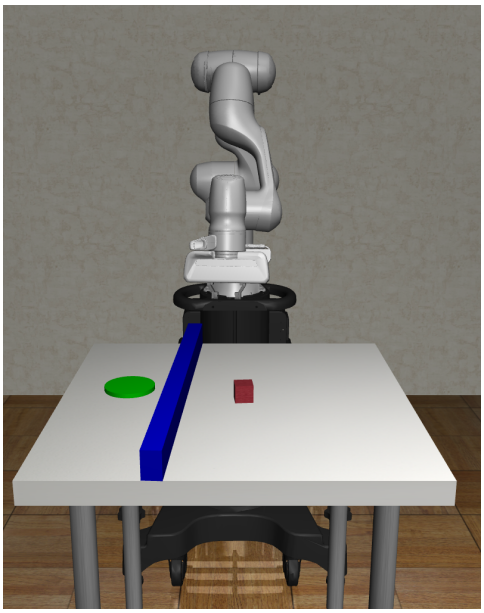

Figure 10: Lift and place task environment with barrier.

As seen in Figure 11 and Table 3, the jumpstart performance and asymptotic performance metrics are both positive. The time to threshold seems faster for the no transfer method but this was because our plotting tool is counting the one data point where the no transfer method in blue passes the performance threshold line. This was an anomaly because the no transfer method did not converge above the performance threshold. The transfer method plot did reach the threshold and shows convergence around the threshold. The total rewards was much bigger for the transfer method than the no transfer method. We did notice that the transfer method did get the block to the green target by picking up the block and then once it was over the green target dropping the block on the target. This was not the behavior that we wanted, but this is something that we would be looking into the future of how to better get the ending behaviour to match placing the block on the green target.

Evaluating Option-Critic Method

In order to demonstrate how our DRL transfer gym is useful for development of novel Transfer Methods and RL algorithms, we tested our Soft Option-Critic implementation on two transfer environment pairs. First, we tested on our continuous FourRooms environment pictured in figure 12. In this environment, the goal position is constant throughout all episodes, but the agent initialization position is randomized every episode. Our source task was learning to navigate the maze given a specified constant position for the goal, and transferring to an environment with a different goal position. We conjecture that after learning to navigate throughout the four rooms in the source task, the transferred policy will be more capable of learning to find a different goal position in the target task.

The specific method of transfer we implemented for all Soft Option-Critic evaluations was model reuse, where the weights of all models in the learned agent on the source task were used as initialization when learning on the target task. Our baseline that we compare this performance to is an agent trained on the target environment with randomly initialized

Condition	Jumpstart Performance	Asymptotic Performance	Time to Threshold (Total Timesteps)	Total Rewards
No Transfer Method	NA	NA	561500	29019
Transfer Method	NA	NA	597000	381878.34
One Metric Only	10.98	145.26	NA	NA

Table 3: Transfer metric data for policy reuse in pick and place environment with barrier.

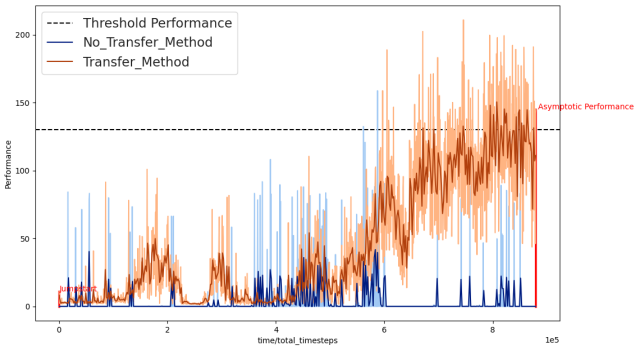

Figure 11: Transfer metric plot for policy reuse method on lift and place environment with barrier.

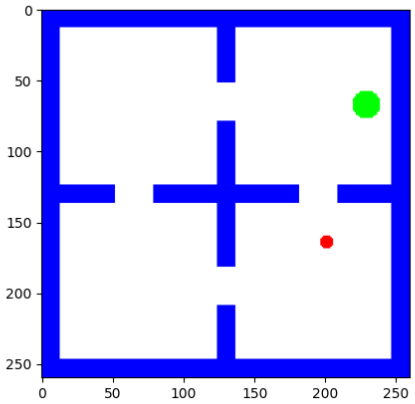

Figure 12: Render of our Continuous FourRooms Environment. Agent in Red, Goal in Green.

weights.

We were specifically interested in investigating the hypothesis that Option implementations should allow better transfer performance when transferring using the model reuse method. In our tests, we compare Soft Actor-Critic with Soft Option-Critic. Specifically, in our implementation, when training with one option, Soft Option-Critic reduces to vanilla Soft Actor-Critic (Haarnoja et al. 2018a), and this was validated with preliminary experiments. This assures that we can isolate our independent variable, with only a single hyper-parameter change, number of options, between our Soft Actor-Critic and Soft Option-Critic tests.

Interestingly, after testing the transfer performance for both Soft Actor-Critic and Soft Option-Critic, we yielded similar findings to that of Bacon et al. (Sutton, Precup, and Singh 1999). Even after modifying the environment to have a continuous action and observation space as well as modifying the Option algorithm to support continuous action spaces, as can be seen in figures 13 and 14, we found that the Option algorithm yielded significant transfer gains over the non-Option algorithm when implementing model reuse.

We also tested the transfer performance for the Robosuite Block Lift and Lift+Place transfer pair. These results are plotted in Figures 15 and 16 and Tables 4 and 5. Unlike the FourRooms Transfer task, we see no evidence of positive transfer for both the Soft Actor-Critic and Soft Option-Critic (4 Options) algorithms. For both algorithms, negative or negligible transfer was observed, where the policy trained from scratch was no less performant than that which reused weights from the source task. This can be observed not only in the performance plots, but also in the various metrics generated by our DRL transfer gym, where, for example, the No-Transfer Soft Actor-Critic agent accumulated greater total rewards. The negative task transfer was relatively less present in the Soft Option Critic evaluation, but there was still no clear advantage to training on the source task. This suggests that our Soft Option-Critic algorithm's transfer performance, as well as that of Soft Actor-Critic, does not scale well to more complex environments and target tasks. However, it does agree with our maze navigation experiments in that the Option Architecture allowed relatively better transfer performance than the no Option algorithm.

Overall, analysis of our algorithm's transfer efficiency, enabled by our evaluation suite, suggests that the addition of Options does appear to improve empirical transfer performance. However, negative or negligible transfer performance was present on the more complex robotic manipulation task. In future work, our experiments suggest that the common denominator, direct model reuse, might need to be improved or be substituted with a more performant transfer method. Even still, our findings support the hypothesis that Options improve transfer efficiency over non-Option algorithms, validating the Option Framework (Sutton, Precup, and Singh 1999) as a promising method for improving transfer efficiency.

Validation Methods

We built unit tests for the software that our group will be added for this project. The unit tests are located in a tests directory within our code repository. There is a test created to test code units within our transfer metric plotter tool.

Condition	Jumpstart Performance	Asymptotic Performance	Time to Threshold (Total Timesteps)	Total Rewards
No Transfer Method	NA	NA	38999	533984
Transfer Method	NA	NA	24999	283444
One Metric Only	30.79	-171.30014038085938	NA	NA

Table 4: Transfer metric data for Soft Actor Critic Model Reuse.

Condition	Jumpstart Performance	Asymptotic Performance	Time to Threshold (Total Timesteps)	Total Rewards
No Transfer Method	NA	NA	54999	343513
Transfer Method	NA	NA	10499	566573
One Metric Only	16.09	59.19	NA	NA

Table 5: Transfer metric data for Soft Option Critic (4 Options) Model Reuse.

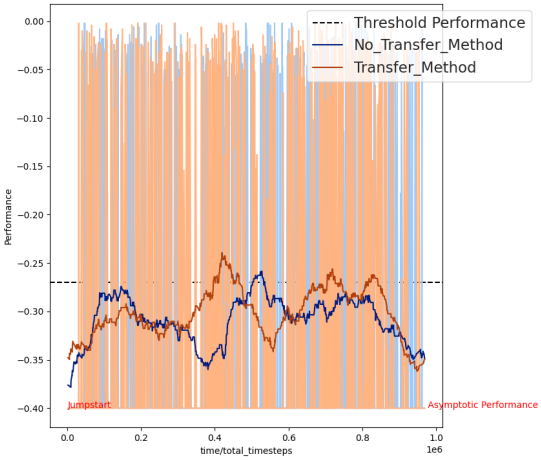

Figure 13: Transfer Plot for Soft Actor Critic.

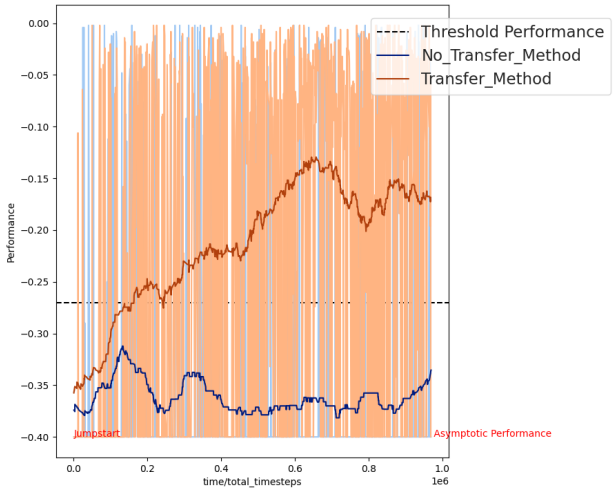

Figure 14: Transfer Plot for Soft Option Critic (4 options).

There were also tests created for our main DRL transfer gym module. Our code currently passes all of the unit tests that we created. Further, our environments that we created pass a unit test created by robosuite to test their environments. There are also demo scripts added to the tests folder that we used in order to ensure that the custom environments that we built operate properly. For instance, we scripted the arm to pick up the block and move it to the green target in the pick and place environment that we created in order to make sure that our rewards were being calculated correctly.

Project Documentation and Distribution

All of our code is hosted in the following GitHub repository, https://github.com/MatthewCQSZ/transfer-drl.git. Our repository is set up to include README files with instructions on how to set up our code and run training, rollout videos, and our transfer learning metric plotting tool. There are also instructions on how to set up your system in order to use the third party software that our code is connected with such as robosuite and Stable Baselines3. Our transfer learning algorithms are also included along with log files from our prior training runs that can be used to compare new methods against. Our custom environments are located in the environments folder and the unit tests are in the tests directory. Lastly, the csv file that is printed when the transfer metric plotter is run is located in the transfer metrics log folder.

Conclusion

The objective of our project was to create a place where users could understand how well their transfer method improved learning on continuous robotic manipulation tasks. This is why we created a Deep Reinforcement Learning Transfer Gym (DRL Transfer Gym), so transfer methods could be evaluated using our transfer metric plotting tool and compared against the transfer learning algorithms that we provided. Besides the changes that we had to make in order for the reward shaping method to work for the block lifting task, we also added the new transfer method of probabilistic policy reuse. Additionally, we added the soft option-critic algorithm. Finally, we showed how our evaluation suite can

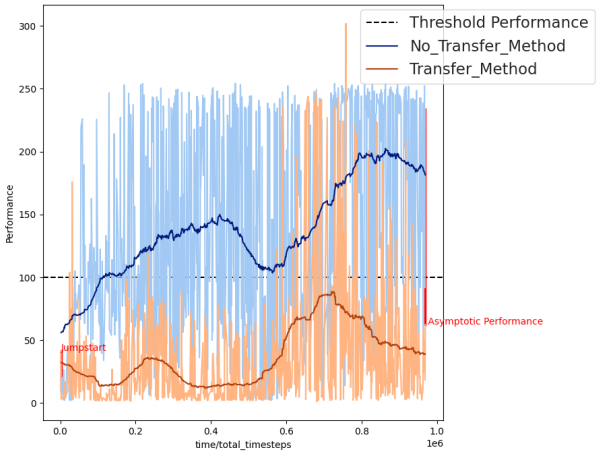

Figure 15: Transfer Plot for Soft Actor Critic on Lift+Place Tranfer Env.

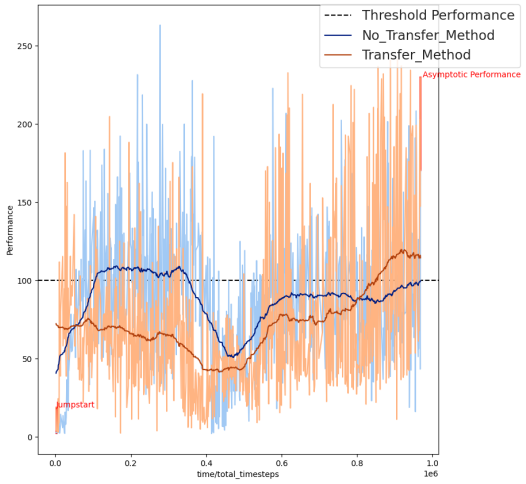

Figure 16: Transfer Plot for Soft Option Critic (4 options) on Lift+Place Transfer Env.

be used to evaluate empirical transfer performance on experimental RL algorithms. While these transfer methods have been applied to discrete grid world tasks, we believe that this was one of the first times augmenting these algorithms to work with continuous robotic manipulation tasks. It was interesting to find out how difficult it was to get the agent to complete the expected behaviour even with using transfer learning. In the future, we would like to investigate better how to take advantage of the jumpstart performance that transfer learning provides and not lose that progress during the remainder of the training task.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mane, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viegas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems.

Arulkumaran, K.; Deisenroth, M. P.; Brundage, M.; and Bharath, A. A. 2017. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*.

Bacon, P.-L.; Harb, J.; and Precup, D. 2017. The option-critic architecture. *Proceedings of the AAAI Conference on Artificial Intelligence* 31(1).

Bahl, S.; Gupta, A.; and Pathak, D. 2022. Human-to-robot imitation in the wild. *RSS*.

Barreto, A.; Dabney, W.; Munos, R.; Hunt, J. J.; Schaul, T.; van Hasselt, H.; and Silver, D. 2016. Successor features for transfer in reinforcement learning.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016a. Openai gym.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016b. Openai gym. *arXiv preprint arXiv:1606.01540*.

Devin, C.; Gupta, A.; Darrell, T.; Abbeel, P.; and Levine, S. 2017. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2169–2176. IEEE Press.

Devlin, S. M., and Kudenko, D. 2012. Dynamic potential-based reward shaping. In *Proceedings of the 11th international conference on autonomous agents and multiagent systems*, 433–440. IFAAMAS.

Fernández, F., and Veloso, M. 2006. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, 720–727. New York, NY, USA: Association for Computing Machinery.

Fujimoto, S.; van Hoof, H.; and Meger, D. 2018. Addressing function approximation error in actor-critic methods.

Gupta, A.; Devin, C.; Liu, Y.; Abbeel, P.; and Levine, S. 2017. Learning invariant feature spaces to transfer skills with reinforcement learning.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018a. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Dy, J., and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 1861–1870. PMLR.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018b. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

Harutyunyan, A.; Devlin, S.; Vrancx, P.; and Nowe, A. 2015. Expressing arbitrary reward functions as potential-based advice. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, 2652–2658. AAAI Press.

Harvey, I. 2009. The microbial genetic algorithm. In *Proceedings of the 10th European Conference on Advances in Artificial Life: Darwin Meets von Neumann - Volume Part II*, ECAL'09, 126–133. Berlin, Heidelberg: Springer-Verlag.

Hinton, G.; Vinyals, O.; Dean, J.; et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* 2(7).

Huang, S.; Dossa, R. F. J.; Ye, C.; Braga, J.; Chakraborty, D.; Mehta, K.; and Araújo, J. G. 2022. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research* 23(274):1–18.

Hussein, A.; Gaber, M. M.; Elyan, E.; and Jayne, C. 2017. Imitation learning: A survey of learning methods. *ACM Comput. Surv.* 50(2).

Kober, J., and Peters, J. 2010. Imitation and reinforcement learning. *IEEE Robotics & Automation Magazine* 17(2):55–62.

Laud, A. D. 2004. *Theory and application of reward shaping in reinforcement learning*. Ph.D. Dissertation, University of Illinois at Urbana-Champaign. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2021-05-26.

Lazaric, A. 2012. *Transfer in Reinforcement Learning: A Framework and a Survey*. Berlin, Heidelberg: Springer Berlin Heidelberg. 143–173.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T. P.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. *CoRR* abs/1602.01783.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, 278–287. Morgan Kaufmann.

Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; and Dormann, N. 2021. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research* 22(268):1–8.

Raffin, A. 2020. Rl baselines3 zoo.

Rusu, A. A.; Colmenarejo, S. G.; Gulcehre, C.; Desjardins, G.; Kirkpatrick, J.; Pascanu, R.; Mnih, V.; Kavukcuoglu, K.; and Hadsell, R. 2015. Policy distillation. *arXiv preprint arXiv:1511.06295*.

Rusu, A. A.; Rabinowitz, N. C.; Desjardins, G.; Soyer, H.; Kirkpatrick, J.; Kavukcuoglu, K.; Pascanu, R.; and Hadsell, R. 2016. Progressive neural networks.

Sutton, R. S.; Barto, A. G.; et al. 1999. Reinforcement learning. *Journal of Cognitive Neuroscience* 11(1):126–134.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.* 112(1–2):181–211.

Taylor, M. E., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.* 10:1633–1685.

Taylor, M. E.; Stone, P.; and Liu, Y. 2007. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research* 8(9).

Terry, J. K.; Black, B.; Grammel, N.; Jayakumar, M.; Hari, A.; Sulivan, R.; Santos, L.; Perez, R.; Horsch, C.; Dieffendahl, C.; Williams, N. L.; Lokesh, Y.; Sullivan, R.; and Ravi, P. 2020. Pettingzoo: Gym for multi-agent reinforcement learning. *arXiv preprint arXiv:2009.14471*.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. IEEE.

Weiss, K.; Khoshgoftaar, T. M.; and Wang, D. 2016. A survey of transfer learning. *Journal of Big data* 3(1):1–40.

Zhu, Y.; Wong, J.; Mandlekar, A.; and Martín-Martín, R. 2020a. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*.

Zhu, Z.; Lin, K.; Jain, A. K.; and Zhou, J. 2020b. Transfer learning in deep reinforcement learning: A survey.