Object Grasping using Hierarchical Reinforcement Learning

Arjun Sharma (arjuns2), Mohit Sharma (mohits1)

1 Introduction

Many complex real world tasks can be represented as a sequential collection of easier sub-tasks. When using reinforcement learning, breaking a task into sub-tasks in this way makes it amenable to solving using hierarchical reinforcement learning. In this work, we look at the task of object grasping using a robot arm. We break the grasping task into two sub-tasks — reaching and gripping. We use two levels in our hierarchical reinforcement learning. Policies at both levels are trained using Generalized Advantage Estimation Schulman et al. [2015b] with Proximal Policy Optimization Schulman et al. [2017]. Unlike previous approaches to hierarchical learning however, we do not keep separate policies at the lower level. This has the two potential benefits. First, this makes the hierarchical approach scalable as the number of sub-tasks grow. Second, it allows knowledge of one sub-task to be shared with another. The higher level policy, called the manager policy, learns to weight the reward functions for each sub-task at any given step. Intuitively, we would expect the contribution from the sub-tasks to smoothly change according to their order in the sequence. Initial experiments with the MuJoCo physics simulator Todorov et al. [2012] shows the potential benefits of our method.

We first cover the preliminaries in the next section. We then describe a recent approach to object grasping which we build upon in this work. Finally, we describe our approach and present our initial results.

2 Preliminaries

We provide a brief overview of Generalized Advantage Estimation, Proximal Policy Optimization and Hierarchical Reinforcement Learning in this section.

2.1 Markov Decision Process

A Markov Decision Process (MDP) is a popular mathematical formulation for modelling decision making. It consists of a tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0)$ with actions $a \in \mathcal{A}$, states $s \in \mathcal{S}$, reward function r, transition dynamics P and policy π. An initial state s_0 is sampled from the initial state distribution ρ_0 and a trajectory $(s_0, a_0, s_1, a_1, \cdots)$ is then sampled using actions sampled according to the policy $a_t \sim \pi(a|s_t)$ and states sampled using the transition dynamics $s_{t+1} \sim P(s|s_t, a_t)$. A reward $r_t = r(s_t, a_t)$ is received at each time step. The value of state $V^\pi(s_t) = \mathbb{E}_{s_{t+1:\infty}, a_{t:\infty}}[\sum_{l=0}^{\infty} r_{t+l}]$ is the expected reward obtained by following policy π from state s_t. The action-value $Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1:\infty}, a_{t+1:\infty}}[\sum_{l=0}^{\infty} r_{t+l}]$ is the expected reward by taking action a_t in state s_t and thereon following policy π.

2.2 Generalized Advantage Estimation

Policy gradient methods Sutton et al. [2000] are a popular class of reinforcement learning algorithms for continuous control. The goal in policy gradient methods is to maximize the total reward $\sum_{t=0}^{\infty} r_t$. Typically, policy gradient methods compute the gradient $g = \nabla_\theta \mathbb{E}[\sum_{t=0}^{\infty} r_t]$ and then use gradient ascent. Several different related expressions for the gradient have been listed in the literature. An expression using the Advantage Function A^π yields the lowest possible variance.

$$g = \mathbb{E}[A^\pi(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)] \tag{1}$$

where $A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$. A step in the policy gradient direction thus, increases the probability of better-than-average actions and reduces probability of taking worse-than-average actions.

The parameter γ is used in the discounted MDP formulation as a discount factor, to down weigh rewards from future time steps. However, it can also be thought of as a parameter to reduce variance in the undiscounted formulation, but at the cost of introducing bias. The new definitions for the value, action-value and advantage functions then becomes

$$V^{\pi,\gamma}(s_t) = \mathbb{E}_{s_{t+1:\infty}, a_{t:\infty}}[\sum_{l=0}^{\infty} \gamma^l r_{t+l}] \tag{2}$$

$$Q^{\pi,\gamma}(s_t, a_t) = \mathbb{E}_{s_{t+1:\infty}, a_{t+1:\infty}}[\sum_{l=0}^{\infty} \gamma^l r_{t+l}] \tag{3}$$

$$A^{\pi,\gamma}(s_t, a_t) = Q^{\pi,\gamma}(s_t, a_t) - V^{\pi,\gamma}(s_t) \tag{4}$$

The discounted approximation to the policy gradient then becomes

$$g^{\gamma} = \mathbb{E}[A^{\pi,\gamma}(s_t, a_t) \nabla_{\theta} \log \pi_{\theta}(a_t|s_t)] \tag{5}$$

Generalized Advantage Estimate Schulman et al. [2015b] is a method to obtain estimator for $A^{\pi,\gamma}$. Given an estimate $\hat{A}_t$ for a batch of N episodes, then policy gradient can then be approximated as

$$\hat{g} = \sum_{n=1}^{N} \sum_{t=0}^{\infty} \hat{A}_t \nabla_{\theta} \log \pi_{\theta}(a_t^n|s_t^n) \tag{6}$$

Next, we discuss how this estimate is calculated. Let V be the approximate value function (for eg. a neural network function approximator). Defite $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$ to be the TD residual. δ_t^V is then an estimate of the advantage function. Consider k step approximations of the advantage function as

$$\hat{A}_t^{(1)} = \delta_t^V = -V(s_t) + r_t + \gamma V(s_{t+1}) \tag{7}$$

$$\hat{A}_t^{(2)} = \delta_t^V + \gamma \delta_{t+1}^V = -V(s_t) + r_t + \gamma r_{t+1} \gamma^2 V(s_{t+2}) \tag{8}$$

$$\hat{A}_t^{(3)} = \delta_t^V + \gamma \delta_{t+1}^V + \gamma^2 \delta_{t+2}^V = -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3}) \tag{9}$$

and so on to get,

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}) \tag{10}$$

$\hat{A}_t^{(k)}$ can be considered as an estimator of the advantage with the bias becoming smaller as $k \to \infty$. The Generalized Advantage Estimate $GAE(\gamma, \lambda)$ is then the exponentially-weighted average of these k-step estimators

$$\begin{aligned}
\hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(1)} + \cdots) \\
&= (1-\lambda)(\delta_t^V + \lambda(\delta_t^V + \gamma \delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma \delta_{t+1}^V + \gamma^2 \delta_{t+2}^V) + \cdots) \\
&= (1-\lambda)(\delta_t^V(1 + \lambda + \lambda^2 + \cdots) + \gamma \delta_{t+1}^V(\lambda + \lambda^2 + \lambda^3 + \cdots) + \cdots) \\
&= (1-\lambda)(\delta_t^V(\frac{1}{1-\lambda}) + \gamma \delta_{t+1}^V(\frac{\lambda}{1-\lambda}) + \gamma^2 \delta_{t+2}^V(\frac{\lambda^2}{1-\lambda}) + \cdots) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V
\end{aligned} \tag{11}$$

2.3 Proximal Policy Optimization

In recent years, several different approaches have been proposed for policy optimization in Reinforcement Learning (Schulman et al. [2015a], Achiam et al. [2017]). One of the main challenges in policy optimization is to find an algorithm that is data efficient and robust across different tasks without a lot of hyperparameter tuning. Recently, Schulman et al. [2017] proposed a robust and simple to implement policy optimization algorithm called Proximal Policy Optimization (PPO).

PPO simplifies the more complex performance objective introduced in Trust Region Policy Optimization (TRPO) (Schulman et al. [2015a]). It uses a first-order approximation of TRPO to simplify the performance objective in TRPO. Specifically, in TRPO we perform the following optimization,

$$\max{}_{\theta} E_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_\theta^{old}(a_t|s_t)} \hat{A}_t \right] \tag{12}$$

$$\text{subject to } E_t \left[KL \left(\pi_\theta^{old}(.|s_t), \pi_\theta(.|s_t) \right) \right] \leq \delta \tag{13}$$

While in PPO we generally use a clipped surrogate objective, making use of the probability ratio, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_\theta^{old}(a_t|s_t)}$ to get

$$L(\theta) = E_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right] \tag{14}$$

Notice that the first term in the minimization in (14) is nothing but the maximization term in the TRPO objective (12). There are other alternate forms of PPO described in Schulman et al. [2017], however, the form discussed above is most commonly used.

2.4 Hierarchical Reinforcement Learning

Learning complex skills using simple Reinforcement Learning techniques is hard because of the reward sparsity problem. Hierarchical Reinforcement Learning offers a solution to this problem. In hierarchical RL we divid a complex task into smaller sub-tasks which are easy to learn. These sub-tasks are then combined together to perform the original task.

There exists a large amount of literature on hierarchical RL within the RL community (Dietterich [2000], Barto and Mahadevan [2003], Sutton et al. [1999]). This literature can be divided into two broad categories, *i.e.* hierarchical RL with and without *options* Sutton et al. [1999]. Options, allow us to use a set of initiaion and termination criterion (made available through the MDP), to decide on the appropriate sub-tasks for the original task. Recently, methods using *options* have been used to show good performance on difficult tasks such as Montezuma's Revenge (Kulkarni et al. [2016]).

In this work we solve the hierarchical RL problem without explicitly decomposing the task into sub-tasks. Instead we define a reward function for each simpler task which should encourage the agent to learn simpler skills more easily. This is similar to the hierarhcy of rewards proposed in Florensa et al. [2017]. In the next section, we discuss the algorithm most closely related with our proposed algorithm. Following this, we discuss the specific changes we introduce in the learning architecture.

3 Related Work

In this section, we discuss a recent work by Klimek et al. Klimek et al. [2017] which we build upon in this work. The task of object grasping was formulated as a hierarchical reinforcement learning problem. First, separate low level policies for each subtask such as those for reaching and gripping were pre-trained individually. Each task was pre-trained independently, using separate reward functions. For example, the reward functions used for the reacher and gripper subtasks were as follows

$$r_{reacher} = -d(H, C) \tag{15}$$

$$r_{gripper} = c_1 d(H, C) + c_2 \mathbb{1}_{d(H,C)} + c_3 d(FT_1, FT_2) \tag{16}$$

where H and C are the coordinates of the hand and the centre of the object respectively and FT_1 and FT_2 are the coordinates of the left and right finger joints of the hand. $c_1, c_3 < 0$ and $c_2 > 0$. Care was taken when training individual policies to keep the starting distribution similar to the distribution of states these policies would encounter when executed sequentially. Once the individual policies were trained, a high level policy called the *manger policy*, was trained to select the low level policy to execute, while keeping the low level policies fixed. The manager policy is executed after every τ time steps ($\tau = 8$ in the experiments) and provides the low level policy with the desired coordinates of the hand of the robot.

4 Our Method

While the experiments of Klimek et al. show impressive results, the authors note that they had to provide the manager policy with the ability to function like a low level policy when needed. We believe that this is because the manager and low level policies were not trained in an end-to-end fashion. Thus, the manager policy needs to learn to *fill in* whenever there is a gap when switching from one low level policy to another. In this work, we train the manager and low-level policies in an end-to-end fashion. Moreover, unlike previous approaches to hierarchical learning, we do not keep separate policies at the lower level. This has the two potential benefits. First, this makes the hierarchical approach scalable as the number of sub-tasks grow. Second, it allows knowledge of one sub-task to be shared with another.

In our approach, similar to prior work, the manager policy is executed after every τ time steps. However, unlike prior methods, the manager policy does not make a hard decision on which lower-level policy to select. Instead, the manager policy returns a soft-weighting of reward functions. The lower-level policies use this weighted reward for the next τ time steps. More formally, at time $n\tau$, the manager policy selects a weight $\alpha_n \in [0, 1]$ which is fixed till time step $(n+1)\tau - 1$. Then, the reward for the lower-level policy is given by

$$r^t = \alpha_n r^t_{reacher} + (1 - \alpha_n) r^t_{gripper} \tag{17}$$

for all time steps $t \in [n\tau, (n+1)\tau)$. Intuitively, we would expect the weight α_n to smoothly vary from 1 to 0, as the progress is made on the reacher task and the manager policy shifts focus on the gripper task. To enforce this behaviour, we also tried biasing α_n towards 0 after a fixed number of steps N. Formally,

$$\alpha'_n = \begin{cases} \alpha_n & \text{if } n < N \\ \max(\alpha_n - b, 0) & \text{if } n \geq N \end{cases}$$

We used $b = 0.5$ in our experiments. N was set to 100. We tried three different values of τ — 5, 10 and 20. The maximum episode length was set to 500.

5 Results

5.1 Environment

We use the physics engine MuJoCo Todorov et al. [2012] inside an OpenAI gym environment for our manipulation task. We focus on the **reach-and-grip** task using a *arm-claw* model. Figure **??** shown an image of the arm along with it's claws and the object. The object is initialized at random locations with a certain region. We make sure that there is sufficient distance between the gripper and the object. As discussed above we use the Proximal Policy Optimization algorithm with Generalized Advantage Estimation to train our reacher and gripper policies. We also experimented with different *manager time* (τ) values.

5.2 Discussion

First, we look at the results with direct application of our algorithm *i.e.*, without forcibly biasing the manager-policy. Specifically, we use α_n and not α'_n in this experiment. Figure 1 shows some of the frames when executing using this policy. Also, Figure 2 plots the reward and alpha values for the last episode during training. As can be seen in the figure, the alpha values during training seem to oscillate between 0.5 and 0.6. This indicates that the manager policy weighs rewards for both the tasks equally and is not able to correctly

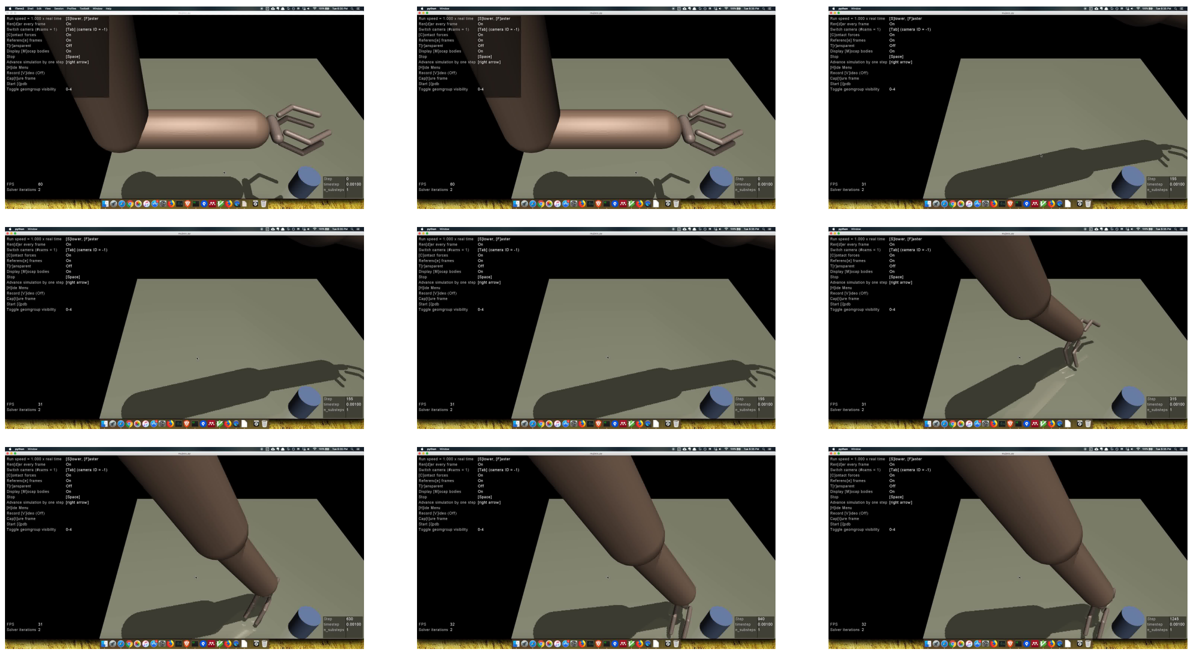

Figure 1: Results for experiment 1 with a manager policy weight determined from the manager policy directly (α_n). As can be seen below, the reacher reaches close to the object and then tries to reach further closer. The transition to the gripper policy does not happen smoothly. This happens since the target policy.

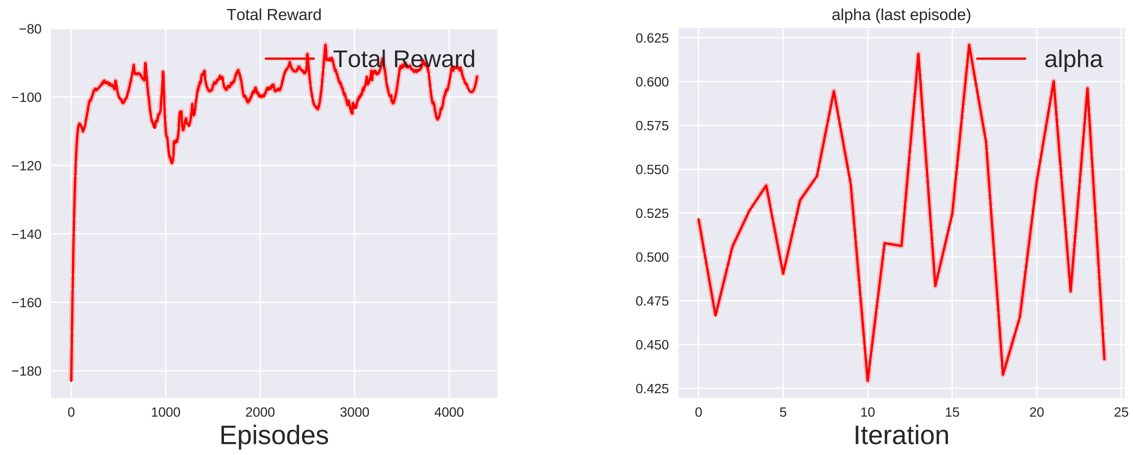

Figure 2: *(Left):* Total Reward during training over 4000 episodes. *(Right):* alpha values (Right) for the last episode during training.

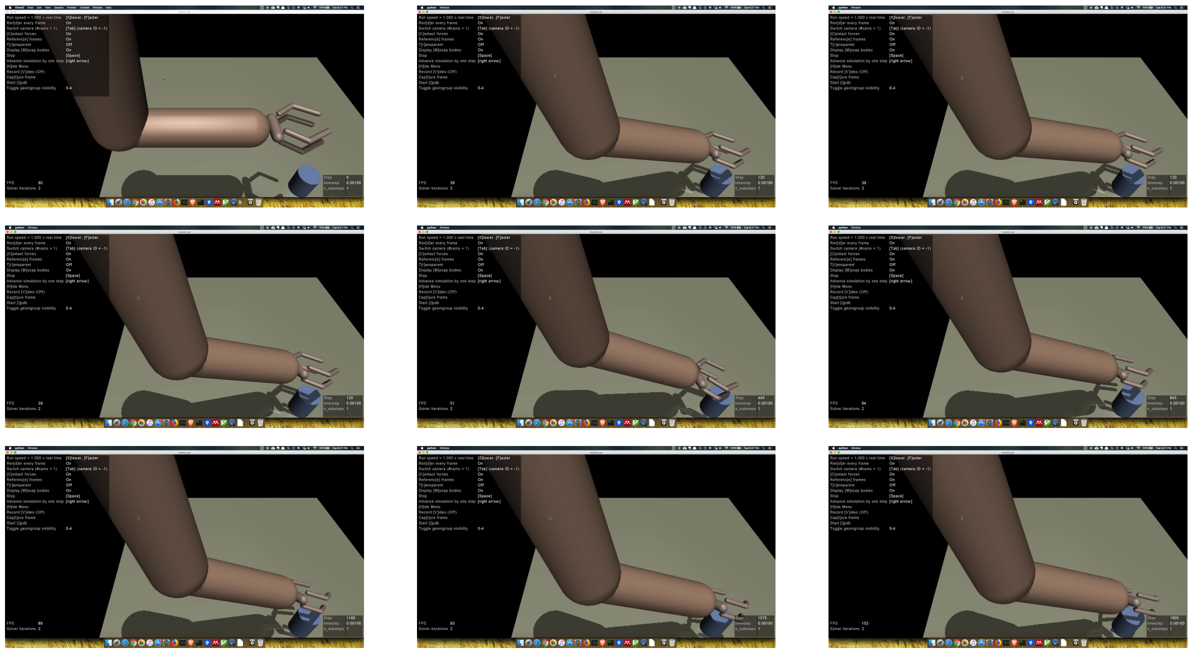

Figure 3: Results for experiment 2 with an adaptive manager policy weight (α_n'). As can be seen below, the reacher reaches close to the object and then the gripper policy tries to take over. However, the gripper policy is not able to correctly grasp the target.

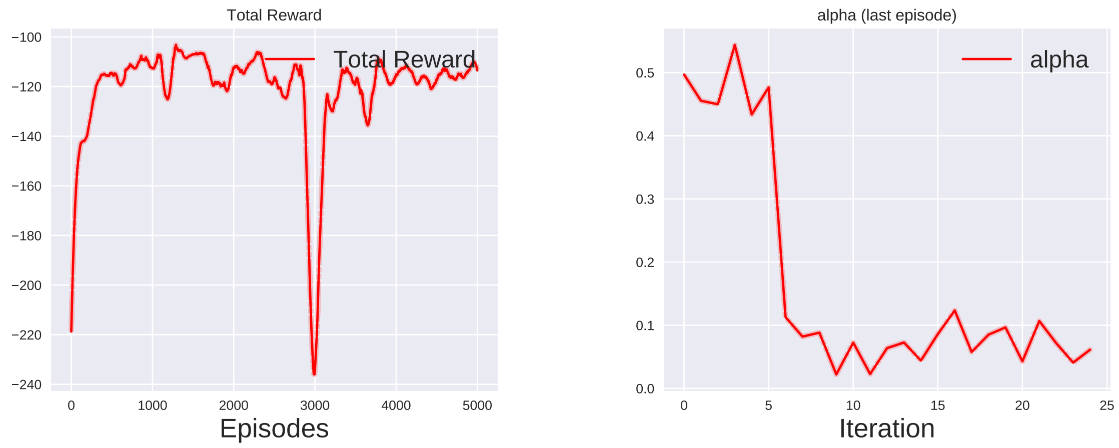

Figure 4: Results for adaptive alpha α_n' experiment. *(Left):* Total Reward during training over 4000 episodes. *(Right):* alpha values (Right) during the course of the training.

choose the appropriate weights for the different rewards. This can also be seen in Figure 1, wherein the claw is able to reach close to the object but is not able to grasp it. We believe that the supervisory signal for the manager policy is not strong enough and might require higher rewards for the gripper task to provide the manager policy with an incentive for switching to it.

Next we look at the results with for adaptive value of alpha *i.e.*, when using (α'_n). Figure 3 shows some of the frames when executing this policy. Also, Figure 4 plots the reward and alpha values for the last episode during training. As can be seen in the figure, the alpha values during training are initially close to 0.6 but after a few iterations it becomes close to 0. This indicates that the manager policy is able to correctly choose the appropriate weights due to the enforced bias. An effect of this bias can also be seen in Figure 3, wherein the claw is able to reach close to the object and is almost able to grasp it. While we could not achieve perfect gripping, possibly because the gripper task is much more complex than the reacher task, we believe that with some more improvement in reward design the claw should be able to grasp the object completely.

6 Conclusion

In this work we look at the task of object grasping using a robot arm. We break the grasping task into two sub-tasks — reaching and gripping. We use two levels in our hierarchical reinforcement learning algorithm. However, rather than training an individual policy for each sub-task we use a unified policy with different rewards for each sub-task. Additionally, we train a manager policy which weighs the reward for each sub-task appropriately during training. This design allows us to train a hierarchical reinforcement learning algorithm end-to-end without requiring to train individual policies separately. We use state-of-the-art policy optimization techniques to train both the manager and task policy. Through our experiments, we see that without any incentivization the manager is not able to correctly weight the different rewards. We also observe the inherent complexity of the grasping task as our simple reward design is not able to correctly grasp the object even when the manager chooses appropriate weights. This reflects the complex nature of manual reward design in traditional RL methods. We believe techniques such as imitation learning and perceptual grounding of rewards can help alleviate some such issues.

References

J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. *arXiv preprint arXiv:1705.10528*, 2017.

A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.(JAIR)*, 13:227–303, 2000.

C. Florensa, Y. Duan, and P. Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

M. Klimek, H. Michalewski, P. Mi, et al. Hierarchical reinforcement learning with parameters. In *Conference on Robot Learning*, pages 301–313, 2017.

T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *CoRR*, abs/1604.06057, 2016. URL `http://arxiv.org/abs/1604.06057`.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015a.

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

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