ROBUST RL FROM A META-RL PERSPECTIVE

KUN HUANG
PRATIK KUNAPULI

1. INTRODUCTION & MOTIVATION

Reinforcement learning is a powerful technique that can be applied to manipulation tasks, but such tasks are very object specific or limited to what the robot is exposed to during training. Thus, robots must learn to interact with multiple objects in their environment and these can be considered to be independent tasks. As human beings we are able to adapt and learn new tasks quickly using our prior knowledge on tasks sharing some similarities. Meta-Reinforcement Learning is interested in learning how best to learn the tasks and how learning some underlying policy from a subset of tasks can expedite the learning of future tasks. Within meta-reinforcement learning, there can be many different formulations of tasks in a collection. Some baselines such as Meta-World aim to set the standard for a number of tasks, but in this work we specifically examine how a domain randomization approach can be used to evaluate Meta-RL algorithms. Domain randomization is technique by which certain parameters are randomized so that a policy learned is invariant to those parameters, such as background colors, shapes, or lighting conditions. In this work, we are interested in investigating if domain randomization applied to physical properties of robots can be used to learn policies that perform well on multiple geometries and can be applied to new geometries.

2. BACKGROUND & METHOD

In a normal Q-learning setting, consider a Markov Decision Processes (MDP)

$$x_{t+1} = f^k(x_t, u_t, \xi_t)$$

where $x_t \in X \subset \mathbb{R}^d$ are the states, $u_t \in U \subset \mathbb{R}^p$ are the actions, ξ_t is the noise in the dynamics, and f^k is the dynamics function for the kth task. Then for a deterministic policy $u_\theta(x_t)$, the quality of state-action pair is defined as

$$q^k(x, u) = \mathop{\mathbb{E}}_{\xi_{(\cdot)}}[\sum_{t=0}^{\infty} \gamma^t \, r_t^k \mid x_0 = x, u_0 = u, u_t = u_\theta(x_t)]$$

where γ is the discount factor and r^k is the reward for kth task. We use φ to represent the Q-function and solve it by minimizing the Bellman error [1]

$$\widehat{\varphi^k} = \arg\min_{\varphi} \mathop{\mathbb{E}}_{\tau \sim D^k}[(q_\varphi(x, u) - r^k - \gamma \, q_\varphi(x', u_{\widehat{\theta^k}}(x')))^2]$$

then the deterministic controller can simply be

$$\widehat{\theta^k} = \arg\max_{\theta} \mathop{\mathbb{E}}_{\tau \sim D^k} [q_{\widehat{\varphi^k}}(x, u_\theta(x))]$$

2.1. **Meta-Reinforcement Learning.** Meta-RL aims to learn to solve a new task by training on a large number of tasks and use the prior knowledge of solving other tasks to accelerate the learning process of new tasks. The objective of meta-RL is to maximize the average returns over all tasks

$$\hat{\theta}_{meta} = \arg\max_{\theta} \frac{1}{n} \sum_{k=1}^{n} \mathop{\mathbb{E}}_{x \sim p_0} [q^k(x, u_\theta(x))]$$

Since our actual goal is to make the network quickly adapt to a new task, the objective can be written as maximizing the returns after one gradient descent [2]

$$\ell^k{}_{meta}(\theta) = \ell^k(\theta + \alpha \bigtriangledown_\theta \ell^k(\theta))$$

where $\ell^k = \mathbb{E}_{x \sim p_0}[q^k(x, u_\theta(x))]$.

2.2. **Meta-Q-Learning (MQL)** [3]**.** Different to the common Meta-Reinforcement Learning objective, the objective of Meta-Q-Learning is set to be the minimization of average TD value

$$\hat{\theta}_{meta} = \arg\min_{\theta} \frac{1}{n} \sum_{k=1}^{n} \mathop{\mathbb{E}}_{\tau \sim D^k} [TD^2(\theta)]$$

Notice that when computing the TD value, MQL uses the double-Q-learning trick [4] to stabilize the training process.

An important contribution of MQL is that they treat the tasks in meta-RL as the hidden variable of an underlying partially-observable Markov-Decision Process (POMDP). They therefore design a recurrent context variable z_t that depends on $\{(x_i, u_i, r_i)\}_{i \leq t}$, and use a Gated Recurrent Unit (GRU) [5] to learn such hidden context variable, which helps the learnt controller pick actions according to different tasks.

To reuse the experience collected during training time and adapt them to a new task at test time, MQL optimizes its control policy through two steps. The first step simply adjust its original control policy towards the new task

$$\arg\max_{\theta} \{ \mathop{\mathbb{E}}_{\tau \sim D^{new}} [l^{new}(\theta)] - \frac{\lambda}{2} \|\theta - \hat{\theta}_{meta}\|_2^2 \}$$

The quadratic penalty $\|\theta - \hat{\theta}_{meta}\|^2$ serves as a regularization term and keeps the network parameters close to original parameters. The second step focuses on exploiting the meta-training replay buffer. But simply using all trajectories in the replay buffer equally would not do well due to the existence of a large number of irrelevant trajectories and states. Therefore, they propose an importance ratio $\beta(\tau; D^{new}, D_{meta})$ to reweigh data from the meta-training replay buffer. The objective is then

$$\arg\max_{\theta} \{ \mathop{\mathbb{E}}_{\tau \sim D_{meta}} [\beta(\tau; D^{new}, D_{meta})\, l^{new}(\theta)] - \frac{\lambda}{2} \|\theta - \hat{\theta}_{meta}\|_2^2 \}$$

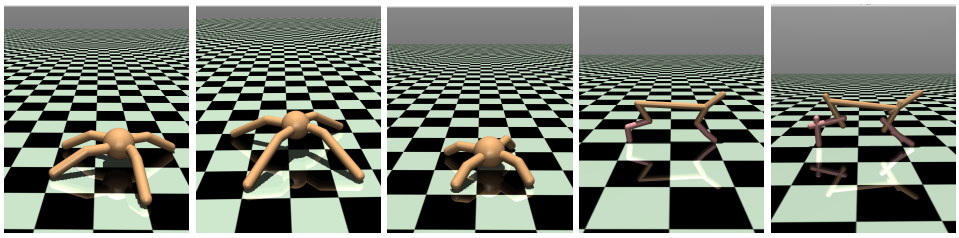

FIGURE 1. OpenAI Gym [7] Ant and Half-cheetah with various sizes. During experiments, we randomly sample a scaling factor for each task to modify the size of the agent.

2.3. **Multi-Task PPO.** Proximal Policy Optimization (PPO) is a model-free reinforcement learning method that has seen success in many controls tasks [6]. PPO is an on-policy method that iteratively improves the policy over time using first-order methods, like TRPO. Concretely, policy update for PPO is done as a greedy search:

$$\theta_{k+1} = \arg\max_{\theta} \mathbb{E}_{s,a\sim\pi_{\theta_k}}[L(s,a,\theta_k,\theta)]$$

where the objective function is given by:

$$L(s,a,\theta_k,\theta) = \min[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), g(\epsilon, A^{\pi_{\theta_k}}(s,a))]$$

such that:

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A, & A \geq 0 \\ (1-\epsilon)A, & A < 0 \end{cases}$$

This clipped formation of PPO ensures that the policy iteration at every step does not vary significantly (given by ϵ) between iterations. A is the advantage calculated for the state and action under policy π_{θ_k}. The Multi-Task variant of PPO (MT-PPO) is derived in a vary similar fashion, and uses the average performance of the multiple training environments in the loss.

3. EXPERIMENTS

This section presents the experimental results of MQL and MTPPO when they are trained and tested on agents with various sizes. We first go through the setup details and then compare the performance between MQL and MTPPO in two sets of experiments: Multi-task Experiments, where we train the policy on multiple agent sizes and test it on all seen agents; Zero-shot Transfer to New Tasks, where we train the policy on multiple agent sizes and test it on an unseen sized agent.

3.1. **Simulation Setup.** We use the MuJoCo [8] simulator with OpenAI Gym [7], specifically "Ant" and "Half-Cheetah" environments, on continuous-control tasks. We modify the length of the agent's legs to generate different tasks, as shown in Figure 1. During experiments, we randomly sample a scaling factor between 0.5 and 2 and multiply it with the original leg length.

3.2. **Multi-task Experiments.** We designed a multi-task experiment such that for each base environment (Ant-v3 or HalfCheetah-v2), we randomly generate 3 variants of the environment based on

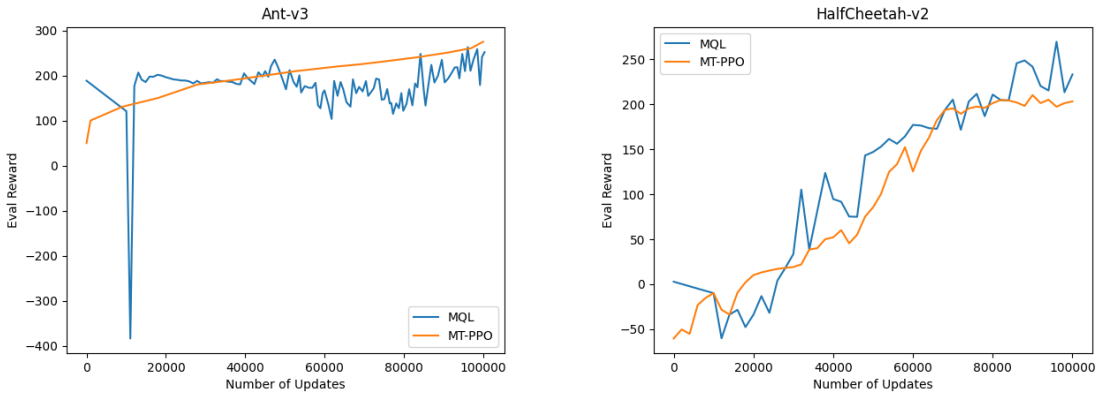

FIGURE 2. Performance of MQL and MT-PPO on Multi-Task Reinforcement Learning.

the scaling criteria explained above. For the multi-task experiment, each model was trained on all three of the environment variants and tested on the same three environments. This forces the models to learn a policy that performs well on average across all tasks, and is in-distribution generalization. The performance of MQL and MT-PPO are shown in Fig. 2.

Overall, both algorithms were able to show good generalization performance in the multi-task learning experiment. MT-PPO performed approximately as good as MQL, but was much slower to learn to the same level in the Ant environment. In the HalfCheetah environment, MQL outperformed MT-PPO. This could be due to the fact that the policy needed for better performance in this environment is more complex, and thus the MT-PPO could have achieved better results given much longer training times.

3.3. **Zero-shot Transfer to New Tasks.** For the zero-shot transfer experiment, we followed a similar procedure to generate three variants of each environment like we did in the multi-task experiment. The difference here, however, is that we withheld one of the three environments at random and only trained on the other two environments. During training, we evaluated the performance of the models on the hidden environment, representing a zero-shot tranfer of policy to a new task, or out-of-distribution generaliztion. The results of the MQL and MT-PPO performance are shown in Fig. 3.

Overall, both models were able to show increasing performance even on a hidden task, indicating that they can indeed perform well on zero-shot learning tasks given enough compute. Between the two algorithms, MQL outperformed MT-PPO on both the Ant and HalfCheetah environments, showing better performance on the hidden environmet. Although MT-PPO showed learning improvements in both environments, the learning rate was quite slow and would have needed much more iterations to achieve similar performance.

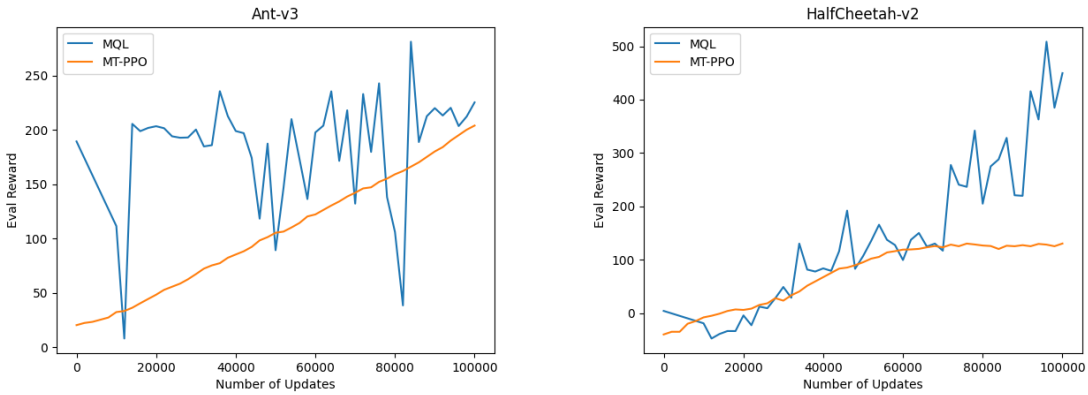

FIGURE 3. Performance of MQL and MT-PPO on Zero-Shot Reinforcement Learning.

4. CONCLUSION

In this work we show a new problem formulation for the Meta-Reinforcement Learning problem, by which domain randomization can be applied to physical properties of robots to generate new tasks. We evaluated the performance of two state-of-the-art algorithms, MQL and MT-PPO, on a set of experiments under this problem formulation. For the multi-task learning experiment, we show that both algorithms are able to learn adequate policies when being evaluated on the same set of tasks they are training on. For the zero-shot learning experiment, we showed that the MQL algorithm outperforms the MT-PPO in performance on an unseen task when trained on a set of other tasks. Overall this problem formulation is one that can be investigated further, and in the aim of achieving robust RL from a Meta-RL perspective should be pursued.

REFERENCES

[1] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[2] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[3] Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J Smola. Meta-q-learning. *arXiv preprint arXiv:1910.00125*, 2019.

[4] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[5] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

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

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[8] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.