AW-Opt: Learning Robotic Skills with Imitation and Reinforcement at Scale

Yao Lu[1], **Karol Hausman**[1], **Yevgen Chebotar**[1], **Mengyuan Yan**[2], **Eric Jang** [1],**Alexander Herzog**[2], **Ted Xiao**[1], **Alex Irpan**[1], **Mohi Khansari**[2], **Dmitry Kalashnikov**[1], **Sergey Levine**[1,3]

[1]Robotics at Google [2]X, The Moonshot Factory [3]UC Berkeley

A Appendix

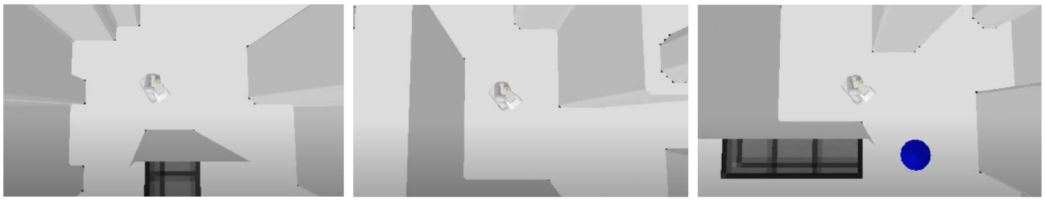

Figure 1: Illustration of the navigation task, based on LIDAR observations.

A.1 Experimental results with navigation task

For an additional test of the proposed algorithm, we compared QT-Opt, AWAC and AW-Opt on a point-to-point LIDAR-based navigation task (Task 6) (shown in Fig. 1) following the navigation training configuration in [**?**]. This task is significantly simpler than the image-based manipulation tasks, since the observation space is lower-dimensional (240 LIDAR points and a 2D goal, rather than 472 X 472 image), and the action space is smaller (a navigation twist with 2 degrees of freedom). Therefore, all three algorithms attain reasonable performance. However, as expected, AW-Opt and QT-Opt converge to significantly better final performance: 90% success rate at reaching the goal versus 70% for AWAC, and AW-Opt attains much better performance after offline pretraining compared to QT-Opt (20% vs 0% success rate), allowing it to learn the task up to the 90% success rate about three times faster than QT-Opt during online finetuning. AWAC requires a similar number of transitions to converge, but reaches a significantly worse final level of performance.

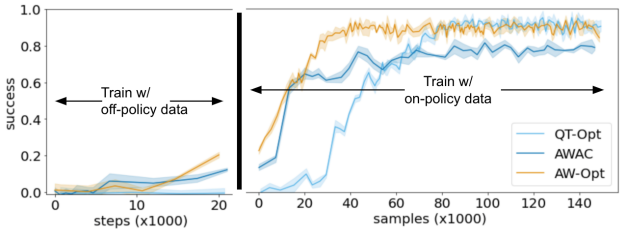

Figure 2: Comparison of QT-Opt, AWAC, AW-Opt on LIDAR-based navigation task.

A.2 Ablation study for episode-level random switcher

In Section **??**, we compared different exploration strategies and concluded that episode-level random switcher is the best strategy. In this section, we further compare the choice of ratio for actor vs. critic-based exploration. The default ratio is 80%/20% (critic/actor). Fig. 3 presents a comparison of different splits, on Task 1 and Task 3. Each run was repeated three times, with each of the following splits: 20%/80%, 50%/50%, 60%/40%, 70%/30%, 80%/20%, 90%/10%. For Task 1, we see similar results for the last three splits, while for Task 2, we see better results with the 30%/70% split.

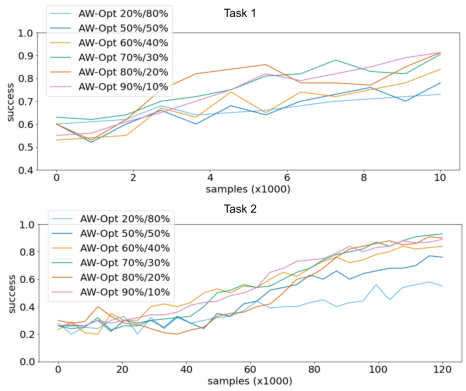

Figure 3: Comparison of different splits for Task and Task 2.

A.3 Experimental results with negative data

In Table **??**, we see that including "negatives" in the offline pretraining data for QT-Opt significantly improves QT-Opt's performance on Task 3 and Task 5, where the data was collected using a scripted policy. The negatives and positives are collected using the same scripted policy, with the positives treated as "successful demonstrations." We might wonder if an analogous trend would hold for other tasks, where human demonstrations are used as positives. In these cases, there are no corresponding negatives, but we can supply additional negatives by using a random policy. To examine the effect of these synthetic negatives, we evaluate Task 4 with either 10,000 or 100,000 additional random negatives. The results are shown in Fig 4. However, in this case, we see that QT-Opt is still not able to make any progress, remaining at 0% success rate throughout. This is not surprising. In the case of Task 3 and Task 5, the randomized scripted policy that collects the data has broad coverage, which means that many different actions are in-distribution. The "negatives" come from the same distribution as the positives (they are collected by the same scripted policy). Such a condition is known to be favorable for offline reinforcement learning [**?**], since when the data distribution is broad, fewer actions are out of distribution. Indeed, this is precisely the distribution used in the original QT-Opt work [**?**]. However, in the case of Task 4, the distribution of positives is very narrow. We cannot add negatives from the same distribution (since the distribution is over successful behaviors), which forces us to add negatives from a *different* distribution. While this does provide for broader coverage, it does not provide broader coverage in the region surrounding the positive examples, and therefore does not relieve the distributional shift challenge faced by offline RL.

A.4 Action space and loss function

For Task 1 and Task 3, the robot is operated in a 7D action space, namely: x, y, z, vertical rotation θ, open gripper, close gripper, terminate episode. The first 4 subactions are continuous while the last 3 subactions are discrete. For Task 2, Task 4 and Task 5, the action space is slightly different. It is operated on a 8D action space, namely: x, y, z angle axis x, angle axis y, angle axis z, gripper closedness, terminate episode.The first 7 subactions are continuous while the last subaction is discrete. The actor loss function considers both discrete and continuous subactions and also applies a weight on each one of the subaction. The full equation can be formulated as the following.

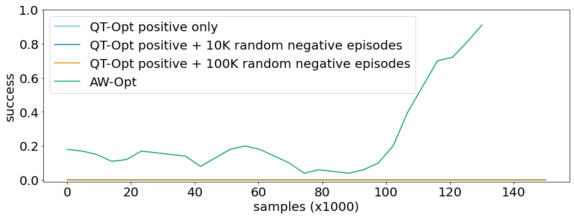

Figure 4: Comparison of QT-Opt offline training on Task 4 with only positives, positives and 10,000 negatives, and positives and 100,000 negatives. We can see that in all cases, QT-Opt makes no progress in this case. However, we can see that AW-Opt learns well even without any negatives.

$$L_A(a_t, A_{\phi_i}(s_t)) = \sum_{k=0}^{K} w_k MSE(a_{tk}, A_{\phi_i}(s_t)_k) \cdot \mathbb{1}_{\text{continuous}} + w_d cross\_entropy(a_t, A_{\phi_i}(s_t)) \cdot \mathbb{1}_{\text{discrete}}$$

where K is the total number of subactions. w_k is the weight to balance between each subaction. w_d is the weight for discrete subactions.

For Task1, the weights are summarized in Table. 1. For Task 2, Task 4, Task 5, the weights are summarized in Table. 2

Table 1: Loss weights for Task1.

Subaction	Weight
x	33.3
y	33.3
z	33.3
θ	5.5
discrete	1.0

Table 2: Loss weights for Task 2, Task 4, Task 5.

Subaction	Weight
x	6.0
y	6.0
z	6.0
angle axis x	3.0
angle axis y	3.0
angle axis z	3.0
gripper closedness	1.0
discrete	1.0

A.5 Network architecture

We apply exactly the same network architecture as QT-Opt[**?**] as the critic network for all RL algorithms and in all experiments we conducted. Actor network also uses the same backbone as the critic network. The difference is that it does not have action input and instead of outputting a single Q value, it outputs the action distribution (mean and variance of Gaussian distribution for continuous subactions and a one_hot vector of the discrete actions).

During the training, critic network and actor network do not share weights between each other. They are trained separately.

We plan to open source AW-Opt algorithm once the paper is published.

A.6 Policy Action Selection Runtime

A significant limitation of QT-Opt and other methods that do not employ an actor in continuous action spaces is that action-selection at evaluation time requires an optimization with respect to the critic. In this section, we compare the run time of action selection for QT-Opt, AWAC, AW-Opt on different robots. Both AWAC and AW-Opt use an actor network after training, while QT-Opt requires optimization over actions with CEM. As we can see in Table 3, this results in AW-Opt and AWAC selecting actions about three times faster than QT-Opt. We measure the inference time on a Quadro P1000 GPU.

Table 3: Run time comparison between QT-Opt, AWAC, AW-Opt on different robots.

Algorithm	Kuka arm	Pica
QT-Opt	125ms	137ms
AWAC	42ms	48ms
AW-Opt	42ms	48ms

A.7 Datasets

In the following we summarize the dataset size and type for different tasks.

Table 4: Dataset information for different tasks.

	Offline data type	sim/real	Offline data size	Positive/ Negative ratio	Finetuning data size	sim/real
Task 1	off-policy	simulation	1000	30%/70%	20K	simulation
Task 2	demonstration	simulation	120	100%/0%	120K	simulation
Task 3	off-policy	real	320K	40%/60%	0	N/A
Task 4	demonstration	real	300	100%/0%	150K	simulation + RL cyclegan
Task 5	demonstration	real	300	100%/0%	200K	simulation + RL cyclegan
Task 6	off-policy	sim	100	100%/0%	1K	simulation