Supplementary Material for: Regret Minimization for Partially Observable Deep Reinforcement Learning

Peter Jin Kurt Keutzer Sergey Levine

A1. Experimental Details

For each experiment, we performed 5 trials for each method (some TRPO experiments on Pong are for 3 trials).

A1.1. Hyperparameters

Our hyperparameter choices for each method are listed below. For all methods using Adam, we roughly tuned the learning rate to find the largest that consistently converged in unmodified variants of tasks.

A1.1.1. ARM

Please see Tables 1 and 2 for hyperparameters used with ARM. Note that our choice of ARM hyperparameters yields an equivalent number of minibatch gradient steps per sample as used by deep Q-learning, i.e. 1 Adam minibatch gradient step per 4 simulator steps; c.f. Table 4 for the deep Q-learning hyperparameters. We kept hyperparameters (other than the learning rate and the number of steps n) constant across tasks.

Table 1. Hyperparameters for ARM.

Hyperparameter	Atari	Doom	Minecraft
Adam learning rate	$1e^{-4}$	$1e^{-5}$	$1e^{-5}$
Adam minibatch size	32	32	32
batch size	12500	12500	12500
gradient steps	3000	3000	3000
moving average (τ)	0.01	0.01	0.01
n-steps	1	5	5

Table 2. Hyperparameters for off-policy ARM.

Hyperparameter	Doom
Adam learning rate	$1e^{-5}$
Adam minibatch size	32
batch size	1563
gradient steps	400
importance weight clip	1
moving average (τ)	0.01
n-steps	5
replay memory max	25000

A1.1.2. A2C

Please see Table 3 for hyperparameters used with A2C. We found that increasing the number of steps n used to calculate the n-step returns was most important for getting A2C/A3C to converge on Doom MyWayHome.

Table 3. Hyperparameters for A2C.

Hyperparameter	Doom
Adam learning rate	$1e^{-4}$
Adam minibatch size	640
entropy bonus (β)	0.01
gradient clip	0.5
n-steps	40
num. workers	16

A1.1.3. DQN

Please see Table 4 for hyperparameters used with deep Q-learning. Dueling double DQN uses the tuned hyperparameters (van Hasselt et al., 2016; Wang et al., 2016). In particular, we found that dueling double DQN generally performed better and was more stable when learning on Atari with the tuned learning rate $6.25 \times 10^{-5} \approx 6 \times 10^{-5}$ from Wang et al. (2016), compared to the slightly larger learning rate of 1×10^{-4} used by ARM.

Table 4. Hyperparameters for dueling + double deep Q-learning.

Hyperparameter	Atari	Doom	Minecraft
Adam learning rate	$6e^{-5}$	$1e^{-5}$	$1e^{-5}$
Adam minibatch size	32	32	32
final exploration	0.01	0.01	0.01
gradient clip	10	—	—
n-steps	1	5	5
replay memory init	50000	50000	12500
replay memory max	10^6	240000	62500
sim steps/grad step	4	4	4
target update steps	30000	30000	12500

A1.1.4. TRPO

Please see Table 5 for hyperparameters used with TRPO. We generally used the defaults, such as the KL step size of 0.01 which we found to be a good default. Decreasing the batch size improved sample efficiency on Doom and Minecraft without adversely affecting the performance of the learned policies.

Table 5. Hyperparameters for TRPO.

HYPERPARAMETER	ATARI	DOOM	MINECRAFT
BATCH SIZE	100000	12500	6250
CG DAMPENING	0.1	0.1	0.1
CG ITERATIONS	10	10	10
KL STEP SIZE	0.01	0.01	0.01

A1.2. Environment and Task Details

Our task-specific implementation details are described below.

A1.2.1. ATARI

For the occluded variant of Pong, we set the middle region of the 160×210 screen with x, y pixel coordinates $[55 \ldots 105), [34 \ldots 194)$ to the RGB color $(144, 72, 17)$. The image of occluded Pong in Figure 4 from the main text has a slightly darker occluded region for emphasis.

We use the preprocessing and convolutional network model of Mnih et al. (2013). Specifically, we view every 4th emulator frame, convert the raw frames to grayscale, and perform downsampling to generate a single observed frame. The input observation of the convnet is a concatenation of the most recent frames (either 4 frames or 1 frame). The convnet consists of an 8×8 convolution with stride 4 and 16 filters followed by ReLU, a 4×4 convolution with stride 2 and 32 filters followed by ReLU, a linear map with 256 units followed by ReLU, and a linear map with $|\mathcal{A}|$ units where $|\mathcal{A}|$ is the action space cardinality ($|\mathcal{A}| = 6$ for Pong).

A1.2.2. DOOM

Our modified environment "Doom Corridor+" is very closely derived from the default "Doom Corridor" environment in ViZDoom. We primarily make two modifications: (a) first, we restrict the action space to the three keys $\{MoveRight, TurnLeft, TurnRight\}$, for a total of $2^3 = 8$ discrete actions; (b) second, we set the difficulty ("Doom skill") to the maximum of 5.

For the occluded variant of Corridor+, we set the middle region of the 160×120 screen with x, y pixel coordinates $[30 \ldots 130), [10 \ldots 110)$ to black, i.e. $(0, 0, 0)$.

For Corridor+, we scaled rewards by a factor of 0.01. We did not scale rewards for MyWayHome.

The Doom screen was rendered at a resolution of 160×120 and downsized to 84×84. Only every 4th frame was rendered, and the input observation to the convnet is a concatenation of the last 4 rendered RGB frames for a total of 12 input channels. The convnet contains 3 convolutions with 32 filters each: the first is size 8×8 with stride 4, the second is size 4×4 with stride 2, and the third is size 3×3 with stride 1. The final convolution is followed by a linear map with 1024 units. A second linear map yields the output. Hidden activations are gated by ReLUs.

A1.2.3. MINECRAFT

Our Minecraft tasks are based on the tasks introduced by Matiisen et al. (2017), with a few differences. Instead of using a continuous action space, we used a discrete action space with 4 move and turn actions. To aid learning on the last level ("L5"), we removed the reward penalty upon episode timeout and we increased the timeout on "L5" from 45 seconds to 75 seconds due to the larger size of the environment. We scaled rewards for all levels by 0.001.

We use the same convolutional network architecture for Minecraft as we use for ViZDoom. The Minecraft screen was rendered at a resolution of 320×240 and downsized to 84×84. Only every 5th frame was rendered, and the input observation of the convnet is a concatenation of the last 4 rendered RGB frames for a total of 12 input channels.

A2. Off-policy ARM via Importance Sampling

Our current approach to running ARM with off-policy data consists of applying an importance sampling correction directly to the n-step returns. Given the behavior policy μ under which the data was sampled, the current policy π_t under which we want to perform estimation, and an importance sampling weight clip c for variance reduction, the corrected n-step return we use is:

$$g_k^n(\mu \| \pi_t) = \sum_{k'=k}^{k+n-1} \gamma^{k'-k} \left(\prod_{\ell=k}^{k'} w_{\mu\|\pi_t}(a_\ell | o_\ell) \right) r_{k'} + \gamma^n V'(o_{k+n}; \varphi) \quad (1)$$

where the truncated importance weight $w_{\mu\|\pi_t}(a|o)$ is defined:

$$w_{\mu\|\pi_t}(a|o) = \min\left(c, \frac{\pi_t(a|o)}{\mu(a|o)} \right). \quad (2)$$

Note that the target value function $V'(o_{k+n}; \varphi)$ does not require an importance sampling correction because V' already approximates the on-policy value function $V_{\pi_t}(o_{k+n};\theta_t)$. Our choice of $c = 1$ in our experiments was inspired by Wang et al., (2017). We found that $c = 1$ worked well but note other choices for c may also be reasonable.

When applying our importance sampling correction, we preserve all details of the ARM algorithm except for two aspects: the transition sampling strategy (a finite memory of previous batches are cached and uniformly sampled) and the regression targets for learning the value functions. Specifically, the regression targets v_k and q_k^+ (Equations (11)–(14) in the main text) are modified to the following:

$$v_k = g_k^n(\mu\|\pi_t) \tag{3}$$
$$q_k = (1 - w_{\mu\|\pi_t}(a_k|o_k))r_k + g_k^n(\mu\|\pi_t) \tag{4}$$
$$\phi_k = Q_{t-1}^+(o_k, a_k; \omega_{t-1}) - V_{\pi_{t-1}}(o_k; \theta_{t-1}) \tag{5}$$
$$q_k^+ = \max(0, \phi_k) + q_k. \tag{6}$$

A3. Additional Experiments

A3.1. Recurrence in Doom MyWayHome

We evaluated the effect of recurrent policy and value function estimation in the maze-like MyWayHome scenario of ViZDoom. For the recurrent policy and value function, we replaced the first fully connected operation with an LSTM featuring an equivalent number of hidden units (1024). We found that recurrence has a small positive effect on the convergence of A2C, but was much less significant than the choice of algorithm; compare Figure 2 in the main text with Figure 7 below.

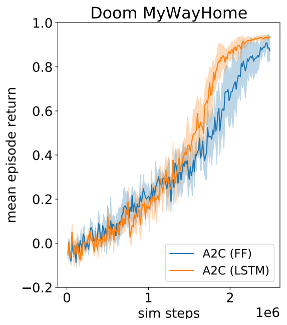

Figure 7. Comparing A2C with a feedforward convolutional network (blue) and a recurrent convolutional-LSTM network (orange) on the ViZDoom scenario MyWayHome.

A3.2. Atari 2600 games

Although our primary interest is in partially observable reinforcement learning domains, we also want to check that ARM works in nearly fully observable and Markovian environments, such as Atari 2600 games. We consider two baselines: double deep Q-learning, and double deep fitted Q-iteration which is a batch counterpart to double DQN. We find that double deep Q-learning is a strong baseline for learning to play Atari games, although ARM still successfully learns interesting policies. One major benefit of Q-learning-based methods is the ability to utilize a large off-policy replay memory. Our results on a suite of Atari games are in Figure 8.

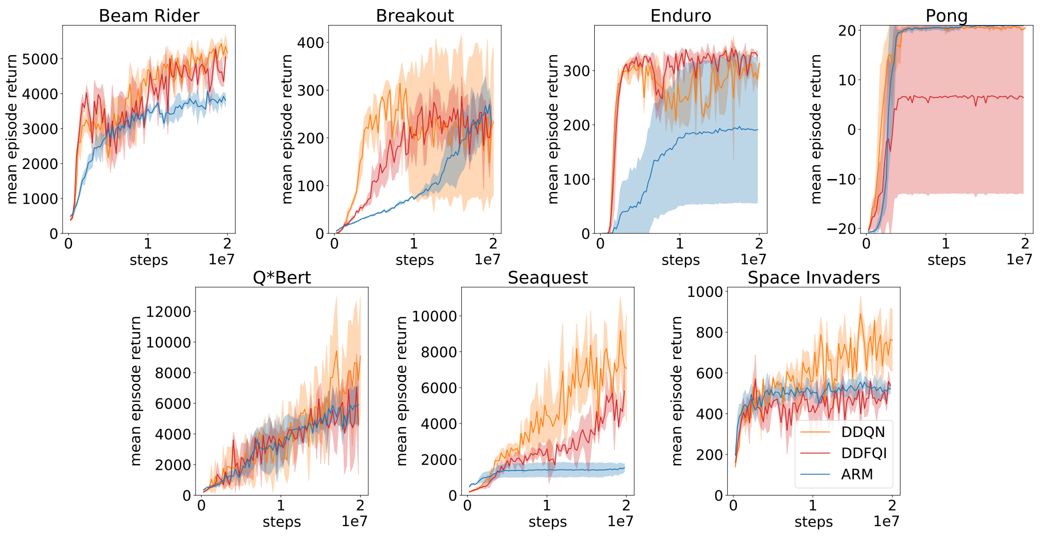

Figure 8. Comparing double deep Q-learning (orange), double deep fitted Q-iteration (red), and ARM (blue) on a suite of seven Atari games from the Arcade Learning Environment.