## 6 APPENDIX

### 6.1 EXPERIMENTAL DETALIS

We run experiments on 4 MuJoCo environments: *Half-Cheetah*, *Walker*, *Hopper* and *Humanoid* and on 6 Atari games: *Boxing*, *Breakout*, *Freeway*, *Gopher*, *Pong* and *Seaquest*. Our code with the exact configurations we use to reproduce the experiments is available as open source[2]. We use QWR as described in Algorithm 2 with the following hyper-parameters:

- We set `n_iterations` to 100 on MuJoCo and on Atari we stop on each game after reaching 100K interactions with the environment (episodes are limited to 2000 interactions during training).
- We set `n_critic_steps` to 3000 and `update_frequency` to 100, so the Q target network is updated 30 times in each epoch.
- We set `n_actor_steps` to 3000 as well.
- The number of samples is 8 by default and the margin is 3, see Section 4.2 for ablations and discussion of those values.

When training the networks, we use the Adam optimizer. Learning rate for the critic is set to $5e-4$ and for the actor it is set to $1e-4$ for all our experiments. We use the standard architectures for deep networks. In MuJoCo experiments we use a multi-layer perceptron with two layers 256 neurons each and ReLU activations. In Atari experiments we use the same convolutional architectures as Mnih et al. (2013a).

### 6.2 ANALYSIS OF AWR WITH LIMITED DATA

Since sample efficiency is one of the key challenges in deep reinforcement learning, it would be desirable to have better tools to understand why any RL algorithm – for instance AWR – is sample efficient or not. This is hard to achieve in the general setting, but we identify a key simplifying assumption that allows us to solve AWR analytically and identify the source of its problems.

The assumption we introduce, called *state-determines-action*, concerns the content of the replay buffer $\mathcal{D}$ of an off-policy RL algorithm. The replay buffer contains all state-action pairs that the algorithm has visited so far during its interactions with the environment. We say that a replay buffer $\mathcal{D}$ satisfies the *state-determines-action* assumption when for each state $s$ in the buffer, there is a unique action that was taken from it, formally:

$$\text{for all } (s,a),(s',a') \in \mathcal{D} : s = s' \implies a = a'.$$

This assumption may seem very limiting and indeed – it is not true in many experimental runs of RL algorithms. Even a random policy starting from the same state will violate the assumption the second time it collects a trajectory. But in the case of limited data, when only few trajectories were collected, this assumption may hold, at least for a large subset of the replay buffer. This makes it relevant to the study of sample efficiency – we argue that an algorithm that does not perform well under this assumption will rarely be sample-efficient. We believe that good performance under the *state-determines-action* assumption is also necessary for RL algorithms to scale well to larger real-world settings. As the Greek philosopher Heraclitus said, no man ever steps in the same river twice. In real world the same state is never visited twice to give a chance to take different actions.

A simplifying assumption like *state-determines-action* is useful only if it indeed simplifies the analysis of RL algorithms. We show that in case of AWR it does even more – it allows us to analytically calculate the final policy that the algorithm produces. In the case of AWR, it turns out that the resulting policy yields no improvement over the sampling policy.

While AWR achieves very good results after longer training, it is not very sample efficient, as noted in the future work section of (Peng et al., 2019). To address this problem, let us analyze a single loop of actor training in AWR:

$$\pi_{\mathcal{D}}^{i+1} \leftarrow \arg\max_{\pi} \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}} \left[ \log \pi(\mathbf{a}|\mathbf{s}) \exp\left( \frac{1}{\beta} (\mathcal{R}_{\mathcal{D}}^{\mathbf{s},\mathbf{a}} - V_{\mathcal{D}}^{i}(\mathbf{s})) \right) \right] \tag{6}$$

[2] `url_removed_to_preserve_anonymity`

For simplicity, let us denote

$$\alpha_{\mathcal{D}}^{\mathbf{s},\mathbf{a}} = \exp\left(\frac{1}{\beta}(\mathcal{R}_{\mathcal{D}}^{\mathbf{s},\mathbf{a}} - V_{\mathcal{D}}^{i}(\mathbf{s}))\right) \tag{7}$$

so the above update can be written as:

$$\pi_{\mathcal{D}}^{i+1} \leftarrow \arg\max_{\pi} \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}\left[\log \pi(\mathbf{a}|\mathbf{s})\alpha_{\mathcal{D}}^{\mathbf{s},\mathbf{a}}\right] \tag{8}$$

How this update acts on a replay buffer that satisfies the *state-determines-action* assumption? It turns out that we can answer this question analytically using the following theorem.

**Theorem 1.** *Let a replay buffer $\mathcal{D}$ satisfy the state-determines-action assumption and let $\pi_{\mathcal{D}}$ be a policy that clones the behaviour from $\mathcal{D}$, i.e., that assigns to each state s from $\mathcal{D}$ the action $a$ such that $(s, a) \in \mathcal{D}$ with probability $1$. Then, under the AWR update, $\pi_{\mathcal{D}}^{i+1} \leftarrow \pi_{\mathcal{D}}$.*

*Proof.* By definition of the AWR update rule, $\pi_{\mathcal{D}}^{i+1} \leftarrow \arg\max_{\pi} \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\mathcal{D}}[\log \pi(\mathbf{a}|\mathbf{s})\alpha_{\mathcal{D}}^{\mathbf{s},\mathbf{a}}]$. Recall that the number $\alpha_{\mathcal{D}}^{\mathbf{s},\mathbf{a}}$ is non-negative as it is an exponent of another number. Thus the value $\log \pi(\mathbf{a}|\mathbf{s})\alpha_{\mathcal{D}}^{\mathbf{s},\mathbf{a}}$ can be at most $0$, since the logarithm of a probability is negative or $0$. The policy $\pi_{\mathcal{D}}$ assigns probability $1$ to the action $a$ in state $s$, so it does reach the value $0$ and therefore reaches $\arg\max_{\pi}$ as required. $\square$

As we can see from the above theorem, the AWR update rule will insist on cloning the action taken in the replay buffer as long as it satisfies the *state-determines-action* assumption. In a deterministic environment, the new policy $\pi_{\mathcal{D}}$ will not add any new data to the buffer, only replay a trajectory already in it. So the whole AWR loop will end with the policy $\pi_{\mathcal{D}}$, which yields no improvement. Note that this theorem is also useful in the case of learning continuous policies. For instance, a Gaussian policy with parameterized variance can learn to decrease the variance to be arbitrarily close to zero, in the limit approaching the distribution defined in the theorem.

How come AWR works so well in practice, even with not-that-many interactions? First of all, note that for the above effect to occur in practice, the neural network used for AWR actor must be large enough and trained long enough to memorize the data from the replay buffer. Even when this is the case, the policy it learns must be allowed to express distrubutions that assign probability 1 to a single action. This holds for environments with discrete actions, but it is not true when using Gaussian distributions with fixed variance in continuous action spaces. But even in that case, using an algorithm that corrects this problem leads to improved sample efficiency, as we show below.

### 6.3 Multi-step targets

To make the training of the Q-value network more efficient, we implement an approach inspired by widely-used multi-step Q-learning (Mnih et al., 2016). We consider targets for the Q-value network computed over multiple different time horizons:

$$Q_{\mu,t}^{\star}(\mathbf{s}_1, \mathbf{a}_1) = \sum_{i=1}^{t} \gamma^{i-1}\mathbf{r}_i + \gamma^t \mathbb{E}_{\mathbf{a}'_1,\ldots,\mathbf{a}'_n\sim\mu(\cdot|\mathbf{s}_{t+1})} F(\{Q_\mu(\mathbf{s}_{t+1}, \mathbf{a}'_1), \ldots, Q_\mu(\mathbf{s}_{t+1}, \mathbf{a}'_n)\}) \tag{9}$$

where $\mathbf{s}_i$, $\mathbf{a}_i$, $\mathbf{r}_i$ are the states, actions and rewards in a collected trajectory, respectively. We aggregate those multi-step targets using a truncated TD($\lambda$) estimator (Sutton & Barto, 2018, p. 236):

$$Q_{\mu}^{\star}(\mathbf{s}, \mathbf{a}) = (1-\lambda)\sum_{t=1}^{T} \lambda^{t-1} Q_{\mu,t}^{\star}(\mathbf{s}, \mathbf{a}) \tag{10}$$