Asymmetric DQN for Partially Observable Reinforcement Learning

Andrea Baisero[1] **Brett Daley**[1] **Christopher Amato**[1]

[1]Khoury College of Computer Sciences, Northeastern University, Boston, Massachusetts, USA

Abstract

Offline training in simulated partially observable environments allows reinforcement learning methods to exploit privileged state information through a mechanism known as asymmetry. Such privileged information has the potential to greatly improve the optimal convergence properties, if used appropriately. However, current research in asymmetric reinforcement learning is often heuristic in nature, with few connections to underlying theory or theoretical guarantees, and is primarily tested through empirical evaluation. In this work, we develop the theory of *Asymmetric Policy Iteration*, an exact model-based dynamic programming solution method, and then apply relaxations which eventually result in *Asymmetric DQN*, a model-free deep reinforcement learning algorithm. Our theoretical findings are complemented and validated by empirical experimentation performed in environments which exhibit significant amounts of partial observability, and require both information gathering strategies and memorization.

1 INTRODUCTION

Offline training and online execution (OTOE) is a modern reinforcement learning (RL) paradigm in which a learning agent is trained *offline* (i.e., in simulation) before becoming operational *online* (i.e., in the "real" environment). Advantages of OTOE are broad and include safety guarantees, training speed, flexibility, and—the focus of our work—access to privileged information. For all these reasons, OTOE has even become the paradigm of preference in some research cliques, such as that of multi-agent RL, where it is often called centralized training and decentralized execution (CTDE). Privileged information is data which is accessible during offline training, but not during standard online training and/or execution. This can take different forms depending on the type of control problem, e.g., other agents' actions and observations in multi-agent RL, or the system's state in partially observable RL (PORL). In OTOE, access to this information is a temporary privilege, available exclusively in the offline phase due to access of the simulation's internal state. However, despite being not available during online execution, such information has the potential (when used appropriately) to improve the agent's overall training performance and/or convergence speed, and therefore its online performance.

In PORL, OTOE and privileged information is most commonly associated with actor-critic methods through a mechanism called *asymmetry*. Asymmetry has a very specific etymological meaning described below; however, we use the term "asymmetry" more loosely to also refer to the general idea of exploiting privileged information during offline training—two concepts which overlap strongly in this work. In actor-critic methods, two separate models are being trained: a *policy* model (representing the agent's behavior) and a *critic* model (representing the agent's evaluation of its situation). Standard actor-critic methods can be said to be implicitly *symmetric* in the sense that both models receive the same information—in PORL, the agent's history. In *asymmetric* actor-critic, this symmetry is broken by providing privileged information to the critic [Pinto et al., 2018, Foerster et al., 2018, Lowe et al., 2017, Yang et al., 2018, Li et al., 2019, Wang et al., 2020, Warrington et al., 2021, Xiao et al., 2021, Baisero and Amato, 2022, Lyu et al., 2022]. This is possible because the critic is exclusively a training construct which is not used or needed during the execution phase. Asymmetry has also been used in some DQN-like RL methods [Rashid et al., 2018, Mahajan et al., 2019, Rashid et al., 2020, Xiao et al., 2020, de Witt et al., 2020], where normally there would not be a secondary model analogous to the critic which is only used during training. In such cases, a second value-based model is introduced exclusively as a means for asymmetry, and which constitutes a training construct analogous to that of the critic in actor-critic.

However, a substantial majority of prior work in asymmetric RL has proposed heuristic forms of asymmetry primarily verified through empirical evaluations, but which lack the support of a theoretical framework which guarantees the state information is used in an appropriate fashion. We argue that, if unverified by proper theoretical analysis, such methods could quite simply make use of state information in ways which actually hinders the training of a partially observable agent. For example, a well-known result in partially observable control is that the optimal action for a partially observable agent can differ greatly from that of an optimal fully observable agent, and that an optimal partially observable agent might even take actions which an optimal fully observable agent would never take under any circumstance, e.g., information-gathering actions that help the partially observable agent learn something about the environment state, but do not help the fully observable agent.

The ultimate goal of this work is to develop a state-of-the-art asymmetric value-based deep RL algorithm for partially observable control that is supported by a sound theoretical analysis. To reach this goal, we employ a bottom-up approach, focusing first on developing the theory of *asymmetric policy improvement*, i.e., mechanisms through which privileged state information can be integrated into a policy improvement process while retaining optimal convergence guarantees. In practice, we begin by developing *Asymmetric Policy Iteration* (API) and *Asymmetric Action-Value Iteration*, model-based dynamic programming solution methods. We then introduce elements of stochastic training from sample experience which result in *Asymmetric Q-Learning* (AQL), a direct RL successor to API and AAVI. Finally, we introduce value-function approximation which results in *Asymmetric DQN* (ADQN), a method comparable to other state-of-the-art deep RL algorithms, but which is also capable of exploiting state informationin a principled fashion. To the best of our knowledge, our work is the first to develop theoretically-driven asymmetric value-based RL.

2 RELATED WORK

Privileged information available offline has been used to improve training performances in a wide range of prior single-agent and multi-agent methods which include both policy-based and value-based methods.

In sigle-agent control, Pinto et al. [2018] employ DDPG with an asymmetric state-based critic to handle robot manipulation tasks; belief-grounded networks [Nguyen et al., 2021] uses a belief-based form of asymmetry and a belief-reconstruction task to train the history representation; Warrington et al. [2021], Chen et al. [2020] use imitation learning to train a partially observable agent via a fully observable agent trained offline. Baisero and Amato [2022] show theoretical issues with state-only forms of asymmetry for policy-gradients, and develop a history-state variant.

In multi-agent control, COMA [Foerster et al., 2018] uses a single centralized asymmetric critic which employs the joint observations and/or the environment state. MADDPG [Lowe et al., 2017] and M3DDPG [Li et al., 2019] use multiple centralized asymmetric critics, one for each agent, which employ the joint observations and/or the environment state. R-MADDPG [Wang et al., 2020] uses a recurrent model and a centralized critic which uses the entire histories of all agents; CM3 [Yang et al., 2018] uses a state-only critic for reactive control; MacDec-DDRQN [Xiao et al., 2020] uses a centralized value model to learn individual centralized value models. ROLA [Xiao et al., 2021] uses both centralized and individual asymmetric critics which employ local history and/or state information to estimate individual advantage values. QMIX [Rashid et al., 2018], MAVEN [Mahajan et al., 2019], and WQMIX [Rashid et al., 2020] use a centralized but factored value model to train individual agent value models. Lyu et al. [2022] extend the theory by Baisero and Amato [2022] to the multi-agent case.

3 BACKGROUND

In the next subsection, we present some of the background required to understand our work. Section 3.1 formally describes partially observable control problems. Section 3.2 contains a review of non-asymmetric value-based control, in the form of DQN. Section 3.3 covers the definition of *history-state* value functions. In Section 3.4, we present operator notation and useful operators.

3.1 POMDPS

A partially observable Markov decision process (POMDP) is a discrete-time control problem represented by tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, b_0, T, O, R, \gamma \rangle$, where (a) $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{O}$ are state, action, and observation spaces, (b) $b_0 \in \Delta\mathcal{S}$ is an initial state distribution, (c) $T\colon \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$ is a stochastic state transition function, (d) $O\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Delta\mathcal{O}$ is a stochastic observation emission function, (e) $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, and (f) $\gamma \in [0, 1)$ is a discount factor.

Partially observable control is based on observable *histories*, i.e., the sequences of past actions and observations. The *history* space $\mathcal{H} \doteq (\mathcal{A} \times \mathcal{O})^*$ represents such sequences. To simplify notation, we overload symbol R to also denote the expected reward function on histories $R(h, a) \doteq \mathbb{E}_{s|h}\left[R(s, a)\right]$. General partially observable policies take the form of mappings from *histories* to action distributions $\pi\colon \mathcal{H} \to \Delta\mathcal{A}$; however, in this work, we will focus exclusively on *deterministic* policies $\pi\colon \mathcal{H} \to \mathcal{A}$. The goal of the control problem is to find a policy which maximizes the episodic expected *return* $\mathbb{E}\left[\sum_t \gamma^t R(s_t, a_t)\right]$.

Every policy π is associated with an action-value function $Q^\pi(h, a)$ which represents the expected return associated with the agent having observed history h, taking action a, and then continuing to behave according to the policy π. Q^π is the unique solution to the Bellman equation,

$$Q^\pi(h, a) = R(h, a) + \gamma\, \mathbb{E}_{o|h,a}\left[Q^\pi(hao, \pi(hao))\right] . \quad (1)$$

The action-value function associated with the optimal policy π^* is denoted as Q^*, and is the unique solution to the Bellman *optimality* equation,

$$Q^*(h, a) = R(h, a) + \gamma\, \mathbb{E}_{o|h,a}\left[\max_{a'} Q^*(hao, a')\right] . \quad (2)$$

Notation We use symbols Q, Q^π, Q^*, and $\hat{Q}$ to denote similar but separate concepts. $Q\colon \mathcal{H} \times \mathcal{A} \to \mathbb{R}$ denotes an arbitrary real-valued function, not necessarily associated with any policy, Q^π denotes the value function associated with a policy, Q^* denotes the value function associated with an optimal policy, while $\hat{Q}$ denotes a (deep) parametric model. We denote the space of all such history-action real functions as $\mathcal{Q} \doteq \{Q \mid Q\colon \mathcal{H} \times \mathcal{A} \to \mathbb{R}\}$. Further, we use $g(Q)$ to denote the policy which acts greedily based on Q, i.e., if $\pi = g(Q)$, then $\pi(h) = \operatorname{argmax}_a Q(h, a)$.

3.2 DQN

Deep Q-Network (DQN) [Mnih et al., 2015] is a highly successful algorithm for training deep neural networks to control high-dimensional fully-observable Markov decision processes (MDPs) based on reward feedback, and the first to achieve human-level performance on a majority of the Atari 2600 games. Rather than relying on a lookup table to track the estimated expected return for each state-action pair (s, a), DQN learns a parametric function $\hat{Q}\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ to generalize over state-action pairs. The algorithm reformulates the incremental Q-Learning update [Watkins, 1989] as a squared-error minimization problem,

$$\mathcal{L}(s, a, r, s') \doteq \left(r + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s', a'; \theta^-) - \hat{Q}(s, a; \theta)\right)^2 , \quad (3)$$

where θ is a set of parameters and θ^- is a time-delayed copy of θ to stabilize learning. The agent interacts with the environment and stores observed transitions (s, a, r, s') in a replay memory, periodically updating θ via gradient descent on randomly sampled minibatches of experience [Lin, 1992]. This approach approximates the i.i.d. supervised training setting commonly used for neural networks and required for first-order optimization methods.

Adapting DQN to Partially Observable Control The DQN algorithm was primarily designed for fully observable control problems represented as MDPs. Nonetheless, as with many other model-free RL algorithms, generalization to partially observable control is conceptually straightforward, and achievable by replacing state variables with history variables in the relevant equations, and by employing architectures capable of processing history data. *Frame stacking*, i.e., the practice of concatenating a small number of recent observations, has been found to be sufficient to tackle problems which feature small amounts of partial observability [Mnih et al., 2015]. On the other hand, larger amounts of partial observability generally require longer-term memorization capabilities. For such problems, the standard choice has become that of combining the DQN training algorithm with a recurrent neural network component used to process history data, also known as *Deep Recurrent Q-Network* (DRQN) [Hausknecht and Stone, 2015]. Although some practitioners use the DQN label exclusively to indicate the variant which lacks a recurrent component, our view is that the essence of the DQN algorithm is in its training regime and its losses, rather than the details of which architecture is used. Therefore, in this document, we use the label DQN more broadly to encompass all architectural variants. In practice, because our work focuses on control problems which feature large amounts of partial observability, we employ appropriate algorithmic and modeling choices, i.e., all methods and baselines employ a history-based model $\hat{Q}$, and all models which receive history data employ a recurrent network component to process it.

3.3 HISTORY-STATE VALUE FUNCTIONS

Recent theoretical work in asymmetric actor-critic for PORL has employed the notion of a history-state value function $U^\pi(h, s, a)$ [Baisero and Amato, 2022, Lyu et al., 2022], which represents the expected return associated with the agent having observed history h, the environment being in state s, taking action a, and then continuing to behave according to the policy π. U^π is the unique solution to the *history-state* Bellman equation,

$$U^\pi(h, s, a) = R(s, a) + \gamma\, \mathbb{E}_{s', o|s,a}\left[U^\pi(hao, s', \pi(hao))\right] . \quad (4)$$

Despite using the state context to represent a more informed measure of the agent's expected return, U^π still relates to a partially observable agent which is unable to exploit that privileged information, i.e., the state determines future rewards, observations, and states, but it does not directly determine future actions, which are rather determined indirectly by the history. U^π is related to Q^π via a simple identity,

$$Q^\pi(h, a) = \mathbb{E}_{s|h}\left[U^\pi(h, s, a)\right] . \quad (5)$$

We denote the history-state value function associated with the optimal policy as U^*. Once again, this notion of optimality is relative to the space of partially observable policies. Among other things, this means that an optimal partially observable policy cannot be recovered by maximizing U^*, i.e., generally, there is no guarantee that $\pi^*(h) = \operatorname{argmax}_a U^*(h, s, a)$ for any given value of s.

Notation We use symbols U, U^π, U^*, and $\hat{U}$ to denote similar but separate concepts. $U\colon \mathcal{H} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes an arbitrary real-valued function, not necessarily associated with any policy, U^π denotes the value function associated with a policy, U^* denotes the value function associated with an optimal policy, while $\hat{U}$ denotes a (deep) parametric model. We denote the space of all such history-state-action real functions as $\mathcal{U} \doteq \{U \mid U\colon \mathcal{H} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}\}$.

3.4 OPERATOR NOTATION

To simplify the upcoming math, we make extensive use of operator notation for mappings between Q and U functions.

Operator $B_\pi\colon \mathcal{Q} \to \mathcal{Q}$ is the Bellman operator defined as $B_\pi Q(h,a) \doteq R(h,a) + \gamma\,\mathbb{E}_{o|h,a}\left[Q(hao, \pi(hao))\right]$, with which Equation (1) can be rewritten as $Q^\pi = B_\pi Q^\pi$.

Lemma 3.1. *Operator B_π is a contraction with fixed point Q^π (proof in Appendix A.1.)*

Operator $B\colon \mathcal{Q} \to \mathcal{Q}$ is the Bellman *optimality* operator defined as $BQ(h,a) \doteq R(h,a) + \gamma\,\mathbb{E}_{o|h,a}\left[\max_{a'} Q(hao, a')\right]$ or, equivalently, $B\colon Q \mapsto B_{g(Q)}Q$, and with which Equation (2) can be rewritten as $Q^* = BQ^*$.

Lemma 3.2. *Operator B is a contraction with fixed point Q^* (proof in Appendix A.2.)*

Some operators for U are analogous to those for Q. To avoid introducing a separate set of symbols for such cases, we overload the previously defined symbols to include these new meanings; the distinction will remain clear from context, usually as the type of the operator's input/output.

Operator $B_\pi\colon \mathcal{U} \to \mathcal{U}$ is the Bellman operator defined as $B_\pi U(h,s,a) \doteq R(s,a) + \gamma\,\mathbb{E}_{s',o|s,a}\left[U(hao, s', \pi(hao))\right]$, with which Equation (4) can be rewritten as $U^\pi = B_\pi U^\pi$.

Lemma 3.3. *Operator B_π is a contraction with fixed point U^π (proof in Appendix A.3.)*

Operator $E\colon \mathcal{U} \to \mathcal{Q}$ converts U functions to Q functions by taking the conditional expectation over states, and is defined as $EU(h,a) \doteq \mathbb{E}_{s|h}\left[U(h,s,a)\right]$.

Definition 3.4 (Mutual Consistency)**.** We say that functions Q and U are *mutually consistent* iff $Q = EU$ holds.

4 ASYMMETRIC VALUE-BASED PORL

In this section, we present the core of our theoretical and algorithmic contributions, which focus on computing or learning optimal action-values $Q^*(h,a)$ by means of asymmetry. In Section 4.1 we present *Asymmetric Policy Iteration* (API), a solution method for tabular models with optimal convergence guarantees. In Section 4.2 we present *Asymmetric Action-Value Iteration* (AAVI), an eager variant of API with similar optimal convergence guarantees. In Section 4.3 we relax aspects of AAVI to make it suitable for learning by means of sample experience, and present *Asymmetric Q-Learning* (AQL). In Section 4.4 we introduce value function approximation to improve generalization, and present *Asymmetric DQN* (ADQN), and other related variants.

Introducing Asymmetry to Value-Based Methods Two fundamental issues make the use of state information in value-based methods not directly possible: (a) because an action-value model $\hat{Q}(h,a)$ is eventually used for online control, it is constrained by the control problem and cannot directly employ privileged state information; and (b) typical value-based methods do not feature a separate model for the purpose of offline training which may access privileged information (akin to the critic in actor-critic). As such, value-based methods seem fundamentally incompatible with the notion of asymmetry and the use of privileged information. We resolve both issues, and introduce an auxiliary history-state model $\hat{U}(h,s,a)$, trained to model the optimal history-state value function $U^*(h,s,a)$, and used exclusively as a training construct through which to implement asymmetry. Our goal is to train $\hat{U}$ and $\hat{Q}$ jointly so as to converge to the optimal value functions U^* and Q^*.

4.1 ASYMMETRIC POLICY ITERATION

Consider *Asymmetric Policy Iteration* (API), an iterative process analogous to Policy Iteration [Sutton and Barto, 2018] which employs both history-state and history values to implement asymmetry. API starts from arbitrary initial values and policy U_0, Q_0, and π_0, and then uses the following update rules to generate sequences U_k, Q_k, and π_k,

$$U_{k+1} \leftarrow \lim_{n\to\infty} B^n_{\pi_k} U_k\,, \qquad \text{(U-evaluation)} \qquad (6)$$

$$Q_{k+1} \leftarrow EU_{k+1}\,, \qquad \text{(Q-evaluation)} \qquad (7)$$

$$\pi_{k+1} \leftarrow g(Q_{k+1})\,. \qquad \text{(improvement)} \qquad (8)$$

The U-evaluation step can be practically implemented as the solution to the system of equations $U_{k+1} = R + \gamma P_{\pi_k} U_{k+1}$, or by using B_{π_k} until convergence (see Algorithm 1).

Theorem 4.1 (API Optimality)**.** *The sequences U_k, Q_k, and π_k generated by API converge to U^*, Q^*, and π^*.*

Proof. By Lemma 3.1, U_{k+1} equals the fixed point of B_{π_k}, i.e., $U_{k+1} = U^{\pi_k}$. Then, by Equation (5), $Q_{k+1} = EU^{\pi_k} = Q^{\pi_k}$ and consequently $\pi_{k+1} = g(Q^{\pi_k})$. Therefore, in each iteration and until π^* is reached, the next policy π_{k+1} is a strict improvement on the previous policy π_k (Policy Improvement Theorem, [Sutton and Barto, 2018]). Let k^* be the smallest index such that π_{k^*} is optimal; for $k > k^*$, we conclude that $U_k = U^*$ and $Q_k = Q^*$. □

Algorithm 1 Asymmetric Policy Iteration (API)

```
Require: $U_0$, $Q_0$, $\pi_0$ arbitrarily initialized tabular models.
Ensure: $\lim_{k\to\infty}\{U_k, Q_k, \pi_k\} = \{U^*, Q^*, \pi^*\}$.
1: for $k \leftarrow 0, 1, 2, 3, \ldots$ do
2:     $U_{k+1} \leftarrow U_k$
3:     repeat
4:         $U_{k+1} \leftarrow B_{\pi_k} U_{k+1}$
5:     until convergence
6:     $Q_{k+1} \leftarrow EU_{k+1}$
7:     $\pi_{k+1} \leftarrow g(Q_{k+1})$
8: end for
```

Limitations While API is formally guaranteed to converge optimally, it also has significant practical limitations: (a) API is a solution method which requires a model of the environment, as well as efficient and accurate methods to compute the expectations in the U-evaluation and the Q-evaluation steps. (b) A practical approximation of the limit operator in the U-evaluation step (see Algorithm 1) might itself require multiple iterations to achieve an adequate precision. (c) API requires tabular models U and Q, which is not only impractical given that the space of histories grows exponentially with episode lengths, but also makes it not applicable to control problems which have continuous observations or states. (d) Perhaps most importantly, API does not offer any significant advantage compared to its non-asymmetric counterpart PI. Ultimately, both API and PI converge to the same optimal value function Q^*; if anything, API requires more memory and computation to achieve the same goal, resulting in a less practical solution method.

Why API? In light of the above limitations, particularly the last one, what is then the purpose of API? We argue that API plays two crucial roles: (a) The first is to show that privileged and asymmetric information such as the system state *can* be properly included into a value-based solution process while maintaining formal optimality guarantees. This theoretical aspect is often overlooked in modern asymmetric RL research, which instead tends to focus on heuristic methods and empirical results, and API represents the first theoretical guarantee of this kind for value-based RL. (b) The second is to serve as a basis for other algorithms which do provide practical advantages compared to their non-asymmetric counterparts. Starting from the next subsection, we relax various aspects of API and develop asymmetric value-based algorithms which address each of API's limitations.

4.2 ASYMMETRIC ACTION-VALUE ITERATION

The first limitation of API which we address is the presence of the limiting operator in its U-step, which makes practical implementations approximate, and/or inefficient. To this end, consider *Asymmetric Action-Value Iteration* (AAVI), an eager variant of API which uses the following updates,

Algorithm 2 Asymmetric Action-Value Iteration (AAVI)

```
Require: $U_0$, $Q_0$ arbitrarily initialized tabular models.
Ensure: $\lim_{k\to\infty}\{U_k, Q_k\} = \{U^*, Q^*\}$.
1: for $k \leftarrow 0, 1, 2, 3, \ldots$ do
2:     $U_{k+1} \leftarrow B_{g(Q_k)} U_k$
3:     $Q_{k+1} \leftarrow EU_{k+1}$
4: end for
```

$$U_{k+1} \leftarrow B_{g(Q_k)} U_k \,, \qquad \text{(U-evaluation)} \qquad (9)$$
$$Q_{k+1} \leftarrow EU_{k+1} \,. \qquad \text{(Q-evaluation)} \qquad (10)$$

Compared to API, the improvement step has been folded in the U-evaluation step, removing the need for an explicit policy representation. Further, the U-evaluation step has been simplified to apply operator $B_{g(Q_k)}$ a single time, making for a simple, faster, and more practical implementation (see Algorithm 2) without compromising optimality guarantees. Both aspects make AAVI analogous to Value Iteration [Sutton and Barto, 2018], with the primary differences being the use of action-values and asymmetry.

Lemma 4.2 (Asymmetric Bellman Equivalence)**.** *For mutually consistent U and Q, the identity $EB_{g(Q)}U = BQ$ holds (proof in Appendix A.4.)*

Theorem 4.3 (AAVI Optimality)**.** *The sequences U_k and Q_k generated by AAVI converge to U^* and Q^*.*

Proof. We can combine the U-evaluation and Q-evaluation steps, and then use Lemma 4.2 to obtain $Q_{k+1} = EU_{k+1} = EB_{g(Q_k)}U_k = BQ_k$. By induction, $Q_k = B^k Q_0$, which converges to the fixed point of B: i.e., $\lim_{k\to\infty} Q_k = Q^*$. This guarantees the existence of some iteration k^* such that $g(Q_k) = \pi^*, \forall k \geq k^*$. Therefore, $U_k = B_{\pi^*}^{k-k^*} U_{k^*}, \forall k \geq k^*$, and U_k converges to the fixed point of B_{π^*}: i.e., $\lim_{k\to\infty} U_k = U^*$. □

4.3 ASYMMETRIC Q-LEARNING

Like all dynamic programming methods, API and AAVI make extensive and often unrealistic assumptions like the model of the environment and being able to compute exact expectations. To bypass many of these requirements, we can employ incremental stochastic updates based on sequentially sampled transitions. We call this new method *Asymmetric Q-Learning* (AQL), as it generalizes the iterative Q-Learning algorithm [Watkins, 1989] to asymmetric PORL.

To handle the randomness induced by the sample transitions, the algorithm must average over the samples using a variable stepsize parameter $\alpha_k \in [0, 1]$. At each iteration k, the agent

Algorithm 3 Asymmetric Q-Learning (AQL)

```
Require: $U$, $Q$ mutually consistent tabular models.
Ensure: $\{U, Q\} \to \{U^*, Q^*\}$.
 1: while True do
 2:     Initialize history and state $(h, s)$
 3:     while $s$ is not terminal do
 4:         Choose action $a$ from $\epsilon$-greedy policy on $Q$
 5:         Take action $a$, observe $r, s', o$
 6:         $y \leftarrow r + \gamma U(hao, s', \mathrm{argmax}_{a'} Q(hao, a'))$
 7:         $U(h, s, a) \leftarrow (1 - \alpha)U(h, s, a) + \alpha y$
 8:         $Q(h, a) \leftarrow (1 - \alpha)Q(h, a) + \alpha y$
 9:         $(s, h) \leftarrow (s', hao)$
10:     end while
11: end while
```

Algorithm 4 Asymmetric DQN (ADQN)

```
Require: $\hat{U}$, $\hat{Q}$ deep models parameterized by $\theta$.
 1: Initialize parameters $\theta$
 2: Initialize and prepopulate episode buffer
 3: while True do
 4:     From the simulated environment, sample episodes
        and append them to the episode buffer
 5:     From the episode buffer, sample batch of transitions
        $\{(h_i, s_i, a_i, r_i, s'_i, o_i)\}_{i=1}^N$
 6:     $L_U \leftarrow \frac{1}{N} \sum_{i=1}^N \mathcal{L}_{\hat{U}}(h_i, s_i, a_i, r_i, s'_i, o_i)$.
 7:     $L_Q \leftarrow \frac{1}{N} \sum_{i=1}^N \mathcal{L}_{\hat{Q}}(h_i, s_i, a_i, r_i, s'_i, o_i)$.
 8:     Perform a gradient step on $\theta$ using $\nabla_\theta(L_U + L_Q)$
 9: end while
```

samples a transition $(h_k, s_k, a_k, r_k, s_{k+1}, o_k)$ and conducts AAVI-like updates[1] on the respective entries of U_k and Q_k.

For notational brevity, we first define the following targets:

$$Y_k(h, s, a) \doteq \begin{cases} \left(B_{g(Q_k)}U_k + w_k\right)(h, s, a) & \text{for } (h_k, s_k, a_k) \\ U_k(h, s, a) & \text{otherwise} \end{cases}, \tag{11}$$

$$Z_k(h, a) \doteq \begin{cases} \left(EB_{g(Q_k)}U_k + v_k\right)(h, a) & \text{for } (h_k, a_k) \\ Q_k(h, a) & \text{otherwise} \end{cases}. \tag{12}$$

Here, $w_k \in \mathcal{U}$ and $v_k \in \mathcal{Q}$ are zero-mean noise processes that represent the randomness in the environment and action selection at iteration k. AQL then conducts the following updates based on the stochastic targets Y_k and Z_k:

$$U_{k+1} \leftarrow U_k + \alpha_k(Y_k - U_k)\,, \tag{13}$$
$$Q_{k+1} \leftarrow Q_k + \alpha_k(Z_k - Q_k)\Pr(s_k \mid h_k)\,. \tag{14}$$

Note that the targets Y_k and Z_k are defined elementwise such that only one entry of U_k and Q_k—the one associated with (h_k, s_k, a_k)—is updated for any given index k. When the stepsizes α_k are annealed towards zero at an appropriate rate, AQL converges optimally despite the noisy updates.

Theorem 4.4 (AQL Optimality)**.** *Assume stepsizes α_k satisfying the following asymptotic conditions,*

$$\sum_{k=0}^\infty \alpha_k = \infty\,, \qquad \sum_{k=0}^\infty \alpha_k^2 < \infty\,. \tag{15}$$

If Q_0, U_0 are mutually consistent ($Q_0 = EU_0$), then the sequences Q_k and U_k generated by AQL converge to Q^* and U^* with probability 1 (proof in Appendix A.5.)

[1]The Q-evaluation step of AAVI (Equation (10)) can equivalently be expressed as $Q_{k+1} \leftarrow EB_{g(Q_k)}U_k$, which is the form AQL employs to guarantee optimal convergence.

Factor $\Pr(s_k \mid h_k)$ is necessary to ensure that U_k and Q_k remain mutually consistent throughout the process; a necessary condition for optimal convergence. $\Pr(s_k \mid h_k)$ can be interpreted as a scaling factor which makes U_k and Q_k update at relatively comparable rates. For any given "full" update on $U_k(h_k, s_k, a_k)$ the corresponding update on $Q_k(h_k, a_k)$ should be scaled down to a "partial amount" relative to the likelihood of s_k. While we were able to remove other forms of model-based requirements, $\Pr(s_k \mid h_k)$ remains, leaving AQL just shy from reaching both optimal convergence and concrete practicality at the same time. While it may be possible to approximate this factor in other model-free ways, AQL remains primarily a conceptual algorithm also due to the requirement of a tabular model over histories. Either way, AQL serves as a fundamental basis to derive the next iteration of asymmetric value-based RL.

4.4 ASYMMETRIC DQN

When acting in POMDPs with high-dimensional observations and states, a tabular-lookup method such as AQL becomes infeasible. In such instances, we must introduce function approximation to generalize over similar experiences. The use of approximation sacrifices the optimal convergence guarantee established by Theorem 4.4, but is necessary to scale algorithms to significantly more challenging partially observable environments. Nevertheless, the value function approximations are an orthogonal matter to how privileged state information is used, and we expect the sound theoretical principles upon which AQL is built will help asymmetric deep methods even when relying on function approximation.

Our primary algorithmic contribution here is *Asymmetric DQN* (ADQN), an asymmetric variant of DQN derived by introducing value function approximation to AQL. We first replace the tabular-lookup models U and Q of AQL with parametric differentiable models $\hat{U}$ and $\hat{Q}$. In practice, these are implemented as deep neural networks whose architectures are chosen according to the structure of the states and observations emitted by the POMDP.

To facilitate the substitution, we must reformulate the stochastic update rules of AQL as squared-error loss minimization. For the rest of the section, due to spacing concerns, we will use $\hat{\pi} = g(\hat{Q})$ as a shorthand to represent actions selected greedily on $\hat{Q}$, i.e., $\hat{\pi}(h) = \operatorname{argmax}_a \hat{Q}(h, a)$. Given a single environment interaction (h, s, a, r, s', o), the corresponding losses can be defined as

$$\mathcal{L}_{\hat{U}} = \left(r + \gamma \operatorname{SG}\left[\hat{U}(hao, s', \hat{\pi}(hao))\right] - \hat{U}(h, s, a)\right)^2 , \tag{16}$$

$$\mathcal{L}_{\hat{Q}} = \left(r + \gamma \operatorname{SG}\left[\hat{U}(hao, s', \hat{\pi}(hao))\right] - \hat{Q}(h, a)\right)^2 , \tag{17}$$

where SG is the *stop-gradient* function which indicates that gradient calculation should not consider the enclosed terms. It is worth noting that $\mathcal{L}_{\hat{U}}$ and $\mathcal{L}_{\hat{Q}}$ use the same target to train $\hat{U}$ and $\hat{Q}$. The crucial difference is that $\hat{U}$ is in able to model the target as a function of s, while $\hat{Q}$ is unable to do so, and can at only model the expectation of the target over values of s. In a way, these losses approximately enforce a "loose" form of mutual consistency $\hat{Q} \approx E\hat{U}$. In practice, the term is generated by "target networks" that rely on stale parameters to stabilize learning when bootstrapping [Mnih et al., 2015]; the stale parameters are periodically updated by copying the main parameters. The total loss $\mathcal{L}_{\hat{U}} + \mathcal{L}_{\hat{Q}}$ can be jointly minimized with respect to the parameters by a single backpropagation step, efficiently approximating the interleaved updates of AQL.

When the function approximators are nonlinear (as is often the case for neural networks), training will fail if the gradient updates are conducted on sequentially collected transitions that are not i.i.d. [Mnih et al., 2015]. The second critical modification to AQL is therefore the adoption of experience replay [Lin, 1992] in order to decorrelate training experiences. Rather than training on a sample immediately when it is collected, each POMDP transition (h, s, a, r, s', o) is deferred to a first-in first-out replay memory. Periodically, when it is time to train the networks, a minibatch of several experiences is sampled from the replay memory; gradients for these samples are computed and averaged together to estimate the true gradient of the joint loss $\mathcal{L}_{\hat{U}} + \mathcal{L}_{\hat{Q}}$, which in turn is used to improve the parameters.

Why ADQN? Having finally addressed the practical disadvantages of API, it is worthwhile to reconsider the "why" question again, this time focusing on why one would prefer to use ADQN compared to DQN. Ultimately, the purpose of both algorithms is to train an approximate $\hat{Q} \approx Q^*$ through which optimal control can be executed, and both algorithms should *in theory* converge to very similar approximations. What is then the advantage of ADQN over DQN? Similarly to the asymmetric actor-critic case [Baisero and Amato, 2022], the advantage is a practical one associated with the difficulties of learning an appropriate representation of history $\phi(h)$, which is one of the major bottlenecks in PORL. History representations are sequence models which notoriously requires lots of data and processing power for proper training. To further compound on this issue, the quality of the data used to train the history representation in PORL is directly related to the quality of $\hat{Q}(\phi(h), \cdot, \cdot)$, which in turn depends on the quality of the history representation itself; unsurprisingly, it can be quite hard to bootstrap the training of an improved history representation when starting from a poor history representation. Note, however, that learning an appropriate state representation $\phi(s)$ is much simpler than learning $\phi(h)$ due to the non-sequential nature of individual states, i.e., the $\phi(s)$ representation model has fixed input and output sizes, and can generally be modeled using a simpler feed-forward architecture. In ADQN, the issues associated with learning a proper history representation are alleviated by the fact that its training is bootstrapped not only on the history representation itself, but also on the state representation. Even when the history representation is poor, we can expect the state representation to contain sufficient contextual information to allow $\hat{U}(\phi(h), \phi(s), \cdot)$ to model meaningful values, which in turns helps further bootstrap the learning of the history representation $\phi(h)$, the history model $\hat{Q}(\phi(h), \cdot)$, and the respective implicit policy $g(\hat{Q})$.

Next, we consider some variants of interest of ADQN.

4.4.1 Variance-Reduced ADQN

In this variant, we approximate the target of $\mathcal{L}_{\hat{Q}}$ from Equation (17) as $r + \gamma\hat{U}(hao, s', \hat{\pi}(hao)) \approx \hat{U}(h, s, a)$, which holds particularly well once $\hat{U}$ has been trained sufficiently. Therefore, this variant uses the following losses,

$$\mathcal{L}_{\hat{U}} = \left(r + \gamma \operatorname{SG}\left[\hat{U}(hao, s', \hat{\pi}(hao))\right] - \hat{U}(h, s, a)\right)^2 , \tag{18}$$

$$\mathcal{L}_{\hat{Q}} = \left(\operatorname{SG}\left[\hat{U}(h, s, a)\right] - \hat{Q}(h, a)\right)^2 . \tag{19}$$

This approximate target results in lower variance throughout the entire training process at the cost of introducing bias primarily in the early stages of training; a trade-off which may result advantageous in some control problems.

4.4.2 State-Only ADQN

Some prior work in asymmetric RL has adopted heuristic forms of asymmetry which uses state-only (i.e., history-less) value functions $U(s, a)$. Such form of asymmetry is however associated with fundamental theoretical issues which may severely compromise the learning performance, ranging from potentially being ill-defined, to introducing bias into the learning process [Baisero and Amato, 2022]. Such issues are inherently related to partial observability, and their effects scale with the amount of partial observability that the agent is subject to, as well as the agent's "reactiveness", i.e. the amount of history that is willfully ignored by the agent itself to select actions. Nonetheless, this state-only form of value function may be more useful than others in control problems which have small amounts of partial observability, such as vision-based tasks with an occlusion-free view of the environment [Pinto et al., 2018, Baisero and Amato, 2022]. Although our main focus is control problems with significant amounts of partial observability, we are still interested in formulating a state-only variant of ADQN as an additional baseline, and as reference for future work which may focus on the kinds of control problems where it thrives.

In this state-only variant, we redefine the parametric model $\hat{U}(s,a)$ to ignore history, and adopt the following losses,

$$\mathcal{L}_{\hat{U}} = \left(r + \gamma \operatorname{SG}[\hat{U}(s', \hat{\pi}(hao))] - \hat{U}(s,a)\right)^2 , \quad (20)$$

$$\mathcal{L}_{\hat{Q}} = \left(r + \gamma \operatorname{SG}[\hat{U}(s', \hat{\pi}(hao))] - \hat{Q}(h,a)\right)^2 . \quad (21)$$

4.4.3 Reduced-Variance State-Only ADQN

This variant applies a state-only variant of the variance reduction approximation from Section 4.4.1 to state-only ADQN. In this case, we approximate the target of $\mathcal{L}_{\hat{Q}}$ from Equation (21) as $r + \gamma\hat{U}(s', \hat{\pi}(hao)) \approx \hat{U}(s,a)$,

$$\mathcal{L}_{\hat{U}} = \left(r + \gamma \operatorname{SG}[\hat{U}(s', \pi(hao))] - \hat{U}(s,a)\right)^2 , \quad (22)$$

$$\mathcal{L}_{\hat{Q}} = \left(\operatorname{SG}[\hat{U}(s,a)] - \hat{Q}(h,a)\right)^2 . \quad (23)$$

5 EVALUATION

We perform an empirical evaluation of our proposed ADQN method and its variants in a variety of environments which feature significant amounts of partial observability.

Methods We compare the performances of 5 value-based PORL algorithms, denoted as follows:

- **DQN** is the standard non-asymmetric DQN algorithm;
- **ADQN** and **ADQN-VR** are the history-state ADQN algorithms from Sections 4.4 and 4.4.1; and
- **ADQN-State** and **ADQN-State-VR** are the state-only ADQN algorithms from Sections 4.4.2 and 4.4.3.

Environments Evaluations are run on 5 partially observable navigation tasks which require information gathering strategies and memorization of the past:

- **Heaven-Hell-3** and **Heaven-Hell-4** [Bonet, 1998], corridor environments where the agent must reach the exit to *heaven* and avoid the exit to *hell*, but must first backtrack to visit a *priest* to learn which exit is which;
- **Car-Flag** [Nguyen, 2021], a 1-dimensional continuous control variant of **Heaven-Hell**;
- **Cleaner** [Jiang and Amato, 2021], a maze environment where two agents must reach all tiles to clean them. In our experiments, the two agents are treated as a single agent, and controlled in a centralized fashion; and
- **GV-MemoryFourRooms-7x7** [Baisero and Katt, 2021], a dynamically generated gridworld with 4 connected rooms, where the agent must reach the *good* exit and avoid the *bad* exit, but must first find and memorize a *beacon* to learn which is which.

A more thorough description of these environments can be found in Appendix C of Baisero and Amato [2022].

Each method is trained and evaluated using code available as a public repository[2]. For each environment and algorithm, we perform an independent grid-search over some hyper-parameters of interest (see Appendix D), and select the combination of hyper-parameters which results in the best final performance and learning stability (prioritizing final performance if necessary). To improve the statistical significance of the results, each combination of environment, algorithm, and hyper-parameters is run 20 independent times.

5.1 RESULTS AND DISCUSSION

Figure 1 shows the results of these evaluations, which broadly confirm our theoretical analysis on asymmetric value-based PORL, the practical advantage of employing history-state forms of evaluation to aid partially observable control, and confirm the superiority of ADQN compared to other similar symmetric and asymmetric variants. This evaluation further confirms other recently developed theoretical analysis on asymmetric PORL, i.e., that state-only forms of asymmetry are inadequate to handle non-trivial amounts of partial observability [Baisero and Amato, 2022].

Across the board, **ADQN** and **ADQN-VR** outperform all baselines in final performance, convergence speed, and/or overall learning stability. The contrast between methods is particularly stark in Figures 1a and 1b, where **ADQN** and **ADQN-VR** are not just the only methods that demonstrate any substantial improvement, but are also able to reach optimal performance. On the other hand, the state-only variants fail to outperform even the **DQN** baseline in most environments (with a single exception discussed later), which further confirms the theoretical issues that have been recently associated with state-only forms of asymmetry. Broadly, the variance-reduced variants **ADQN-VR** and **ADQN-State-VR** only differ in relatively minor ways from their default counterparts. Such differences can be found in Figures 1a and 1d, where **ADQN** is more stable or has better convergence properties than **ADQN-State-VR**, and in Figures 1c and 1d, where **ADQN-State-VR** has better final convergence values than **ADQN-State**. This seems to indicate

[2]`https://github.com/abaisero/asym-rlpo/`

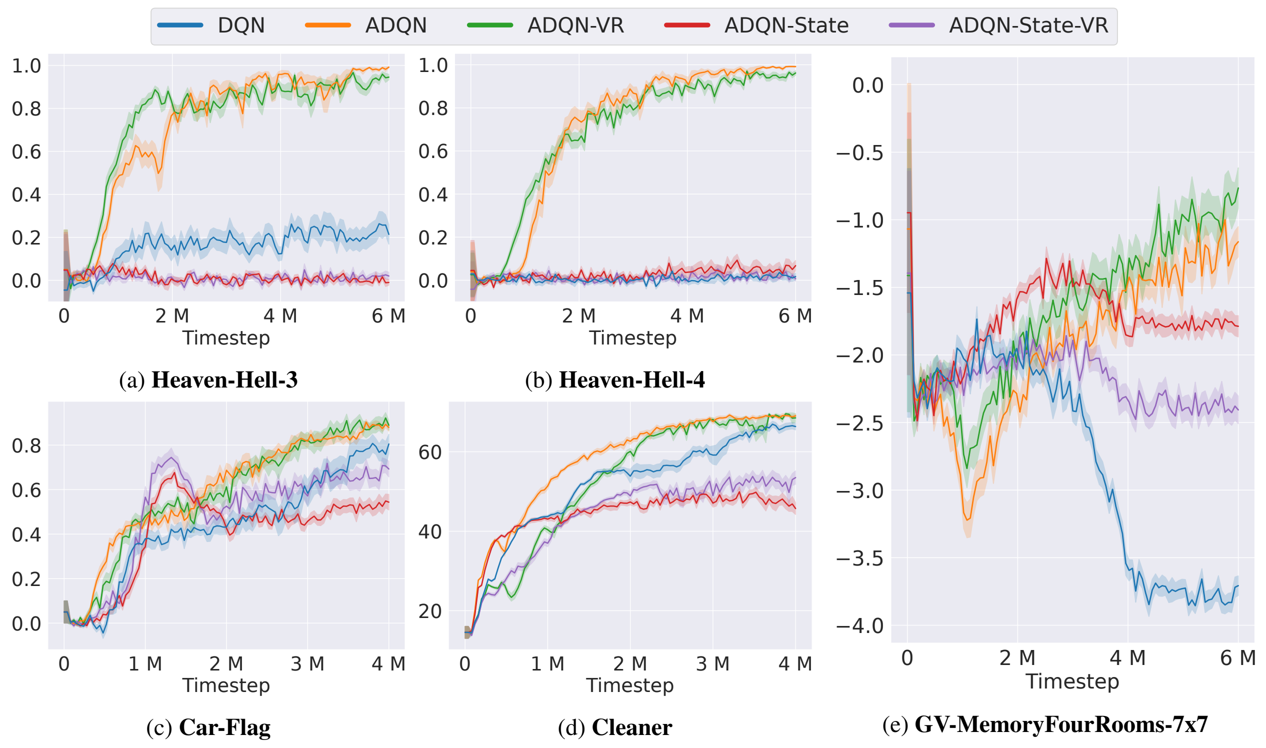


Figure 1: Performance curves showing episodic returns averaged over the last 100 completed episodes, with statistics computed over 20 independent runs. The shaded areas represent one standard error around the mean.

that the type of asymmetry (history-state or state-only) is a larger contributor to overall performance than the choice of using the standard or the variance-reduced variant of the same method. In Figure 1e too, **ADQN** and **ADQN-VR** outperform all other baselines. However, these results also represent an interesting exception to some of the above analysis, e.g., **ADQN-State** outperforms **DQN** and **ADQN-VR** outperforms **ADQN**. To explain this, we note that this is the only task not reliably solved by any of the methods, which is likely due to the highly dynamic nature of the randomly generated map and object locations. In fact, the fact that some of the trends found in the other results do not also appear here may be explained by the fact that none of the methods achieve their full potential performance.

6 CONCLUSIONS

OTOE is a RL framework where agents are trained offline in a simulated environment, which allows temporary access to privileged information which would otherwise be unavailable, like the partially observable environment's state. Asymmetry is a common mechanism through which such privileged information can be used during training, and has the potential to greatly boost learning performance and efficiency when implemented correctly. However, modern work in asymmetric RL tends to focus on unproven heuristics which lack a theoretical justification. In this work, we filled this void and developed the theory of asymmetric value-based RL. We achieved our primary goal of developing a theoretically-sound asymmetric value-based RL algorithm by employing a bottom-up approach, and by first focusing on the base theory of asymmetric policy improvement. This took the form of API, a conceptual solution method with strict optimal convergence guarantees but concrete practical limitations. Then, we applied a series of relaxations to API which addressed those limitations and ultimately resulted in ADQN, a practical and competitive deep RL algorithm. We performed an empirical evaluation to compare the performances of ADQN and its variants to standard non-asymmetric DQN in a series of environments which are specifically selected to exhibit high levels of partial observability, and which require information-gathering strategies and memorization of the past. In all these environments, ADQN achieved the best performance, even solving control problems which standard DQN could not. Overall, our evaluation confirmed the potential offered by privileged information, the importance of using it in principled and theoretically-guided ways, and the overall success our ADQN algorithm in partially observable control problems. Future work may focus on extending ADQN to the multi-agent control case, which poses further learning challenges, on finding applications where state-only ADQN may thrive (such as vision-based robotic tasks with little partial observability), and on extending the evaluation of ADQN in more complicated partially observable vision-based tasks.

Author Contributions

Andrea Baisero conceived the idea, developed proofs, ran experiments, and wrote the paper. Brett Daley developed proofs and wrote the paper. Christopher Amato supervised.

Acknowledgements

This research was funded by NSF awards 1816382 and 2024790.

References

Andrea Baisero and Christopher Amato. Unbiased Asymmetric Reinforcement Learning under Partial Observability. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pages 44–52, 2022.

Andrea Baisero and Sammie Katt. gym-gridverse: Gridworld domains for fully and partially observable reinforcement learning, 2021. URL `https://github.com/abaisero/gym-gridverse`.

Dimitri P. Bertsekas and John N. Tsitsiklis. Neuro-dynamic programming: an overview. In *Proceedings of 1995 34th IEEE conference on decision and control*, volume 1, pages 560–564. IEEE, 1995.

Blai Bonet. Solving large POMDPs using real time dynamic programming. In *Proc. AAAI Fall Symp. on POMDPs*, 1998.

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Christian Schroeder de Witt, Bei Peng, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Böhmer, and Shimon Whiteson. Deep Multi-Agent Reinforcement Learning for Decentralized Continuous Cooperative Control. *arXiv preprint arXiv:2003.06709*, 2020.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

Shuo Jiang and Christopher Amato. Multi-agent reinforcement learning with directed exploration and selective memory reuse. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 777–784, 2021.

Shihui Li, Yi Wu, Xinyue Cui, Honghua Dong, Fei Fang, and Stuart Russell. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4213–4220, 2019.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3):293–321, 1992.

Ryan Lowe, Yi I. Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, pages 6379–6390, 2017.

Xueguang Lyu, Andrea Baisero, Yuchen Xiao, and Christopher Amato. A Deeper Understanding of State-Based Critics in Multi-Agent Reinforcement Learning. In *Proceedings of the AAAI conference on artificial intelligence*, 2022.

Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. In *Advances in Neural Information Processing Systems*, pages 7613–7624, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, and Andreas K. Fidjeland. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Hai Nguyen. POMDP Robot Domains, 2021. URL `https://github.com/hai-h-nguyen/pomdp-domains`.

Hai Nguyen, Brett Daley, Xinchao Song, Christopher Amato, and Robert Platt. Belief-Grounded Networks for Accelerated Robot Learning under Partial Observability. In *Conference on Robot Learning*, pages 1640–1653. PMLR, 2021.

Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric Actor Critic for Image-Based Robot Learning. In *Proceedings of Robotics: Science and Systems*, 2018.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International conference on machine learning*, pages 4295–4304. PMLR, 2018.

Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 33:10199–10210, 2020.

Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Rose E. Wang, Michael Everett, and Jonathan P. How. R-maddpg for partially observable environments and limited communication. *arXiv preprint arXiv:2002.06684*, 2020.

Andrew Warrington, Jonathan W. Lavington, Adam Scibior, Mark Schmidt, and Frank Wood. Robust asymmetric learning in pomdps. In *International Conference on Machine Learning*, pages 11013–11023. PMLR, 2021.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD Thesis, King's College, Cambridge, 1989.

Yuchen Xiao, Joshua Hoffman, Tian Xia, and Christopher Amato. Learning multi-robot decentralized macro-action-based policies via a centralized Q-net. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10695–10701. IEEE, 2020.

Yuchen Xiao, Xueguang Lyu, and Christopher Amato. Local Advantage Actor-Critic for Robust Multi-Agent Deep Reinforcement Learning. In *2021 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 155–163. IEEE, 2021.

Jiachen Yang, Alireza Nakhaei, David Isele, Kikuo Fujimura, and Hongyuan Zha. Cm3: Cooperative multi-goal multi-stage multi-agent reinforcement learning. *arXiv preprint arXiv:1809.05188*, 2018.

A PROOFS

A.1 PROOF OF LEMMA 3.1

We first note that B_π may also be expressed as $B_\pi \colon Q \mapsto R + \gamma P_\pi Q$, where $P_\pi \colon \mathcal{Q} \to \mathcal{Q}$ is the linear operator defined by $P_\pi Q(h,a) \doteq \mathbb{E}_{o|h,a}\left[Q(hao, \pi(hao))\right]$, and has operator ∞-norm $\|P_\pi\| = 1$. Next, we show that B_π is a γ-contraction in ∞-norm, i.e., for any two $Q, Q' \in \mathcal{Q}$, the following inequality holds:

$$\begin{aligned}
\|B_\pi Q - B_\pi Q'\| &= \|R + \gamma P_\pi Q - R - \gamma P_\pi Q'\| \\
&= \gamma \|P_\pi \left(Q - Q'\right)\| \\
&\leq \gamma \|P_\pi\| \|Q - Q'\| \\
&= \gamma \|Q - Q'\| \,.
\end{aligned} \tag{24}$$

Therefore, we conclude that B_π has a unique fixed point which is Q^π by definition.

A.2 PROOF OF LEMMA 3.2

Proof. We show that B is a γ-contraction in ∞-norm, i.e., for any two $Q, Q' \in \mathcal{Q}$, the following inequality holds:

$$\begin{aligned}
\|BQ - BQ'\| &= \max_{h,a} \left| R(h,a) + \gamma\, \mathbb{E}_{o|h,a} \max_{a'} Q(hao,a') - R(h,a) - \gamma\, \mathbb{E}_{o|h,a} \max_{a'} Q'(hao,a') \right| \\
&= \gamma \max_{h,a} \left| \mathbb{E}_{o|h,a}\left[\max_{a'} Q(hao,a')\right] - \mathbb{E}_{o|h,a}\left[\max_{a'} Q'(hao,a')\right] \right| \\
&= \gamma \max_{h,a} \left| \mathbb{E}_{o|h,a}\left[\max_{a'} Q(hao,a') - \max_{a'} Q'(hao,a')\right] \right| \\
&\leq \gamma \max_{h,a} \mathbb{E}_{o|h,a}\left[\left|\max_{a'} Q(hao,a') - \max_{a'} Q'(hao,a')\right|\right] \\
&\leq \gamma \max_{h,a,o} \left|\max_{a'} Q(hao,a') - \max_{a'} Q'(hao,a')\right| \\
&\leq \gamma \max_{h,a,o,a'} \left|Q(hao,a') - Q'(hao,a')\right| \\
&= \gamma \|Q - Q'\| \,.
\end{aligned} \tag{25}$$

Therefore, we conclude that B has a unique fixed point which is Q^* by definition. □

A.3 PROOF OF LEMMA 3.3

Proof. We first note that B_π may also be expressed as $B_\pi \colon U \mapsto R + \gamma P_\pi U$, where $P_\pi \colon \mathcal{U} \to \mathcal{U}$ is the linear operator defined by $P_\pi U(h,s,a) \doteq \mathbb{E}_{s',o|s,a}\left[U(hao, s', \pi(hao))\right]$, and has operator ∞-norm $\|P_\pi\| = 1$. Next, we show that B_π is a γ-contraction in ∞-norm, i.e., for any two $U, U' \in \mathcal{U}$, the following inequality holds:

$$\begin{aligned}
\|B_\pi U - B_\pi U'\| &= \|R + \gamma P_\pi U - R - \gamma P_\pi U'\| \\
&= \gamma \|P_\pi \left(U - U'\right)\| \\
&\leq \gamma \|P_\pi\| \|U - U'\| \\
&= \gamma \|U - U'\| \,.
\end{aligned} \tag{26}$$

Therefore, we conclude that B_π has a unique fixed point which is U^π by definition. □

A.4 PROOF OF LEMMA 4.2 (ASYMMETRIC BELLMAN EQUIVALENCE)

Proof. We assume $Q = EU$, and show that the following elementwise identity holds,

$$\begin{aligned}
(EB_{g(Q)}U)(h,a) &= \mathbb{E}_{s|h}\left[R(s,a) + \gamma\,\mathbb{E}_{s',o|s,a}\left[U(hao, s', \operatorname*{argmax}_{a'} Q(hao,a'))\right]\right] \\
&= R(h,a) + \gamma\,\mathbb{E}_{s|h}\left[\mathbb{E}_{s',o|s,a}\left[U(hao, s', \operatorname*{argmax}_{a'} Q(hao,a'))\right]\right] \\
&= R(h,a) + \gamma\,\mathbb{E}_{s',o|h,a}\left[U(hao, s', \operatorname*{argmax}_{a'} Q(hao,a'))\right] \\
&= R(h,a) + \gamma\,\mathbb{E}_{o|h,a}\left[\mathbb{E}_{s'|hao}\left[U(hao, s', \operatorname*{argmax}_{a'} Q(hao,a'))\right]\right] \\
&= R(h,a) + \gamma\,\mathbb{E}_{o|h,a}\left[Q(hao, \operatorname*{argmax}_{a'} Q(hao,a'))\right] \\
&= R(h,a) + \gamma\,\mathbb{E}_{o|h,a}\left[\max_{a'} Q(hao,a')\right] \\
&= BQ(h,a)\,.
\end{aligned} \tag{27}$$

Therefore, $EB_{g(Q)}U = BQ$. □

A.5 PROOF OF THEOREM 4.4 (AQL OPTIMALITY)

Proof. Let (h_k, s_k, a_k) denote the history, state, and action visited at the k-th iteration of the AQL algorithm. To facilitate our analysis, we would like to remove the explicit conditional updates in Equations (13) and (14). We define the binary random process $\chi_k \in \mathcal{U}$ such that

$$\chi_k(h,s,a) = \begin{cases} 1 & \text{if } (h,s,a) = (h_k,s_k,a_k) \\ 0 & \text{otherwise} \end{cases}\,. \tag{28}$$

Using elementwise multiplication and division, the AQL updates in Equations (13) and (14) can be written as

$$U_{k+1} \leftarrow U_k + \alpha_k \chi_k (B_{g(Q_k)}U_k + w_k - U_k)\,, \tag{29}$$

$$Q_{k+1} \leftarrow Q_k + \alpha_k (E\chi_k)(EB_{g(Q_k)}U_k + v_k - Q_k)\,. \tag{30}$$

We note that the noise processes w_k and v_k are not statistically independent because they are computed using a shared transition, which guarantees that $v_k = Ew_k$. This allows us to prove that U_k and Q_k remain mutually consistent after each update, i.e., $Q_k = EU_k$, which we show by induction. From the assumed initialization in the theorem, the base case $Q_0 = EU_0$ is satisfied. Assume that $Q_k = EU_k$ holds for the inductive hypothesis. It follows that

$$\begin{aligned}
EU_{k+1} &= E(U_k + \alpha_k\chi_k(B_{g(Q_k)}U_k + w_k - U_k)) \\
&= EU_k + \alpha_k(E\chi_k)(EB_{g(Q_k)}U_k + Ew_k - EU_k)) \\
&= Q_k + \alpha_k(E\chi_k)(EB_{g(Q_k)}U_k + v_k - Q_k) \\
&= Q_{k+1}\,,
\end{aligned} \tag{31}$$

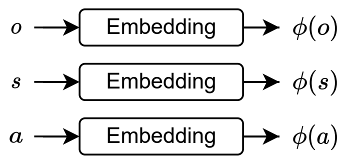
and therefore U_k and Q_k are mutually consistent for all $k \geq 0$. By Lemma 4.2, Equation (30) reduces to

$$Q_{k+1} = Q_k + \alpha_k(E\chi_k)(BQ_k + v_k - Q_k)\,. \tag{32}$$

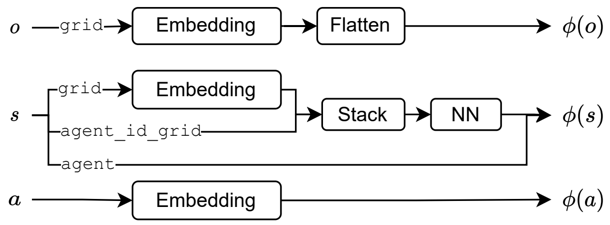
Now let $p_k(h,s,a) \in [0,1]$ be the probability that $(h,s,a) = (h_k,s_k,a_k)$ conditioned on iteration $k-1$. Additionally, let $\mathbf{1}$ denote vectors whose components are all equal to one. We can equivalently express Equations (13) and (14) in the form

$$U_{k+1} = (\mathbf{1} - \alpha_k p_k)U_k + \alpha_k p_k (B_{g(Q_k)}U_k + w'_k)\,, \tag{33}$$

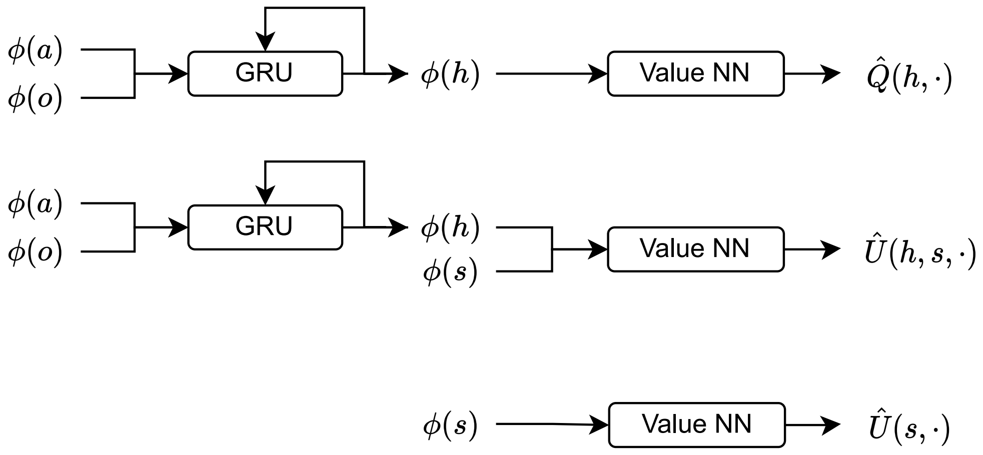
$$Q_{k+1} = (\mathbf{1} - \alpha_k E p_k)Q_k + \alpha_k E p_k (BQ_k + v'_k)\,, \tag{34}$$



(a) Observation, state, and action models for *categorical* POMDPs.



(b) Observation, state, and action models for *gridverse* POMDPs.



(c) DQN and ADQN architectures. Separate components are used for $\hat{Q}(h,a)$, $\hat{U}(h,s,a)$, and $\hat{U}(s,a)$. In each case, the final layer returns an array of values, one for each action $a \in \mathcal{A}$.

Figure 2: For *categorical* and *gridverse* environments, the observation, state, and action models $\phi(o)$, $\phi(s)$, and $\phi(a)$, are those respectively depicted in Figure 2a and Figure 2b. For the *feature-vector* environments, observation and state models $\phi(o)$ and $\phi(s)$ are directly provided as feature-vectors by the environment itself, while action models $\phi(a)$ are implemented as one-hot encodings. Figure 2c shows the architecture for the DQN models $\hat{Q}(h,a)$, $\hat{U}(h,s,a)$, and $\hat{U}(s,a)$.

where

$$w'_k \doteq w_k + \left(\frac{\chi_k}{p_k} - \mathbf{1}\right)\left(B_{g(Q_k)}U_k + w_k - U_k\right), \tag{35}$$

$$v'_k \doteq v_k + \left(\frac{E\chi_k}{Ep_k} - \mathbf{1}\right)\left(BQ_k + v_k - Q_k\right). \tag{36}$$

It can be verified that $\mathbb{E}\left[w'_k\right] = \mathbb{E}\left[v'_k\right] = 0$ and that the conditional variances of w'_k and v'_k are bounded such that Proposition 4.4 of Bertsekas and Tsitsiklis [1995] applies given the conditions on α_k. It follows that Q_k converges with probability 1 to Q^*, the unique fixed point of the contraction mapping B (Lemma 3.2).

The fact that $Q_k \to Q^*$ guarantees the existence of some iteration k^*, with probability 1, such that $g(Q_k) = \pi^*$, for $k \geq k^*$. Therefore,

$$U_{k+1} = (\mathbf{1} - \alpha_k p_k)U_k + \alpha_k p_k(B_{\pi^*}U_k + w'_k), \quad \forall k \geq k^*. \tag{37}$$

B_{π^*} is a contraction mapping that admits U^* as its unique fixed point (Lemma 3.3). Once again, Proposition 4.4 of Bertsekas and Tsitsiklis [1995] applies, and we conclude that $U_k \to U^*$ with probability 1.

□

B MODEL ARCHITECTURES

This section contains the model architectures used by each method when run on each environment of our evaluation (see Figure 2). Some components of the architectures are the same for all methods and environments, while some components are domain-specific, e.g. to accommodate the different structures of states and observations in the different environments. In this section, we refer to the **Heaven-Hell-3** and **Heaven-Hell-4** environments as *categorical* environments; to the **Car-Flag** and **Cleaner** environments as *feature-vector* environments; and to the **GV-MemoryFourRooms-7x7** environment as a *gridverse* environment. For a thorough description of each environment, refer to Appendix C of Baisero and Amato [2022].

General Architecture The architecture components are shown in Figure 2. Action and observation features are concatenated to form the input to a 128-dimensional single-layer *gated recurrent unit* (GRU) Cho et al. [2014], which acts as the history model $\phi(h)$. The value NN is a feedforward model which varies in each type of environment.

Categorical POMDPs The action, observation, and state feature components are shown in Figure 2a. Categorical environments provide actions, observations, and states, as categorical indices, which we convert to parametric feature vectors using 64-dimensional embedding models. The value NN model is a 2-layer feedforward network with 512 and 256 nodes, and ReLU non-linearities.

Feature-Vector POMDPs The observations and states provided by the feature-vector environments already come in a feature-vector form, which we do not process further. Actions are modeles as one-hot encodings. The value NN model is a 2-layer feedforward network with 512 and 256, and ReLU non-linearities.

Gridverse POMDPs The gridverse environments provide observations and states in a dictionary format containing different fields representing different aspects of the environment, see Appendix C of Baisero and Amato [2022]. Because observations and states already contain a lot of relevant information about the past, we use scalar 1-dimensional embedding models for the actions. The $3 \times 2 \times 3$ observations are first processed using an 8-dimensional embedding layer, and then flattened, which produces a 144-dimensional observation feature $\phi(o)$. The states contain relevant information in different forms, and require a more complex model. The `grid` component is processed using an embedding layer, which is then stacked with the `agent_id_grid` component, and processed by a 1- or $2-$ layer feedforward network (this is hyper-parameter L in Appendix D) with ReLU non-linearities. All outputs of the `grid` and `agent_id_grid` components are then concated to form the overall state feature $\phi(s)$.

C TRAINING DETAILS

We perform *fully episodic* training, by which we mean that various aspects of the training involve and are measured based on complete episodes:

- The replay buffer is more specifically an *episode buffer* which contains full episodes.
- The episode buffer is pre-populated using episodes sampled from a random policy. Episodes are sampled and inserted into the episode buffer until the episode buffer contains a total of $50k$ timesteps.
- The main training loop iterates an environment interaction phase with an optimization phase.
- In the environment interaction phase, a full episode is sampled from the environment using an ϵ-greedy policy based on the current $\hat{Q}$. The sampled episode is then inserted into the episode buffer.
- Any time the episode buffer exceeds a total of $1M$ timesteps, old episodes are removed until that is no longer the case.
- In the optimization phase, a variable number of optimization steps are performed. Optimization steps are performed until the total number of timesteps used for training exceeds the number of total timesteps sampled from the real environment by a given factor. This factor is determined by hyper-parameter F (usually 8 or 16). In each optimization step, a number of full episodes are sampled from the episode buffer, and used fully to form the minibatch of transitions used for the optimization step. Because each episode may contain a variable number of transitions, that means that the size of the minibatch of transitions used for optimization is variable. The number of episodes which is sampled is determined by hyper-parameter B (usually 1).
- The whole process is repeated until a given total number of timesteps have been sampled from the environment.

Although this form of training introduces correlations between the transitions in a minibatch $\{(h, s, a, r, s', o)_i\}_{i=1}^N$, it is also significantly more efficient than if transitions were completely i.i.d, due to the shared computation between the features of consecutive histories.

D HYPERPARAMETERS AND GRID SEARCH

For each combination of control problem and method, we perform a separate grid-search over hyper-parameters, and find the combination of hyper-parameters which results in the best performance, in each case using statistics aggregated over 20 independent runs. The hyper-parameter grid is domain dependent. For **Heaven-Hell-3**, **Heaven-Hell-4**, **Car-Flag**, and **Cleaner**, we search over the following:

- $\alpha \in \{0.0001, 0.0003\}$, the learning rate.
- $N_\epsilon \in \{1M, 2M\}$, the number of timesteps it takes for ϵ to decay linearly from its initial value of 1.0 to its final value of 0.1.
- $F \in \{8, 16\}$, the ratio between number of training timesteps and number of simulation timesteps, used to determine the frequency of optimization steps.
- $B \in \{1, 2\}$, the number of episodes sampled from the episode buffer for each optimization step.

For **GV-MemoryFourRooms-7x7**, we set $\alpha = 0.0001$, and search over the following:

- $N_\epsilon \in \{1M, 2M, 4M\}$, the number of timesteps it takes for ϵ to decay linearly from its initial value of 1.0 to its final value of 0.1.
- $F \in \{8, 16\}$, the ratio between number of training timesteps and number of simulation timesteps, used to determine the frequency of optimization steps.
- $B \in \{2, 4\}$, the number of episodes sampled from the episode buffer for each optimization step.
- $L \in \{1, 2\}$, the number of final linear layers in the state and observation representation models.

Factoring in the 4 control problems with 2^4 combinations of hyper-parameters, and 1 control problem with $3 \cdot 2^3$ combinations of hyper-parameters, 5 methods for each, and 20 independent runs for each combination, we obtain a total number of 8800 independent runs necessary to present all the results of this work. Table 1 shows the hyper-parameters chosen from each grid search, which are the ones used for the results depicted in Figure 1.

Table 1: Hyper-parameter grid search results.

Domain	Method	α	N_ϵ	F	B	L
Heaven-Hell-3	**DQN**	0.0003	$2M$	16	2	–
	ADQN	0.0001	$2M$	16	2	–
	ADQN-VR	0.0001	$2M$	16	2	–
	ADQN-State	0.0001	$2M$	16	2	–
	ADQN-State-VR	0.0001	$2M$	16	2	–
Heaven-Hell-4	**DQN**	0.0003	$2M$	16	1	–
	ADQN	0.0001	$2M$	16	2	–
	ADQN-VR	0.0001	$2M$	16	1	–
	ADQN-State	0.0001	$2M$	16	2	–
	ADQN-State-VR	0.0001	$2M$	16	2	–
Car-Flag	**DQN**	0.0003	$1M$	16	2	–
	ADQN	0.0003	$1M$	16	1	–
	ADQN-VR	0.0003	$1M$	16	2	–
	ADQN-State	0.0003	$1M$	16	2	–
	ADQN-State-VR	0.0003	$1M$	16	1	–
Cleaner	**DQN**	0.0001	$2M$	8	2	–
	ADQN	0.0001	$1M$	16	2	–
	ADQN-VR	0.0001	$2M$	16	2	–
	ADQN-State	0.0001	$1M$	8	2	–
	ADQN-State-VR	0.0001	$2M$	8	2	–
GV-MemoryFourRooms-7x7	**DQN**	0.0001	$4M$	16	2	1
	ADQN	0.0001	$1M$	16	2	1
	ADQN-VR	0.0001	$1M$	16	2	1
	ADQN-State	0.0001	$4M$	16	2	1
	ADQN-State-VR	0.0001	$4M$	16	2	1