Hierarchical Imitation and Reinforcement Learning

Hoang M. Le [1] **Nan Jiang** [2] **Alekh Agarwal** [2] **Miroslav Dudík** [2] **Yisong Yue** [1] **Hal Daumé III** [3][2]

Abstract

We study how to effectively leverage expert feedback to learn sequential decision-making policies. We focus on problems with sparse rewards and long time horizons, which typically pose significant challenges in reinforcement learning. We propose an algorithmic framework, called *hierarchical guidance*, that leverages the hierarchical structure of the underlying problem to integrate different modes of expert interaction. Our framework can incorporate different combinations of imitation learning (IL) and reinforcement learning (RL) at different levels, leading to dramatic reductions in both expert effort and cost of exploration. Using long-horizon benchmarks, including Montezuma's Revenge, we demonstrate that our approach can learn significantly faster than hierarchical RL, and be significantly more label-efficient than standard IL. We also theoretically analyze labeling cost for certain instantiations of our framework.

1. Introduction

Learning good agent behavior from reward signals alone—the goal of reinforcement learning (RL)—is particularly difficult when the planning horizon is long and rewards are sparse. One successful method for dealing with such long horizons is imitation learning (IL) (Abbeel & Ng, 2004; Daumé et al., 2009; Ross et al., 2011; Ho & Ermon, 2016), in which the agent learns by watching and possibly querying an expert. One limitation of existing imitation learning approaches is that they may require a large amount of demonstration data in long-horizon problems.

The central question we address in this paper is: *when experts are available, how can we most effectively leverage their feedback?* A common strategy to improve sample ef-

ficiency in RL over long time horizons is to exploit hierarchical structure of the problem (Sutton et al., 1998; 1999; Kulkarni et al., 2016; Dayan & Hinton, 1993; Vezhnevets et al., 2017; Dietterich, 2000). Our approach leverages hierarchical structure in imitation learning. We study the case where the underlying problem is hierarchical, and subtasks can be easily elicited from an expert. Our key design principle is an algorithmic framework called *hierarchical guidance*, in which feedback (labels) from the high-level expert is used to focus (guide) the low-level learner. The high-level expert ensures that low-level learning only occurs when necessary (when subtasks have not been mastered) and only over relevant parts of the state space. This differs from a naïve hierarchical approach which merely gives a subtask decomposition. Focusing on relevant parts of the state space speeds up learning (improves sample efficiency), while omitting feedback on the already mastered subtasks reduces expert effort (improves label efficiency).

We begin by formalizing the problem of hierarchical imitation learning (Section 3) and carefully separate out cost structures that naturally arise when the expert provides feedback at multiple levels of abstraction. We first apply hierarchical guidance to IL, derive hierarchically guided variants of behavior cloning and DAgger (Ross et al., 2011), and theoretically analyze the benefits (Section 4). We next apply hierarchical guidance to the hybrid setting with high-level IL and low-level RL (Section 5). This architecture is particularly suitable in settings where we have access to high-level semantic knowledge, the subtask horizon is sufficiently short, but the low-level expert is too costly or unavailable. We demonstrate the efficacy of our approaches on a simple but extremely challenging maze domain, and on Montezuma's Revenge (Section 6). Our experiments show that incorporating a modest amount of expert feedback can lead to dramatic improvements in performance compared to pure hierarchical RL.[1]

2. Related Work

For brevity, we provide here a short overview of related work, and defer to Appendix C for additional discussion.

---

[1]California Institute of Technology, Pasadena, CA [2]Microsoft Research, New York, NY [3]University of Maryland, College Park, MD. Correspondence to: Hoang M. Le <hmle@caltech.edu>.

---

[1]Code and experimental setups are available at https://sites.google.com/view/hierarchical-il-rl

Imitation Learning. One can broadly dichotomize IL into passive collection of demonstrations (behavioral cloning) versus active collection of demonstrations. The former setting (Abbeel & Ng, 2004; Ziebart et al., 2008; Syed & Schapire, 2008; Ho & Ermon, 2016) assumes that demonstrations are collected a priori and the goal of IL is to find a policy that mimics the demonstrations. The latter setting (Daumé et al., 2009; Ross et al., 2011; Ross & Bagnell, 2014; Chang et al., 2015; Sun et al., 2017) assumes an interactive expert that provides demonstrations in response to actions taken by the current policy. We explore extension of both approaches into hierarchical settings.

Hierarchical Reinforcement Learning. Several RL approaches to learning hierarchical policies have been explored, foremost among them the options framework (Sutton et al., 1998; 1999; Fruit & Lazaric, 2017). It is often assumed that a useful set of options are fully defined a priori, and (semi-Markov) planning and learning only occurs at the higher level. In comparison, our agent does not have direct access to policies that accomplish such subgoals and has to learn them via expert or reinforcement feedback. The closest hierarchical RL work to ours is that of Kulkarni et al. (2016), which uses a similar hierarchical structure, but no high-level expert and hence no hierarchical guidance.

Combining Reinforcement and Imitation Learning. The idea of combining IL and RL is not new (Nair et al., 2017; Hester et al., 2018). However, previous work focuses on flat policy classes that use IL as a "pre-training" step (e.g., by pre-populating the replay buffer with demonstrations). In contrast, we consider feedback at multiple levels for a hierarchical policy class, with different levels potentially receiving different types of feedback (i.e., imitation at one level and reinforcement at the other). Somewhat related to our hierarchical expert supervision is the approach of Andreas et al. (2017), which assumes access to symbolic descriptions of subgoals, without knowing what those symbols mean or how to execute them. Previous literature has not focused much on comparisons of sample complexity between IL and RL, with the exception of the recent work of Sun et al. (2017).

3. Hierarchical Formalism

For simplicity, we consider environments with a natural two-level hierarchy; the HI level corresponds to choosing subtasks, and the LO level corresponds to executing those subtasks. For instance, an agent's overall goal may be to leave a building. At the HI level, the agent may first choose the subtask *"go to the elevator,"* then *"take the elevator down,"* and finally *"walk out."* Each of these subtasks needs to be executed at the LO level by actually navigat-

ing the environment, pressing buttons on the elevator, etc.[2]

Subtasks, which we also call *subgoals*, are denoted as $g \in \mathcal{G}$, and the primitive actions are denoted as $a \in \mathcal{A}$. An agent (also referred to as learner) acts by iteratively choosing a subgoal g, carrying it out by executing a sequence of actions a until completion, and then picking a new subgoal. The agent's choices can depend on an observed state $s \in \mathcal{S}$.[3] We assume that the horizon at the HI level is H_{HI}, i.e., a trajectory uses at most H_{HI} subgoals, and the horizon at the LO level is H_{LO}, i.e., after at most H_{LO} primitive actions, the agent either accomplishes the subgoal or needs to decide on a new subgoal. The total number of primitive actions in a trajectory is thus at most $H_{\text{FULL}} \coloneqq H_{\text{HI}} H_{\text{LO}}$.

The hierarchical learning problem is to simultaneously learn a HI-level policy $\mu : \mathcal{S} \rightarrow \mathcal{G}$, called the *meta-controller*, as well as the subgoal policies $\pi_g : \mathcal{S} \rightarrow \mathcal{A}$ for each $g \in \mathcal{G}$, called *subpolicies*. The aim of the learner is to achieve a high reward when its meta-controller and subpolicies are run together. For each subgoal g, we also have a (possibly learned) termination function $\beta_g : \mathcal{S} \rightarrow \{True, False\}$, which terminates the execution of π_g. The hierarchical agent behaves as follows:

1: **for** $h_{\text{HI}} = 1 \ldots H_{\text{HI}}$ **do**
2:     observe state s and choose subgoal $g \leftarrow \mu(s)$
3:     **for** $h_{\text{LO}} = 1 \ldots H_{\text{LO}}$ **do**
4:         observe state s
5:         **if** $\beta_g(s)$ **then break**
6:         choose action $a \leftarrow \pi_g(s)$

The execution of each subpolicy π_g generates a LO-*level trajectory* $\tau = (s_1, a_1, \ldots, s_H, a_H, s_{H+1})$ with $H \leq H_{\text{LO}}$.[4] The overall behavior results in a *hierarchical trajectory* $\sigma = (s_1, g_1, \tau_1, s_2, g_2, \tau_2, \ldots)$, where the last state of each LO-level trajectory τ_h coincides with the next state s_{h+1} in σ and the first state of the next LO-level trajectory τ_{h+1}. The subsequence of σ which excludes the LO-level trajectories τ_h is called the HI-*level trajectory*, $\tau_{\text{HI}} \coloneqq (s_1, g_1, s_2, g_2, \ldots)$. Finally, the *full trajectory*, τ_{FULL}, is the concatenation of all the LO-level trajectories.

We assume access to an *expert*, endowed with a meta-

---

[2]An important real-world application is in goal-oriented dialogue systems. For instance, a chatbot assisting a user with reservation and booking for flights and hotels (Peng et al., 2017; El Asri et al., 2017) needs to navigate through multiple turns of conversation. The chatbot developer designs the hierarchy of subtasks, such as *ask_user_goal*, *ask_dates*, *offer_flights*, *confirm*, etc. Each subtask consists of several turns of conversation. Typically a global state tracker exists alongside the hierarchical dialogue policy to ensure that cross-subtask constraints are satisfied.

[3]While we use the term state for simplicity, we do not require the environment to be fully observable or Markovian.

[4]The trajectory might optionally include a reward signal after each primitive action, which might either come from the environment, or be a pseudo-reward as we will see in Section 5.

controller $\mu^\star$, subpolicies $\pi_g^\star$, and termination functions $\beta_g^\star$, who can provide one or several types of supervision:

- `HierDemo`(s): *hierarchical demonstration*. The expert executes its hierarchical policy starting from s and returns the resulting hierarchical trajectory $\sigma^\star = (s_1^\star, g_1^\star, \tau_1^\star, s_2^\star, g_2^\star, \tau_2^\star, \dots)$, where $s_1^\star = s$.

- `Label`$_{\text{HI}}(\tau_{\text{HI}})$: HI-*level labeling*. The expert provides a good next subgoal at each state of a given HI-level trajectory $\tau_{\text{HI}} = (s_1, g_1, s_2, g_2, \dots)$, yielding a labeled data set $\{(s_1, g_1^\star), (s_2, g_2^\star), \dots\}$.

- `Label`$_{\text{LO}}(\tau; g)$: LO-*level labeling*. The expert provides a good next primitive action towards a given subgoal g at each state of a given LO-level trajectory $\tau = (s_1, a_1, s_2, a_2, \dots)$, yielding a labeled data set $\{(s_1, a_1^\star), (s_2, a_2^\star), \dots\}$.

- `Inspect`$_{\text{LO}}(\tau; g)$: LO-*level inspection*. Instead of annotating every state of a trajectory with a good action, the expert only verifies whether a subgoal g was accomplished, returning either *Pass* or *Fail*.

- `Label`$_{\text{FULL}}(\tau_{\text{FULL}})$: *full labeling*. The expert labels the agent's full trajectory $\tau_{\text{FULL}} = (s_1, a_1, s_2, a_2, \dots)$, from start to finish, ignoring hierarchical structure, yielding a labeled data set $\{(s_1, a_1^\star), (s_2, a_2^\star), \dots\}$.

- `Inspect`$_{\text{FULL}}(\tau_{\text{FULL}})$: *full inspection*. The expert verifies whether the agent's overall goal was accomplished, returning either *Pass* or *Fail*.

When the agent learns not only the subpolicies π_g, but also termination functions β_g, then `Label`$_{\text{LO}}$ also returns good termination values $\omega^\star \in \{True, False\}$ for each state of $\tau = (s_1, a_1 \dots)$, yielding a data set $\{(s_1, a_1^\star, \omega_1^\star), \dots\}$.

Although `HierDemo` and `Label` can be both generated by the expert's hierarchical policy $(\mu^\star, \{\pi_g^\star\})$, they differ in the mode of expert interaction. `HierDemo` returns a hierarchical trajectory *executed by the expert*, as required for passive IL, and enables a hierarchical version of behavioral cloning (Abbeel & Ng, 2004; Syed & Schapire, 2008). `Label` operations provide labels *with respect to the learning agent's trajectories*, as required for interactive IL. `Label`$_{\text{FULL}}$ is the standard query used in prior work on learning flat policies (Daumé et al., 2009; Ross et al., 2011), and `Label`$_{\text{HI}}$ and `Label`$_{\text{LO}}$ are its hierarchical extensions.

`Inspect` operations are newly introduced in this paper, and form a cornerstone of our interactive hierarchical guidance protocol that enables substantial savings in label efficiency. They can be viewed as "lazy" versions of the corresponding `Label` operations, requiring less effort. Our underlying assumption is that if the given hierarchical trajectory $\sigma = \{(s_h, g_h, \tau_h)\}$ agrees with the expert on HI level, i.e., $g_h = \mu^\star(s_h)$, and LO-level trajectories pass the

---

Algorithm 1 Hierarchical Behavioral Cloning (**h-BC**)

1: Initialize data buffers $\mathcal{D}_{\text{HI}} \leftarrow \emptyset$ and $\mathcal{D}_g \leftarrow \emptyset, g \in \mathcal{G}$
2: **for** $t = 1, \dots, T$ **do**
3:     Get a new environment instance with start state s
4:     $\sigma^\star \leftarrow$ `HierDemo`(s)
5:     **for all** $(s_h^\star, g_h^\star, \tau_h^\star) \in \sigma^\star$ **do**
6:         Append $\mathcal{D}_{g_h^\star} \leftarrow \mathcal{D}_{g_h^\star} \cup \tau_h^\star$
7:         Append $\mathcal{D}_{\text{HI}} \leftarrow \mathcal{D}_{\text{HI}} \cup \{(s_h^\star, g_h^\star)\}$
8: Train subpolicies $\pi_g \leftarrow$ `Train`$(\pi_g, \mathcal{D}_g)$ for all g
9: Train meta-controller $\mu \leftarrow$ `Train`$(\mu, \mathcal{D}_{\text{HI}})$

---

inspection, i.e., `Inspect`$_{\text{LO}}(\tau_h; g_h) = $ *Pass*, then the resulting full trajectory must also pass the full inspection, `Inspect`$_{\text{FULL}}(\tau_{\text{FULL}}) = $ *Pass*. This means that a hierarchical policy need not always agree with the expert's execution at LO level to succeed in the overall task.

Besides algorithmic reasons, the motivation for separating the types of feedback is that different expert queries will typically require different amount of effort, which we refer to as *cost*. We assume the costs of the `Label` operations are C_{HI}^L, C_{LO}^L and C_{FULL}^L, the costs of each `Inspect` operation are C_{LO}^I and C_{FULL}^I. In many settings, LO-level inspection will require significantly less effort than LO-level labeling, i.e., $C_{\text{LO}}^I \ll C_{\text{LO}}^L$. For instance, identifying if a robot has successfully navigated to the elevator is presumably much easier than labeling an entire path to the elevator. One reasonable cost model, natural for the environments in our experiments, is to assume that `Inspect` operations take time $O(1)$ and work by checking the final state of the trajectory, whereas `Label` operations take time proportional to the trajectory length, which is $O(H_{\text{HI}})$, $O(H_{\text{LO}})$ and $O(H_{\text{HI}} H_{\text{LO}})$ for our three `Label` operations.

4. Hierarchically Guided Imitation Learning

Hierarchical guidance is an algorithmic design principle in which the feedback from high-level expert guides the low-level learner in two different ways: (i) the high-level expert ensures that low-level expert is only queried when necessary (when the subtasks have not been mastered yet), and (ii) low-level learning is limited to the relevant parts of the state space. We instantiate this framework first within passive learning from demonstrations, obtaining *hierarchical behavioral cloning* (Algorithm 1), and then within interactive imitation learning, obtaining *hierarchically guided DAgger* (Algorithm 2), our best-performing algorithm.

4.1. Hierarchical Behavioral Cloning (h-BC)

We consider a natural extension of behavioral cloning to the hierarchical setting (Algorithm 1). The expert provides a set of hierarchical demonstrations $\sigma^\star$, each consisting of LO-level trajectories $\tau_h^\star = \{(s_\ell^\star, a_\ell^\star)\}_{\ell=1}^{H_{\text{LO}}}$ as well as a HI-level trajectory $\tau_{\text{HI}}^\star = \{(s_h^\star, g_h^\star)\}_{h=1}^{H_{\text{HI}}}$. We then run

---

Algorithm 2 Hierarchically Guided DAgger (**hg-DAgger**)

1: Initialize data buffers $\mathcal{D}_{\text{HI}} \leftarrow \emptyset$ and $\mathcal{D}_g \leftarrow \emptyset, g \in \mathcal{G}$
2: Run Hierarchical Behavioral Cloning (Algorithm 1)
      up to $t = T_{\text{warm-start}}$
3: **for** $t = T_{\text{warm-start}} + 1, \ldots, T$ **do**
4:     Get a new environment instance with start state s
5:     Initialize $\sigma \leftarrow \emptyset$
6:     **repeat**
7:         $g \leftarrow \mu(s)$
8:         Execute π_g, obtain LO-level trajectory τ
9:         Append (s, g, τ) to σ
10:        $s \leftarrow$ the last state in τ
11:    **until** end of episode
12:    Extract τ_{FULL} and τ_{HI} from σ
13:    **if** $\texttt{Inspect}_{\text{FULL}}(\tau_{\text{FULL}}) = $ *Fail* **then**
14:        $\mathcal{D}^\star \leftarrow \texttt{Label}_{\text{HI}}(\tau_{\text{HI}})$
15:        Process $(s_h, g_h, \tau_h) \in \sigma$ in sequence as long as
              g_h agrees with the expert's choice $g_h^\star$ in $\mathcal{D}^\star$:
16:            **if** $\texttt{Inspect}(\tau_h; g_h) = $ *Fail* **then**
17:                Append $\mathcal{D}_{g_h} \leftarrow \mathcal{D}_{g_h} \cup \texttt{Label}_{\text{LO}}(\tau_h; g_h)$
18:                **break**
19:        Append $\mathcal{D}_{\text{HI}} \leftarrow \mathcal{D}_{\text{HI}} \cup \mathcal{D}^\star$
20:    Update subpolicies $\pi_g \leftarrow \texttt{Train}(\pi_g, \mathcal{D}_g)$ for all g
21:    Update meta-controller $\mu \leftarrow \texttt{Train}(\mu, \mathcal{D}_{\text{HI}})$

$\texttt{Train}$ (lines 8–9) to find the subpolicies π_g that best predict $a_\ell^\star$ from $s_\ell^\star$, and meta-controller μ that best predicts $g_h^\star$ from $s_h^\star$, respectively. $\texttt{Train}$ can generally be any supervised learning subroutine, such as stochastic optimization for neural networks or some batch training procedure. When termination functions β_g need to be learned as part of the hierarchical policy, the labels $\omega_g^\star$ will be provided by the expert as part of $\tau_h^\star = \{(s_\ell^\star, a_\ell^\star, \omega_\ell^\star)\}$.[5] In this setting, hierarchical guidance is automatic, because subpolicy demonstrations only occur in relevant parts of the state space.

4.2. Hierarchically Guided DAgger (hg-DAgger)

Passive IL, e.g., behavioral cloning, suffers from the distribution mismatch between the learning and execution distributions. This mismatch is addressed by interactive IL algorithms, such as SEARN (Daumé et al., 2009) and DAgger (Ross et al., 2011), where the expert provides correct actions along the learner's trajectories through the operation $\texttt{Label}_{\text{FULL}}$. A naïve hierarchical implementation would provide correct labels along the entire hierarchical trajectory via $\texttt{Label}_{\text{HI}}$ and $\texttt{Label}_{\text{LO}}$. We next show how to use hierarchical guidance to decrease LO-level expert costs.

We leverage two HI-level query types: $\texttt{Inspect}_{\text{LO}}$ and $\texttt{Label}_{\text{HI}}$. We use $\texttt{Inspect}_{\text{LO}}$ to verify whether the subtasks are successfully completed and $\texttt{Label}_{\text{HI}}$ to check whether we are staying in the relevant part of the state space. The details are presented in Algorithm 2, which uses

---

[5]In our hierarchical imitation learning experiments, the termination functions are all learned. Formally, the termination signal ω_g, can be viewed as part of an augmented action at LO level.

DAgger as the learner on both levels, but the scheme can be adapted to other interactive imitation learners.

In each episode, the learner executes the hierarchical policy, including choosing a subgoal (line 7), executing the LO-level trajectories, i.e., rolling out the subpolicy π_g for the chosen subgoal, and terminating the execution according to β_g (line 8). Expert only provides feedback when the agent fails to execute the entire task, as verified by $\texttt{Inspect}_{\text{FULL}}$ (line 13). When $\texttt{Inspect}_{\text{FULL}}$ fails, the expert first labels the correct subgoals via $\texttt{Label}_{\text{HI}}$ (line 14), and only performs LO-level labeling as long as the learner's meta-controller chooses the correct subgoal g_h (line 15), but its subpolicy fails (i.e., when $\texttt{Inspect}_{\text{LO}}$ on line 16 fails). Since all the preceding subgoals were chosen and executed correctly, and the current subgoal is also correct, LO-level learning is in the "relevant" part of the state space. However, since the subpolicy execution failed, its learning has not been mastered yet. We next analyze the savings in expert cost that result from hierarchical guidance.

Theoretical Analysis. We analyze the cost of hg-DAgger in comparison with flat DAgger under somewhat stylized assumptions. We assume that the learner aims to learn the meta-controller μ from some policy class $\mathcal{M}$, and subpolicies π_g from some class Π_{LO}. The classes $\mathcal{M}$ and Π_{LO} are finite (but possibly exponentially large) and the task is realizable, i.e., the expert's policies can be found in the corresponding classes: $\mu^\star \in \mathcal{M}$, and $\pi_g^\star \in \Pi_{\text{LO}}, g \in \mathcal{G}$. This allows us to use the *halving algorithm* (Shalev-Shwartz et al., 2012) as the online learner on both levels. (The implementation of our algorithm does not require these assumptions.)

The halving algorithm maintains a version space over policies, acts by a majority decision, and when it makes a mistake, it removes all the erring policies from the version space. In the hierarchical setting, it therefore makes at most $\log |\mathcal{M}|$ mistakes on the HI level, and at most $\log |\Pi_{\text{LO}}|$ mistakes when learning each π_g. The mistake bounds can be further used to upper bound the total expert cost in both hg-DAgger and flat DAgger. To enable an apples-to-apples comparison, we assume that the flat DAgger learns over the policy class $\Pi_{\text{FULL}} = \{(\mu, \{\pi_g\}_{g\in\mathcal{G}}) : \mu \in \mathcal{M}, \pi_g \in \Pi_{\text{LO}}\}$, but is otherwise oblivious to the hierarchical task structure. The bounds depend on the cost of performing different types of operations, as defined at the end of Section 3. We consider a modified version of flat DAgger that first calls $\texttt{Inspect}_{\text{FULL}}$, and only requests labels ($\texttt{Label}_{\text{FULL}}$) if the inspection fails. The proofs are deferred to Appendix A.

Theorem 1. *Given finite classes $\mathcal{M}$ and Π_{LO} and realizable expert policies, the total cost incurred by the expert in hg-DAgger by round T is bounded by*

$$TC_{\text{FULL}}^I + \left(\log_2 |\mathcal{M}| + |\mathcal{G}_{opt}| \log_2 |\Pi_{\text{LO}}|\right)\left(C_{\text{HI}}^L + H_{\text{HI}} C_{\text{LO}}^I\right)$$
$$+ \left(|\mathcal{G}_{opt}| \log_2 |\Pi_{\text{LO}}|\right) C_{\text{LO}}^L, \tag{1}$$

where $\mathcal{G}_{opt} \subseteq \mathcal{G}$ is the set of the subgoals actually used by the expert, $\mathcal{G}_{opt} := \mu^\star(\mathcal{S})$.

Theorem 2. *Given the full policy class* $\Pi_{\text{FULL}} = \{(\mu, \{\pi_g\}_{g \in \mathcal{G}}) : \mu \in \mathcal{M}, \pi_g \in \Pi_{\text{LO}}\}$ *and a realizable expert policy, the total cost incurred by the expert in flat DAgger by round T is bounded by*

$$TC^I_{\text{FULL}} + \left(\log_2 |\mathcal{M}| + |\mathcal{G}| \log_2 |\Pi_{\text{LO}}|\right) C^L_{\text{FULL}}. \quad (2)$$

Both bounds have the same leading term, TC^I_{FULL}, the cost of full inspection, which is incurred every round and can be viewed as the "cost of monitoring." In contrast, the remaining terms can be viewed as the "cost of learning" in the two settings, and include terms coming from their respective mistake bounds. The ratio of the cost of hierarchically guided learning to the flat learning is then bounded as

$$\frac{\text{Eq. (1)} - TC^I_{\text{FULL}}}{\text{Eq. (2)} - TC^I_{\text{FULL}}} \le \frac{C^L_{\text{HI}} + H_{\text{HI}} C^I_{\text{LO}} + C^L_{\text{LO}}}{C^L_{\text{FULL}}}, \quad (3)$$

where we applied the upper bound $|\mathcal{G}_{opt}| \le |\mathcal{G}|$. The savings thanks to hierarchical guidance depend on the specific costs. Typically, we expect the inspection costs to be $O(1)$, if it suffices to check the final state, whereas labeling costs scale linearly with the length of the trajectory. The cost ratio is then $\propto \frac{H_{\text{HI}} + H_{\text{LO}}}{H_{\text{HI}} H_{\text{LO}}}$. Thus, we realize most significant savings if the horizons on each individual level are substantially shorter than the overall horizon. In particular, if $H_{\text{HI}} = H_{\text{LO}} = \sqrt{H_{\text{FULL}}}$, the hierarchically guided approach reduces the overall labeling cost by a factor of $\sqrt{H_{\text{FULL}}}$. More generally, whenever H_{FULL} is large, we reduce the costs of learning be at least a constant factor—a significant gain if this is a saving in the effort of a domain expert.

5. Hierarchically Guided IL / RL

Hierarchical guidance also applies in the hybrid setting with interactive IL on the HI level and RL on the LO level. The HI-level expert provides the hierarchical decomposition, including the pseudo-reward function for each subgoal,[6] and is also able to pick a correct subgoal at each step. Similar to hg-DAgger, the labels from HI-level expert are used not only to train the meta-controller μ, but also to limit the LO-level learning to the relevant part of the state space. In Algorithm 3 we provide the details, with DAgger on HI level and Q-learning on LO level. The scheme can be adapted to other interactive IL and RL algorithms.

The learning agent proceeds by *rolling in* with its meta-controller (line 7). For each selected subgoal g, the subpolicy π_g selects and executes primitive actions via the

---

[6]This is consistent with many hierarchical RL approaches, including options (Sutton et al., 1999), MAXQ (Dieterich, 2000), UVFA (Schaul et al., 2015a) and h-DQN (Kulkarni et al., 2016).

Algorithm 3 Hierarchically Guided DAgger / Q-learning (**hg-DAgger/Q**)

input Function `pseudo(s; g)` providing the pseudo-reward
input Predicate `terminal(s; g)` indicating the termination of g
input Annealed exploration probabilities $\epsilon_g > 0$, $g \in \mathcal{G}$
1: Initialize data buffers $\mathcal{D}_{\text{HI}} \leftarrow \emptyset$ and $\mathcal{D}_g \leftarrow \emptyset$, $g \in \mathcal{G}$
2: Initialize subgoal Q-functions Q_g, $g \in \mathcal{G}$
3: **for** $t = 1, \dots, T$ **do**
4:     Get a new environment instance with start state s
5:     Initialize $\sigma \leftarrow \emptyset$
6:     **repeat**
7:         $s_{\text{HI}} \leftarrow s$, $g \leftarrow \mu(s)$ and initialize $\tau \leftarrow \emptyset$
8:         **repeat**
9:             $a \leftarrow \epsilon_g$-greedy(Q_g, s)
10:            Execute a, next state $\tilde{s}$, $\tilde{r} \leftarrow$ `pseudo(`$\tilde{s}; g$`)`
11:            Update Q_g: a (stochastic) gradient descent step
                         on a minibatch from $\mathcal{D}_g$
12:            Append $(s, a, \tilde{r}, \tilde{s})$ to τ and update $s \leftarrow \tilde{s}$
13:         **until** `terminal(`$s; g$`)`
14:         Append (s_{HI}, g, τ) to σ
15:     **until** end of episode
16:     Extract τ_{FULL} and τ_{HI} from σ
17:     **if** `Inspect`$_{\text{FULL}}(\tau_{\text{FULL}})$ = *Fail* **then**
18:         $\mathcal{D}^\star \leftarrow$ `Label`$_{\text{HI}}(\tau_{\text{HI}})$
19:         Process $(s_h, g_h, \tau_h) \in \sigma$ in sequence as long as
            g_h agrees with the expert's choice $g_h^\star$ in $\mathcal{D}^\star$:
20:            Append $\mathcal{D}_{g_h} \leftarrow \mathcal{D}_{g_h} \cup \tau_h$
           Append $\mathcal{D}_{\text{HI}} \leftarrow \mathcal{D}_{\text{HI}} \cup \mathcal{D}^\star$
21:     **else**
22:         Append $\mathcal{D}_{g_h} \leftarrow \mathcal{D}_{g_h} \cup \tau_h$ for all $(s_h, g_h, \tau_h) \in \sigma$
23:     Update meta-controller $\mu \leftarrow$ `Train(`$\mu, \mathcal{D}_{\text{HI}}$`)`

ϵ-greedy rule (lines 9–10), until some termination condition is met. The agent receives some pseudo-reward, also known as intrinsic reward (Kulkarni et al., 2016) (line 10). Upon termination of the subgoal, agent's meta-controller μ chooses another subgoal and the process continues until the end of the episode, where the involvement of the expert begins. As in hg-DAgger, the expert inspects the overall execution of the learner (line 17), and if it is not successful, the expert provides HI-level labels, which are accumulated for training the meta-controller.

Hierarchical guidance impacts how the LO-level learners accumulate experience. As long as the meta-controller's subgoal g agrees with the expert's, the agent's experience of executing subgoal g is added to the experience replay buffer $\mathcal{D}_g$. If the meta-controller selects a "bad" subgoal, the accumulation of experience in the current episode is terminated. This ensures that experience buffers contain only the data from the relevant part of the state space.

Algorithm 3 assumes access to a real-valued function `pseudo(s; g)`, providing the pseudo-reward in state s when executing g, and a predicate `terminal(s; g)`, indicating the termination (not necessarily successful) of subgoal g. This setup is similar to prior work on hierarchical RL (Kulkarni et al., 2016). One natural defini-

tion of pseudo-rewards, based on an additional predicate success$(s; g)$ indicating a successful completion of subgoal g, is as follows:

$$\begin{cases} 1 & \text{if } \texttt{success}(s; g) \\ -1 & \text{if } \neg\texttt{success}(s; g) \text{ and } \texttt{terminal}(s; g) \\ -\kappa & \text{otherwise,} \end{cases}$$

where $\kappa > 0$ is a small penalty to encourage short trajectories. The predicates success and terminal are provided by an expert or learnt from supervised or reinforcement feedback. In our experiments, we explicitly provide these predicates to both hg-DAgger/Q as well as the hierarchical RL, giving them advantage over hg-DAgger, which needs to learn when to terminate subpolicies.

6. Experiments

We evaluate the performance of our algorithms on two separate domains: (i) a simple but challenging maze navigation domain and (ii) the Atari game Montezuma's Revenge.

6.1. Maze Navigation Domain

Task Overview. Figure 1 (left) displays a snapshot of the maze navigation domain. In each episode, the agent encounters a new instance of the maze from a large collection of different layouts. Each maze consists of 16 rooms arranged in a 4-by-4 grid, but the openings between the rooms vary from instance to instance as does the initial position of the agent and the target. The agent (white dot) needs to navigate from one corner of the maze to the target marked in yellow. Red cells are obstacles (lava), which the agent needs to avoid for survival. The contextual information the agent receives is the pixel representation of a bird's-eye view of the environment, including the partial trail (marked in green) indicating the visited locations.

Due to a large number of random environment instances, this domain is not solvable with tabular algorithms. Note that rooms are not always connected, and the locations of the hallways are not always in the middle of the wall. Primitive actions include going one step *up*, *down*, *left* or *right*. In addition, each instance of the environment is designed to ensure that there is a path from initial location to target, and the shortest path takes at least 45 steps ($H_{\text{FULL}} = 100$). The agent is penalized with reward -1 if it runs into lava, which also terminates the episode. The agent only receives positive reward upon stepping on the yellow block.

A hierarchical decomposition of the environment corresponds to four possible subgoals of going to the room immediately to the *north*, *south*, *west*, *east*, and the fifth possible subgoal *go_to_target* (valid only in the room containing the target). In this setup, $H_{\text{LO}} \approx 5$ steps, and $H_{\text{HI}} \approx 10\text{--}12$ steps. The episode terminates after 100 prim-

itive steps if the agent is unsuccessful. The subpolicies and meta-controller use similar neural network architectures and only differ in the number of action outputs. (Details of network architecture are provided in Appendix B.)

Hierarchically Guided IL. We first compare our hierarchical IL algorithms with their flat versions. The algorithm performance is measured by success rate, defined as the average rate of successful task completion over the previous 100 test episodes, on random environment instances not used for training. The cost of each Label operation equals the length of the labeled trajectory, and the cost of each Inspect operation equals 1.

Both h-BC and hg-DAgger outperform flat imitation learners (Figure 2, left). hg-DAgger, in particular, achieves consistently the highest success rate, approaching 100% in fewer than 1000 episodes. Figure 2 (left) displays the median as well as the range from minimum to maximum success rate over 5 random executions of the algorithms.

Expert cost varies significantly between the two hierarchical algorithms. Figure 2 (middle) displays the same success rate, but as a function of the expert cost. hg-DAgger achieves significant savings in expert cost compared to other imitation learning algorithms thanks to a more efficient use of the LO-level expert through hierarchical guidance. Figure 1 (middle) shows that hg-DAgger requires most of its LO-level labels early in the training and requests primarily HI-level labels after the subgoals have been mastered. As a result, hg-DAgger requires only a fraction of LO-level labels compared to flat DAgger (Figure 2, right).

Hierarchically Guided IL / RL. We evaluate hg-DAgger/Q with deep double Q-learning (DDQN, Van Hasselt et al., 2016) and prioritized experience replay (Schaul et al., 2015b) as the underlying RL procedure. Each subpolicy learner receives a pseudo-reward of 1 for each successful execution, corresponding to stepping through the correct door (e.g., door to the north if the subgoal is *north*) and negative reward for stepping into lava or through other doors.

Figure 1 (right) shows the learning progression of hg-DAgger/Q, implying two main observations. First, the number of HI-level labels rapidly increases initially and then flattens out after the learner becomes more successful, thanks to the availability of Inspect$_{\text{FULL}}$ operation. As the hybrid algorithm makes progress and the learning agent passes the Inspect$_{\text{FULL}}$ operation increasingly often, the algorithm starts saving significantly on expert feedback. Second, the number of HI-level labels is higher than for both hg-DAgger and h-BC. Inspect$_{\text{FULL}}$ returns *Fail* often, especially during the early parts of training. This is primarily due to the slower learning speed of Q-learning at the LO level, requiring more expert feedback at the HI
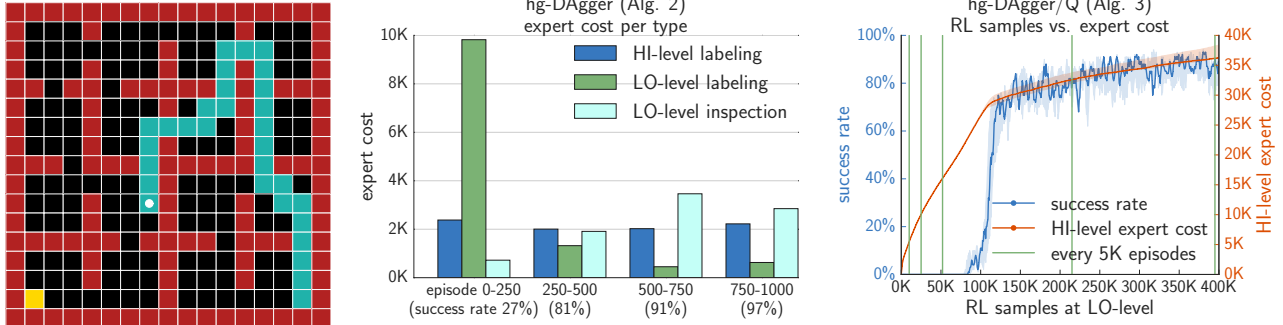
Figure 1. Maze navigation. (Left) One sampled environment instance; the agent needs to navigate from bottom right to bottom left. *(Middle)* Expert cost over time for hg-DAgger; the cost of `Label` operations equals the length of labeled trajectory, the cost of `Inspect` operations is 1. *(Right)* Success rate of hg-DAgger/Q and the HI-level label cost as a function of the number of LO-level RL samples.
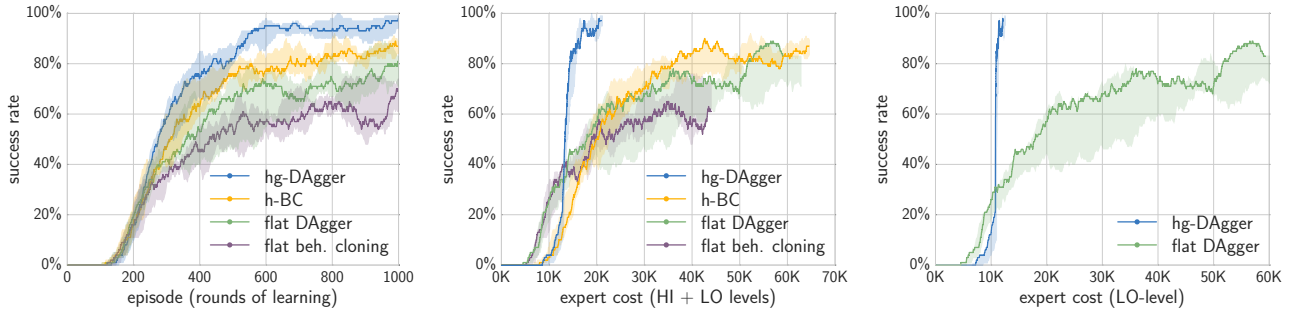

Figure 2. Maze navigation: hierarchical versus flat imitation learning. Each episode is followed by a round of training and a round of testing. The success rate is measured over previous 100 test episodes; the expert cost is as in Figure 1. *(Left)* Success rate per episode. *(Middle)* Success rate versus the expert cost. *(Right)* Success rate versus the LO-level expert cost.

level. This means that the hybrid algorithm is suited for settings where LO-level expert labels are either not available or more expensive than the HI-level labels. This is exactly the setting we analyze in the next section.

In Appendix B.1, we compare hg-DAgger/Q with hierarchical RL (h-DQN, Kulkarni et al., 2016), concluding that h-DQN, even with significantly more LO-level samples, fails to reach success rate comparable to hg-DAgger/Q. Flat Q-learning also fails in this setting, due to a long planning horizon and sparse rewards (Mnih et al., 2015).

6.2. Hierarchically Guided IL / RL vs Hierarchical RL: Comparison on Montezuma's Revenge

Task Overview. Montezuma's Revenge is among the most difficult Atari games for existing deep RL algorithms, and is a natural candidate for hierarchical approach due to the sequential order of subtasks. Figure 3 (left) displays the environment and an annotated sequence of subgoals. The four designated subgoals are: go to bottom of the right stair, get the key, reverse path to go back to the right stair, then go to open the door (while avoiding obstacles throughout).

The agent is given a pseudo-reward of 1 for each subgoal

completion and -1 upon loss of life. We enforce that the agent can only have a single life per episode, preventing the agent from taking a shortcut after collecting the key (by taking its own life and re-initializing with a new life at the starting position, effectively collapsing the task horizon). Note that for this setting, the actual game environment is equipped with two positive external rewards corresponding to picking up the key (subgoal 2, reward of 100) and using the key to open the door (subgoal 4, reward of 300). Optimal execution of this sequence of subgoals requires more than 200 primitive actions. Unsurprisingly, flat RL algorithms often achieve a score of 0 on this domain (Mnih et al., 2015; 2016; Wang et al., 2016).

hg-DAgger/Q versus h-DQN. Similar to the maze domain, we use DDQN with prioritized experience replay at the LO level of hg-DAgger/Q. We compare its performance with h-DQN using the same neural network architecture as Kulkarni et al. (2016). Figure 3 (middle) shows the learning progression of our hybrid algorithm. The HI-level horizon $H_{\text{HI}} = 4$, so meta-controller is learnt from fairly few samples. Each episode roughly corresponds to one Label_{HI} query. Subpolicies are learnt in the order of subgoal execution as prescribed by the expert.
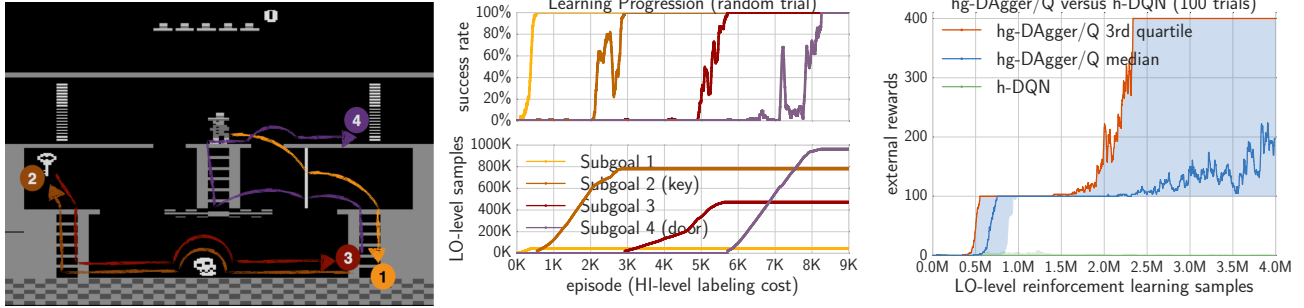
Figure 3. Montezuma's revenge: hg-DAgger/Q versus h-DQN. (Left) Screenshot of Montezuma's Revenge in black-and-white with color-coded subgoals. *(Middle)* Learning progression of hg-DAgger/Q in solving the first room of Montezuma's Revenge for a typical successful trial. Subgoal colors match the left pane; success rate is the fraction of times the LO-level RL learner achieves its subgoal over the previous 100 attempts. *(Right)* Learning performance of hg-DAgger/Q versus h-DQN (median and inter-quartile range).

We introduce a simple modification to Q-learning on the LO level to speed up learning: the accumulation of experience replay buffer does not begin until the first time the agent encounters positive pseudo-reward. During this period, in effect, only the meta-controller is being trained. This modification ensures the reinforcement learner encounters at least some positive pseudo-rewards, which boosts learning in the long horizon settings and should naturally work with any off-policy learning scheme (DQN, DDQN, Dueling-DQN). For a fair comparison, we introduce the same modification to the h-DQN learner (otherwise, h-DQN failed to achieve any reward).

To mitigate the instability of DQN (see, for example, learning progression of subgoal 2 and 4 in Figure 3, middle), we introduce one additional modification. We terminate training of subpolicies when the success rate exceeds 90%, at which point the subgoal is considered learned. Subgoal success rate is defined as the percentage of successful subgoal completions over the previous 100 attempts.

Figure 3 (right) shows the median and the inter-quartile range over 100 runs of hg-DAgger/Q and hg-DQN.[7] The LO-level sample sizes are not directly comparable with the middle panel, which displays the learning progression for a random successful run, rather than an aggregate over multiple runs. In all of our experiments, the performance of the imitation learning component is stable across many different trials, whereas the performance of the reinforcement learning component varies substantially. Subgoal 4 (door) is the most difficult to learn due to its long horizon whereas subgoals 1–3 are mastered very quickly, especially compared to h-DQN. Our algorithm benefits from hierarchical guidance and accumulates experience for each subgoal only within the relevant part of the state space, where the subgoal is part of an optimal trajectory. In contrast, h-DQN

may pick bad subgoals and the resulting LO-level samples then "corrupt" the subgoal experience replay buffers and substantially slow down convergence.[8]

The number of HI-level labels in Figure 3 (middle) can be further reduced by using a more efficient RL procedure than DDQN at the LO level. In the specific example of Montezuma's Revenge, the actual human effort is in fact much smaller, since the human expert needs to provide a sequence of subgoals only once (together with simple subgoal detectors), and then HI-level labeling can be done automatically. The human expert only needs to understand the high level semantics, and does not need to be able to play the game.

7. Conclusion

We have presented *hierarchical guidance* framework and shown how it can be used to speed up learning and reduce the cost of expert feedback in hierarchical imitation learning and hybrid imitation–reinforcement learning.

Our approach can be extended in several ways. For instance, one can consider weaker feedback such as preference or gradient-style feedback (Fürnkranz et al., 2012; Loftin et al., 2016; Christiano et al., 2017), or a weaker form of imitation feedback, only saying whether the agent action is correct or incorrect, corresponding to bandit variant of imitation learning (Ross et al., 2011).

Our hybrid IL/RL approach relied on the availability of a subgoal termination predicate indicating when the subgoal is achieved. While in many settings such a termination predicate is relatively easy to specify, in other settings this predicate needs to be learned. We leave the question of learning the termination predicate, while learning to act from reinforcement feedback, open for future research.

---

[7]In Appendix B, we present additional plots, including 10 best runs of each algorithm, subgoal completion rate over 100 trials, and versions of Figure 3 (middle) for additional random instances.

[8]In fact, we further reduced the number of subgoals of h-DQN to only two initial subgoals, but the agent still largely failed to learn even the second subgoal (see the appendix for details).

Acknowledgments

References

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *ICML*, pp. 1. ACM, 2004.

Andreas, J., Klein, D., and Levine, S. Modular multitask reinforcement learning with policy sketches. In *ICML*, 2017.

Chang, K.-W., Krishnamurthy, A., Agarwal, A., Daume III, H., and Langford, J. Learning to search better than your teacher. In *ICML*, 2015.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. In *NIPS*, 2017.

Daumé, H., Langford, J., and Marcu, D. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

Dayan, P. and Hinton, G. E. Feudal reinforcement learning. In *NIPS*, 1993.

Dietterich, T. G. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res.(JAIR)*, 13(1):227–303, 2000.

El Asri, L., Schulz, H., Sharma, S., Zumer, J., Harris, J., Fine, E., Mehrotra, R., and Suleman, K. Frames: a corpus for adding memory to goal-oriented dialogue systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pp. 207–219, 2017.

Fruit, R. and Lazaric, A. Exploration–exploitation in mdps with options. *arXiv preprint arXiv:1703.08667*, 2017.

Fürnkranz, J., Hüllermeier, E., Cheng, W., and Park, S.-H. Preference-based reinforcement learning: a formal framework and a policy iteration algorithm. *Machine learning*, 89(1-2):123–156, 2012.

Hausknecht, M. and Stone, P. Deep reinforcement learning in parameterized action space. In *ICLR*, 2016.

He, R., Brunskill, E., and Roy, N. Puma: Planning under uncertainty with macro-actions. In *AAAI*, 2010.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., et al. Deep q-learning from demonstrations. In *AAAI*, 2018.

Ho, J. and Ermon, S. Generative adversarial imitation learning. In *NIPS*, pp. 4565–4573, 2016.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NIPS*, pp. 3675–3683, 2016.

Loftin, R., Peng, B., MacGlashan, J., Littman, M. L., Taylor, M. E., Huang, J., and Roberts, D. L. Learning behaviors via human-delivered discrete feedback: modeling implicit feedback strategies to speed up learning. In *AAMAS*, 2016.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *ICML*, pp. 1928–1937, 2016.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Overcoming exploration in reinforcement learning with demonstrations. In *ICRA*, 2017.

Peng, B., Li, X., Li, L., Gao, J., Celikyilmaz, A., Lee, S., and Wong, K.-F. Composite task-completion dialogue policy learning via hierarchical deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 2231–2240, 2017.

Ross, S. and Bagnell, J. A. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

Ross, S., Gordon, G. J., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, pp. 627–635, 2011.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International Conference on Machine Learning*, pp. 1312–1320, 2015a.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015b.

Shalev-Shwartz, S. et al. Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2012.

Sun, W., Venkatraman, A., Gordon, G. J., Boots, B., and Bagnell, J. A. Deeply aggrevated: Differentiable imitation learning for sequential prediction. *arXiv preprint arXiv:1703.01030*, 2017.

Sutton, R. S., Precup, D., and Singh, S. P. Intra-option learning about temporally abstract actions. In *ICML*, volume 98, pp. 556–564, 1998.

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Syed, U. and Schapire, R. E. A game-theoretic approach to apprenticeship learning. In *NIPS*, pp. 1449–1456, 2008.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *AAAI*, volume 16, pp. 2094–2100, 2016.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. Dueling network architectures for deep reinforcement learning. In *ICML*, pp. 1995–2003, 2016.

Zheng, S., Yue, Y., and Lucey, P. Generating long-term trajectories using deep hierarchical networks. In *NIPS*, 2016.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.