Hierarchical Imitation and Reinforcement Learning:

https://arxiv.org/pdf/1803.00590.pdf

Paper Summary Notes

1 Main Ideas, High Level Assumptions

Central Question of the Paper: *When experts are available, how can we most effectively leverage their feedback?*

- **Context**:
  - standard imitation learning can solve long-horizon sparse rewards, but may require lots of expert behaviour
  - hierarchy helps long horizon problems by decomposing into sub-problems and acting recursively optimally
  - *Hierarchical Guidance*: framework where high level expert focuses (guides) low-level learner.
    - High level expert ensures low-level learning only occurs when necessary, and only over relevant parts of state space, thereby speeding up learning.
    - Omitting feedback on already mastered subtasks reduces expert effort.
- **Goals**:
  - reduce *expert cost* which is defines how much effort the expert needs to make the system work
  - decompose problem into hierarchy to reduce cost of exploration
  - faster learning that standard hierarchical RL, and more label-efficient than normal Imitation learning

2 Formulation

2.1 Hierarchical Rl Basics

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

- *low level trajectory*: $\langle s_t, a_t \rangle$ tuples of low level actions taken for a given goal g
- *high level trajectory*: $\langle s_t, g_t \rangle$ tuples of states and goals, where s_{t+1} is given by the terminal state from low level policy acting from state s_t under goal g_t
- *full trajectory*: concatenation of all low level actions throughout the entire hierarchical trajectory

2.2 Expert Feedback Supervision Types

- **Hierarchical Demonstration** HierDemo(s): expert executes hierarchical policy starting from s and returns the hierarchical trajectory (passive IL)
- **High Level Labelling** LabelHigh(τ_{high}): expert provides next subgoal at each state s of a high level trajectory giving labelled dataset of tuples $\langle s, g, \rangle$
- **Low Level Labelling** LabelLow(τ_{low}): expert provides next primitive actions towards subgoal g at each state s of a low level trajectory giving labelled dataset of tuples $\langle s, a, \rangle$

- **Full Level Labelling** LabelFull(τ_{full}): expert labels the agent full trajectory ignoring hierarchical structure, giving labelled dataset of tuples $\langle s, a, \rangle$
- **Low level Inspection** InspectLow($\tau_{low}; g$): expert just verifies whether subgoal g was accomplished in the trajectory
- **Full Inspection** InspectFull(τ_{full}): expert verifies whether agent's overall goal was accomplished

Inspect ops are *lazy* version of label op, requiring less effort (easier to say whether or not robot is in elevator than providing an actual path to the elevator)

idea: if hierarchical trajectory agrees with expert on high level, and low level trajectories pass inspection, the the resulting full trajectory must also pass full inspection (hierarchical policy need not always agree with expert execution at low level to succeed in the overall task)

2.3 DAgger Algorithm Background

[1]

Dagger (Dataset aggregation) is an iterative reinforcement learning algorithm in the imitation learning space, that aims to trains a deterministic policy that achieves good performance in an online fashion (contrary to passive imitation learning which does rather poorly)

"DAGGER proceeds by collecting a dataset at each iteration under the current policy and trains the next policy under the aggregate of all collected datasets. The intuition behind this algorithm is that over the iterations, we are building up the set of inputs that the learned policy is likely to encounter during its execution based on previous experience (training iterations). This algorithm can be interpreted as a Follow-The-Leader algorithm in that at iteration n we pick the best policy π_{n+1} in hindsight"

3 Hierarchical Guidance

Hierarchical Guidance is paradigm where feedback from high-level expert guides low level learner in 2 ways:

1. high level expert ensures that low level expert is only queried when necessary (when subtasks have not yet been mastered)
2. low level learning is limited to relevant parts of the state space

[1] DAgger Algorithm is used as a subroutine for the algorithms in the current paper.

3.1 Hierarchical Behaviour Cloning

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

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

- expert provides hierarchical demonstrations, with both low and high level trajectories, getting dataset for supervised learning which we can optimize via sgd
- then we train subpolicies predicting action from state, and meta controller predicting goal from state
- here hierarchical guidance is automatic since subpolicy demonstrations only occur In relevant parts of the state space
- but this is passive learning, we iterate on these ideas in the online learning context next

3.2 Hierarchically Guided Dagger

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

Initialize data buffers $\mathcal{D}_{\text{HI}} \leftarrow \emptyset$ and $\mathcal{D}_g \leftarrow \emptyset, g \in \mathcal{G}$
Run Hierarchical Behavioral Cloning (Algorithm 1)
    up to $t = T_{\text{warm-start}}$
for $t = T_{\text{warm-start}} + 1, \ldots, T$ **do**
    Get a new environment instance with start state s
    Initialize $\sigma \leftarrow \emptyset$
    repeat
        $g \leftarrow \mu(s)$
        Execute π_g, obtain LO-level trajectory τ
        Append (s, g, τ) to σ
        $s \leftarrow$ the last state in τ
    until end of episode
    Extract τ_{FULL} and τ_{HI} from σ
    if $\texttt{Inspect}_{\text{FULL}}(\tau_{\text{FULL}}) =$ *Fail* **then**
        $\mathcal{D}^\star \leftarrow \texttt{Label}_{\text{HI}}(\tau_{\text{HI}})$
        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 $\mathcal{D}^\star$:
            if $\texttt{Inspect}(\tau_h; g_h) =$ *Fail* **then**
                Append $\mathcal{D}_{g_h} \leftarrow \mathcal{D}_{g_h} \cup \texttt{Label}_{\text{LO}}(\tau_h; g_h)$
                break
        Append $\mathcal{D}_{\text{HI}} \leftarrow \mathcal{D}_{\text{HI}} \cup \mathcal{D}^\star$
    Update subpolicies $\pi_g \leftarrow \texttt{Train}(\pi_g, \mathcal{D}_g)$ for all g
    Update meta-controller $\mu \leftarrow \texttt{Train}(\mu, \mathcal{D}_{\text{HI}})$

- start by pretraining using clonging alg 1
- learner executes hierarchical policy, including choosing a subgoal, executing low level trajectory, and terminating according to end signal
- expert only gives feedback if the agent fails entire task
- if failed, expert first gets the subgoals that should have been taken by using **LabelHigh**
- only perform labelling on low level if meta controller chooses correct subgoal, but subpolicy fails
- since all preceding subgoals were chosen and executed correctly, and current subgoal is also correct, low level learning is in the relevant part of state space, and since subgoal failed, this is where we need to learn

3.3 Hierarchically Guided Imitation Learning

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

input Function $\texttt{pseudo}(s; g)$ providing the pseudo-reward
input Predicate $\texttt{terminal}(s; g)$ indicating the termination of g
input Annealed exploration probabilities $\epsilon_g > 0$, $g \in \mathcal{G}$
Initialize data buffers $\mathcal{D}_{\text{HI}} \leftarrow \emptyset$ and $\mathcal{D}_g \leftarrow \emptyset$, $g \in \mathcal{G}$
Initialize subgoal Q-functions Q_g, $g \in \mathcal{G}$
for $t = 1, \ldots, T$ **do**
  Get a new environment instance with start state s
  Initialize $\sigma \leftarrow \emptyset$
  repeat
    $s_{\text{HI}} \leftarrow s$, $g \leftarrow \mu(s)$ and initialize $\tau \leftarrow \emptyset$
    repeat
      $a \leftarrow \epsilon_g$-greedy(Q_g, s)
      Execute a, next state $\tilde{s}$, $\tilde{r} \leftarrow \texttt{pseudo}(\tilde{s}; g)$
      Update Q_g: a (stochastic) gradient descent step on a minibatch from $\mathcal{D}_g$
      Append $(s, a, \tilde{r}, \tilde{s})$ to τ and update $s \leftarrow \tilde{s}$
    until $\texttt{terminal}(s; g)$
    Append (s_{HI}, g, τ) to σ
  until end of episode
  Extract τ_{FULL} and τ_{HI} from σ
  if $\texttt{Inspect}_{\text{FULL}}(\tau_{\text{FULL}}) = \textit{Fail}$ **then**
    $\mathcal{D}^\star \leftarrow \texttt{Label}_{\text{HI}}(\tau_{\text{HI}})$
    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 $\mathcal{D}^\star$:
      Append $\mathcal{D}_{g_h} \leftarrow \mathcal{D}_{g_h} \cup \tau_h$
    Append $\mathcal{D}_{\text{HI}} \leftarrow \mathcal{D}_{\text{HI}} \cup \mathcal{D}^\star$
  else
    Append $\mathcal{D}_{g_h} \leftarrow \mathcal{D}_{g_h} \cup \tau_h$ for all $(s_h, g_h, \tau_h) \in \sigma$
  Update meta-controller $\mu \leftarrow \texttt{Train}(\mu, \mathcal{D}_{\text{HI}})$

- high level expert provides decomposition and pseudo-reward for each subgoal and picks a new subgoal at each step
- *hierarchical guidance impacts how the low level learner accumulates experience*:
  - as longas meta controller subgoal g agrees with expert, agent's experience is added to buffer
  - if meta controller selects wrong subgoal, no experience is there, ensuring buffer contains only relevant data
  - requires defining a success(s, g) signal determining if subgoal g is complete and pseudo reward indicator

4 Results

4.1 Cost Savings

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*

$$TC^I_{\text{FULL}} + \left(\log_2|\mathcal{M}| + |\mathcal{G}_{opt}|\log_2|\Pi_{\text{LO}}|\right)\left(C^L_{\text{HI}} + H_{\text{HI}}C^I_{\text{LO}}\right) + \left(|\mathcal{G}_{opt}|\log_2|\Pi_{\text{LO}}|\right)C^L_{\text{LO}}, \quad (1)$$

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*

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

Therefore the cost ratio is $\propto \frac{H_{\text{HI}}+H_{\text{LO}}}{H_{\text{HI}}H_{\text{LO}}}$

4.2 Maze

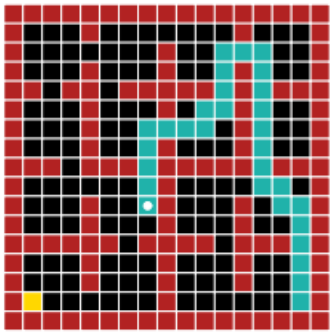
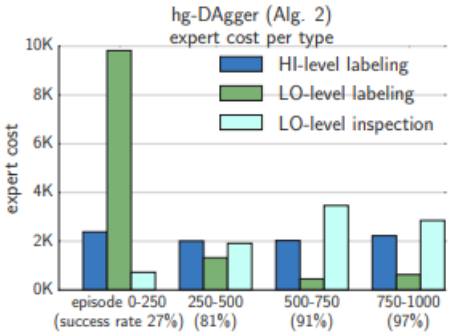
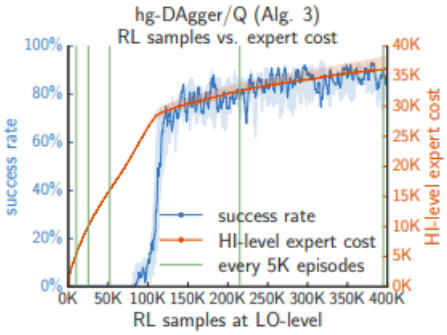


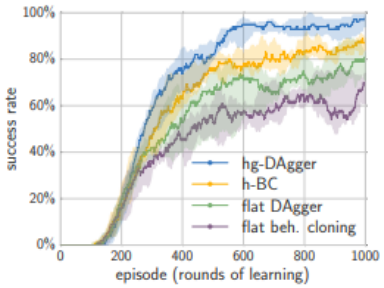
Table 1. Network Architecture—Maze Domain

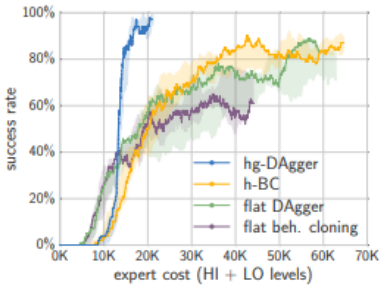
1: Convolutional Layer	32 filters, kernel size 3, stride 1
2: Convolutional Layer	32 filters, kernel size 3, stride 1
3: Max Pooling Layer	pool size 2
4: Convolutional Layer	64 filters, kernel size 3, stride 1
5: Convolutional Layer	64 filters, kernel size 3, stride 1
6: Max Pooling Layer	pool size 2
7: Fully Connected Layer	256 nodes, relu activation
8: Output Layer	softmax activation (dimension 4 for subpolicy, dimension 5 for meta-controller)

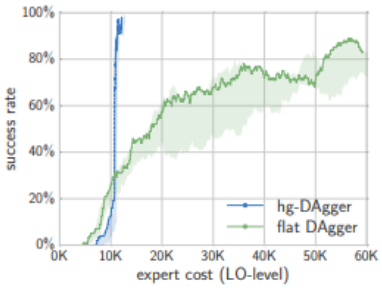
- Primitive Actinos: left, right, up, down
- Meta Actions: go to north, south, west, east room, go to target





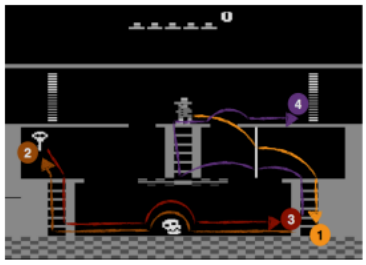


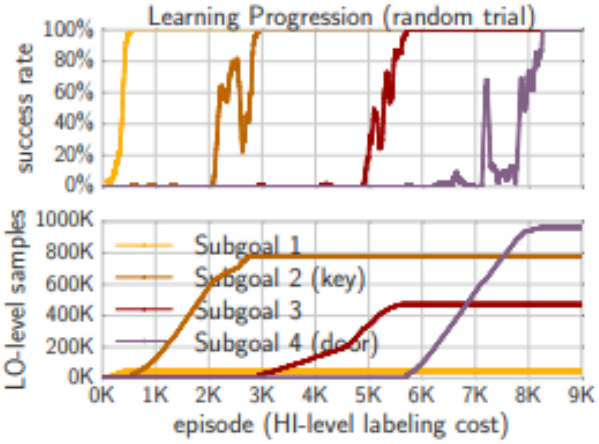


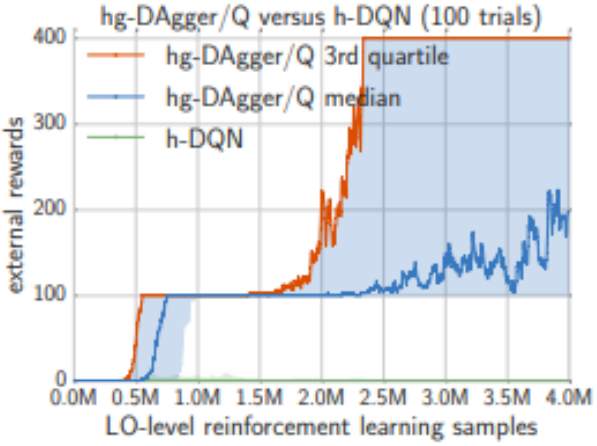


- *Hierarchical Version Outperforms Flat*
- *Using guidance saves alot of agent cost due to hierarchy*
- *number of High level labels rapdily icnreases initially and then flattens out after the learner becomes more successful*

4.3 Montezuma Revenge







- *The algo is able to get non zero rewards on this challenging first room task*
- *h-sqn may be picking the wrong subgoals, and hence getting noisy experience in the replay buffer, making it difficult to convergence*