Learning Hierarchical Control Policies from Demonstrations

1 What is a hierarchical control policy

In a hierarchical control policy, the lowest level is a generalization of a standard control policy $\pi(a|s)$: it also takes in a signal h generated by the higher level, $\pi_{\text{low}}(a|s,h)$. The hint signal helps the low level planning by saving effort in processing state information, acting both like a short-term memory and like a short-term subgoal. The signal changes slowly, and represents reuse of processing efforts between adjacent time steps.

The high level doesn't need to output a control signal a, only a meta-control signal h, according to a meta-policy $\pi_{\text{high}}(h|s)$. It can plan on a longer timescale, and reuse the low-level planning efforts between distant time steps.

In our formulation, the low level decides when to terminate its control and return it to the high level, like a return from a function call. This is in contrast to the DeepMind paper, where the low level terminates every 2 steps.

2 Why use a hierarchical control policy

2.1 Policy complexity reduction

The reinforcement learning problem is hard, partly due to the complexity of the solution space - all (stochastic) mappings of states into actions. As we move toward harder problems, with richer state and action spaces, this problem very quickly becomes intractable.

The only way to scale up without an inhibiting amount of samples and computation, is to have an inductive bias - a prior assumption about the structure of the problem. The hierarchical structure is very appealing, also for other reasons, but mostly because it can greatly reduce the complexity of the policy. In principle, it can allow scaling the policy up to arbitrary system complexity and time horizons, and down to arbitrary spacial and temporal precision.

2.2 Sensor/actuator complexity reduction

One aspect of the reduction in complexity of the control function is the reduction in the dimensionality of its input and its output. This has the benefit of

simplifying sensors and actuators, which is important both computationally and for the practical engineering aspect.

In the DeepMind paper the low level achieves a hard-coded reduction in input complexity, by only taking part of the input observation. We're taking a more principled approach, by learning an emergent input/output complexity reduction as part of our learning algorithm.

2.3 Specialization and reusability

As mentioned above, the low level specializes in short-term planning, while the high level specializes in long-term planning. This is also reflected in how each level reuses the other: the low level reuses the high-level signal h in multiple adjacent time steps, creating a short-term correlation that allows it to focus on a subgoal. The high level, on the other hand, reuses the low-level primitives in multiple distant time steps, creating a long-term correlation between repeatable behaviors in similar states.

In the DeepMind paper, the high-level *modulation* signal h is continuous, and the low-level policy parameters are shared for all values of h. We have a discrete *switching* signal, that indexes a different control primitive for each h, like a named subroutine. This allows these policies to be archived and shared discretely between various systems.

All but the highest-level primitives can be reused for new unseen tasks. The highest level only needs to decide how to best use these precomputed options to perform the new task.

All but the lowest-level primitives can be reused for new unseen hardware. The lowest level only needs to decide how to implement these subgoals on the new hardware.

3 How to interpret a hierarchical control policy

3.1 As a structure / constraint on policy space

A hierarchical control policy has a particular structure, that reduces the effective volume of the solution space. However, it is not a special case of a flat control policy, since the high level can create temporal correlations.

For example, consider the task of repeatedly picking an object at the right end of a corridor and dropping it at the left end. With flat control, the robot needs at each time step to examine the state and figure out what it needs to do: move right, pick up, move left or drop. With hierarchical control, the high level switches between these subtasks, and the low level only needs to perform a primitive or detect its success.

The constraint on the low level comes from the simplified parameterization of the low-level primitives. The constraint on the high level comes from the limited number of low-level primitives.

3.2 As a modular control system

The division of planning effort into hierarchical levels is a form of modularization. This allows different parts of the computation to be distributed and run at different times and locations. The only requirement is a slowly changing interface between the components.

3.3 As a structured memory state representation

The persistent high-level signal can be viewed as a memory state. This memory state has a structure which is distributed between the levels, with higher-level features more persistent than lower-level ones.

Memory-based policies are not necessary for optimal control in fully-observable environments. However, they become useful in the following cases:

- In partially-observable environments, where past observations are useful in inferring the hidden state;
- To model controllers that have latent variables, such as human demonstrations;
- When memory is cheaper than perception, e.g. when persistent features, that can be kept constant in memory with very little processing, are useful in reducing the complexity of processing each new observation.

The space of general memory-based controllers is huge and contains many local optima, however the constraints mentioned above allow optimization over a useful subset of this space.

3.4 As a stack of function calls

The flow of control in a hierarchical policy follows the same pattern as in digital computer programs. It starts at the highest level and goes down the hierarchy. When each level finishes executing its policy, it terminates and returns control to the level above it. The call stack represents the structured memory of the controller, with each frame a control signal to the level below.

3.5 As a renormalization operator

Humans can remember arbitrarily far histories and plan to arbitrarily long horizons. This implies that their highest levels are a fixed point of the "next higher level" operator. In statistical physics, this kind of self-similarity is captured by the notion of renormalization. The properties of the hierarchy-construction operator can then be studied by expansion in the neighborhood of the renormalization operator's fixed point.

4 How to learn a hierarchical control policy

In the following we present algorithms for learning the low-level primitives from demonstrations, by using a naive model for the high level. A similar algorithm can be used to learn while planning, rather than from demonstrations.

Each such algorithm can be viewed as:

- A clustering algorithm, with the learned primitives as the centroids and the cluster assignment as the high-level control. Unsupervised algorithms for reinforcement learning are gaining a lot of attention in recent months.
- A maximum-likelihood optimization of a generative model. This has connections to variational inference, which has been of major interest in the past couple of years.
- Optimal control under the constraints of a communication model. This connects directly to a growing community interested in information-theoretical aspects of dynamical systems.

4.1 Clustering subtask demonstrations

Parameters:

- $P \in \Delta([k])$
- $\forall h \in [k]$, $\theta_h \in \Theta$, inducing $\pi_h : \mathcal{S} \to \Delta(\mathcal{A})$

Generative model:

- For $1 \leqslant i \leqslant m$, independently
  - Draw $h_i \sim P$ (latent)
  - Draw $\rho_i \sim \mathbb{P}_{h_i}$ (observed)

Log likelihood of observed given latent:

$$\mathcal{L}(\rho|h) = \sum_i \log \mathbb{P}_{h_i}(\rho_i) = \sum_i \sum_{t=0}^{T_i-1} \log \pi_{h_i}(a_{i,t}|s_{i,t}) + \text{const}$$

Update iteration:

$$q_i(h) = \frac{P(h)\,\mathbb{P}_{\pi_h}(\rho_i)}{\sum_{h'} P(h')\,\mathbb{P}_{\pi_{h'}}(\rho_i)} = \frac{P(h)\prod_{t=0}^{T_i-1}\pi_h(a_{i,t}|s_{i,t})}{\sum_{h'} P(h')\prod_{t=0}^{T_i-1}\pi_{h'}(a_{i,t}|s_{i,t})}$$

$$P(h) = \frac{1}{m}\sum_i q_i(h)$$

$$\delta\theta_h = \alpha\frac{1}{m}\sum_i q_i(h)\sum_{t=0}^{T_i-1}\partial_{\theta_h}\log\pi_h(a_{i,t}|s_{i,t})$$

4.2 Segmenting and clustering task demonstrations

Same parameters.
Same generative model, except output first T steps of $(\rho_i)_i$ concatenated (latent segmentation).
Same log likelihood of observed given latent (since segmentation is revealed).
Update iteration (with $\rho_{\leqslant t} = (s_0, a_0, \ldots, s_t)$, $\rho_{>t} = (a_t, s_{t+1}, a_{t+1}, \ldots, s_T)$, h_t the hint for a_t, and $\perp$ the termination action):

$$
\begin{aligned}
q_{t,h} &\propto \mathbb{P}(\rho, h_t = h) = \mathbb{P}(\rho_{\leqslant t}, h_t = h)\, \mathbb{P}(\rho_{>t} | s_t, h_t = h) \\
b_{t,h} &\propto \mathbb{P}(\rho, h_t = h, \perp \text{ at } s_{t+1}) = \sum_{h'} \mathbb{P}(\rho, h_t = h, \perp \text{ at } s_{t+1}, h_{t+1} = h') \\
&= \sum_{h'} \mathbb{P}(\rho_{\leqslant t}, h_t = h) \pi_h(a_t|s_t) p(s_{t+1}|s_t, a_t) \psi_h(s_{t+1}) P(h'|h)\, \mathbb{P}(\rho_{>t+1} | s_{t+1}, h_{t+1} = h') \\
\mathbb{P}(\rho_{\leqslant t+1}, h_{t+1} = h') &= \sum_{h} \mathbb{P}(\rho_{\leqslant t}, h_t = h) \pi_h(a_t|s_t) p(s_{t+1}|s_t, a_t) (\psi_h(s_{t+1}) P(h'|h) + (1 - \psi_h(s_{t+1})) \delta_{h,h'}) \\
\mathbb{P}(\rho_{>t} | s_t, h_t = h) &= \pi_h(a_t|s_t) p(s_{t+1}|s_t, a_t) \sum_{h'} (\psi_h(s_{t+1}) P(h'|h) + (1 - \psi_h(s_{t+1})) \delta_{h,h'})\, \mathbb{P}(\rho_{>t+1} | s_{t+1}, h_t = h') \\
\delta\theta_h &= \alpha \sum_{t=0}^{T-1} (q_{t,h}\, \partial_{\theta_h} \log \pi_h(a_t|s_t) + b_{t,h}\, \partial_{\theta_h} \log \psi_h(s_t))
\end{aligned}
$$

$p(s_{t+1}|s_t, a_t)$ can be dropped due to the normalization.