Matching options to tasks using Option-Indexed Hierarchical RL

S Aniruddha
Data Science Dual Degree Student
Guide: Prof. Balaraman Ravindran
(Collaboration with Google Research, India)

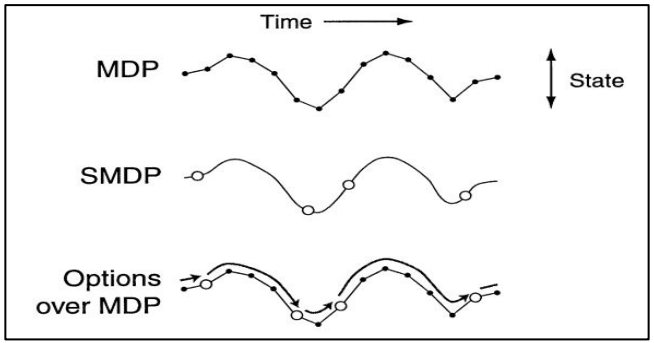
Agenda

- Problem setting
- Our Approach
- Experiments & Results

Problem setting

Options Framework and HRL

- The Options Framework in HRL breaks down overall goals into a combination of options or simpler tasks and associated policies, providing abstraction in the action space .
- An option (or macro action) is a generalization of the concept of action.
- A set of options defined over an MDP constitutes a Semi-Markov decision process (SMDP), which are MDPs where the time between actions is not constant but a random variable.
- Examples in a kitchen setting: picking up a bowl, picking up an egg, cracking an egg, etc.



Motivation

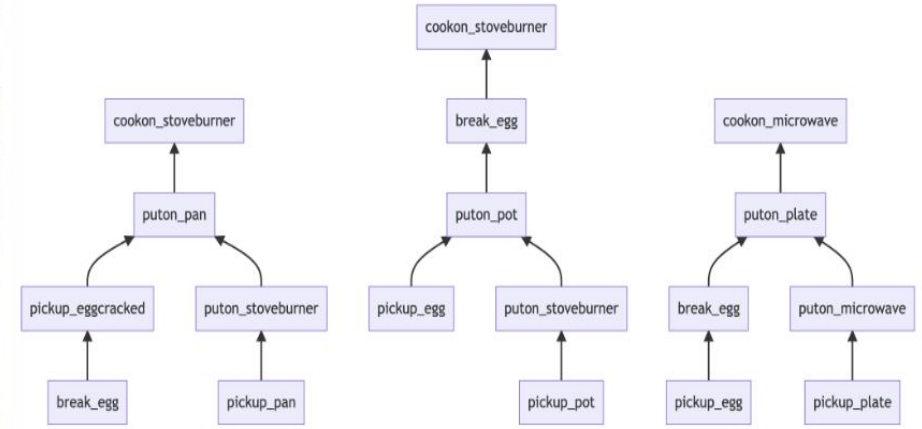
- Suppose an agent encounters a new task in a new environment.
- Without prior experience, the agent will need to explore and interact with the environment using primitive actions as it learns to navigate and accomplish various goals.
- For the agent to effectively use hierarchical reinforcement learning, it has to first partition the overall task into appropriate subtasks.
- It then has to learn how to accomplish the subtasks and how to sequence them together in an appropriate manner. In general, this is a very expensive process due to the combinatorial nature of the space of subtasks.
- Instead, we ask the following question: Suppose the agent has significant experience with a range of related environments and tasks; how can it leverage its prior experience to accomplish the task on hand in an efficient manner?

Option Indexing

- For a life-long learning agent performing multiple related tasks sequentially in a given domain, it is critical for the hierarchical policy to only explore necessary parts of the search space.
- **Option indexing:** Efficient retrieval of relevant options for performing a new HRL task, from a large library of pre-learned options.
- OI-HRL aims to accomplish new goals in a given domain by:
  - fetching the relevant options from the option library (prior knowledge of the continual learning agent).
  - constructing a hierarchical policy using the fetched options.
- Key ideas:
  - Learn an affinity score between a given environment state and the options relevant to goals achievable in that environment (measure of relevancy).
  - a meta-training loop for learning this affinity score by solving a series of related HRL tasks in a given problem domain.

Example – Kitchen Robot

- A kitchen with various food items, utensils and appliances.
- An agent can accomplish various goals in this kitchen like cracking an egg, cooking on a stove, making an omelette.
- Some goals (such as making an omelette) depend upon other goals having been achieved first, in a specific order.
- A task refers to the accomplishment of a specific end goal: a task is said to be completed when its associated goal has been achieved.
- The conditions that need to be satisfied before a particular goal can be achieved are called preconditions of the task.
- Some task can be completed in multiple ways. Each such way to complete is called a task variant.



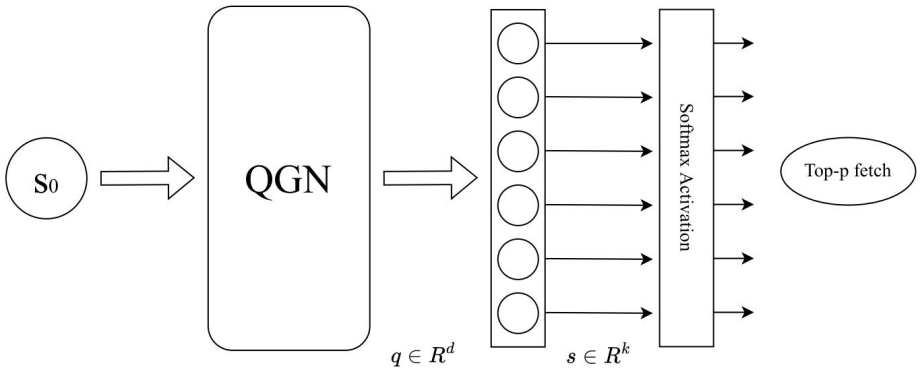
Some Notation and Terminology

- Task variant: $t_{i,j}$ ($i \in \{1, ..., n\}$ & $j \in \{1, ..., m_i\}$)
- Preconditions of a task variant $t_{i,j}$: $C(t_{i,j}) \subset \{t_{i,j}\}$
- Task (set of all task variants that achieve the same goal): $t_i = \{t_{i,1}, ..., t_{i,mi}\}$
- Set of tasks: $T = \{t_1, \ldots, t_n\}$
- When an environment is initialized, a goal task t_i is chosen from T, and a particular task variant $t_{i,j}$ is chosen from t_i. The chosen goal task is used to get an MDP $M(t_i) = (S, A, R(t_i), P)$.
- Base Tasks: Tasks that don't have any preconditions ($|C(t_{i,j})| = 0$). As a design choice, base tasks have only 1 variant ($|C(t_{i,j})| = 0 \Rightarrow m_i = 1$). A task that is not a base task is known as a composite task.
- Set of base tasks: $B = \{t_{i,1}, ..., t_{i,mi}\}$, $|B| = k$.
- The continual learning agent has a set of pretrained option policies $O = \{o_1, \ldots, o_k\}$ for completing each base task.
- Reward function is sparse with the reward being non-zero only when the task t_i is completed.

Approach

Option Retrieval Architecture

- The Query Generation Network (QGN) generates a query $q \in R^d$ based on the objects initially present in the environment s_0.
- The index $\mathfrak{Ψ} \subset R^d \times O$ stores the options O together with a key vector $\Psi(O_i) \subset R^d$ for option O_i, i.e. $\mathfrak{Ψ} = \{(\Psi(O_i), O_i)\}$.
- The QGN is followed by a linear hidden layer in the architecture, whose weights represent the option indices. The subset of options retrieved consists of the top-p options based on $p = \text{Softmax}(q^T \Psi(O_i))$.



Option Indexing

We propose two distinct ways of meta-learning the parameters of the QGN and option embeddings $\Psi(O_i)$ - Pretrained Index and Learned Index.

1. **Pretrained Index:** Since our agent has considerable experience with a range of related environments and tasks, we have access to many valid sequences of options that accomplish different task variants during the meta-training phase of the OI-HRL. We can leverage this library of recipes to obtain a vector representation for each option, which can be used as the option key vectors. To demonstrate this idea, we construct a co-occurrence matrix of options from the recipes in the training data.
2. **Learned Index:** We meta-learn both the parameters of the QGN and option embeddings $\Psi(O_i)$ in an end-to-end manner.

Algorithm 2 Querying the Index

Inputs: Initial env. state s_0, Index $\Psi = \{(\psi(O_i), O_i)\}$, QGN $\mathcal{N}$
function SELECTOPTIONS(s_0, Ψ, $\mathcal{N}$)
    query $\mathbf{q} \leftarrow \mathcal{N}(s_0)$
    similarities $\mathbf{s} \leftarrow [\mathbf{q}^\intercal \psi(t_1), \mathbf{q}^\intercal \psi(t_2), \dots]^\intercal$
    retrieval probabilities $\hat{\mathbf{p}} \leftarrow \text{Softmax}(\mathbf{s})$
    retrieved option set $\widehat{O}$ s.t. $\widehat{O} \subset O$ is the smallest set
        that satisfies $\sum_{O_i \in \widehat{O}} \hat{\mathbf{p}}_i > p$ (top-p fetch, p is set to 0.9)
    return $\widehat{O}$, $\mathbf{p}$

Algorithm 1 Creating the co-occurrence matrix

function CREATECOOCCURRENCEMATRIX( )
    A = [ ]
    for task variant $t_{i,j}$, in train tasks **do**
        $\Lambda \leftarrow$ HRLSOLVER(O, $\mathcal{M}_{t_{i,j}}$)
        options $\leftarrow$ OptionsFromTrajectories(Λ)
        A.append(options)
    CoOccurrenceMatrix $\leftarrow A^\intercal A$
    CoOccurrenceMatrix $\leftarrow$ NormRows(CoOccurrenceMatrix)
    return CoOccurrenceMatrix

Algorithm 4 Meta Training

if PretrainIndex is True **then**
    CoOccurrenceMatrix ← createCoOccurrenceMatrix()
    $\Psi \leftarrow$ rows(CoOccurrenceMatrix)
for a number of train iterations **do**
    Sample train task variant $t_{i,j}$ and MDP $\mathcal{M}_{t_i}$
    $\widehat{O}$, $\mathbf{p} \leftarrow$ SELECTOPTIONS(s_0, Ψ, $\mathcal{N}$)
    $\Lambda \leftarrow$ HRLSOLVER($\widehat{O}$, $\mathcal{M}_{t_{i,j}}$)
    $\mathbf{y} \leftarrow$ TARGETFROMTRAJECTORIES(Λ)
    if PretrainIndex is True **then**
        Train $\mathcal{N}$ to minimize $\mathcal{L}(\mathbf{y}, \mathbf{p})$
    else
        Train $\mathcal{N}$ and ψ to minimize $\mathcal{L}(\mathbf{y}, \mathbf{p})$

Algorithm 5 Meta Testing

for a number of test task variants **do**
    Sample task $t_{i,j}$ and MDP $\mathcal{M}_{t_i}$
    $\widehat{O}$, $\mathbf{p} \leftarrow$ SELECTOPTIONS(s_0, Ψ, $\mathcal{N}$)
    Train HRL policy using $\widehat{O}$

Algorithm 3 Target from Trajectories

function TARGETFROMTRAJECTORIES(Λ)
    Let R = sum of rewards of all trajectories in Λ
    Init $\mathbf{y} = [0, 0, \dots\ k \text{ times}]$
    for $T_i = (o_{i,1}, o_{i,2}, \dots o_{i,n},\ r_i) \in \Lambda$ **do**
        for $o_{i,j} \in T_i$ **do**
            $\mathbf{y}[o_{i,j}] = \mathbf{y}[o_{i,j}] + \frac{r_i/n}{R}$
    return $\mathbf{y}$

Experiments & Results

Baselines and Environment

- To study the effect of selecting a subset of options, OI-HRL is compared with the following baselines:
  - HRL-N: Selects the set of options that correspond to task variants in the test task variant's recipe. This is an oracle selector that gives an upper bound on performance for OI-HRL.
  - HRL-N+K: same as HRL-N but with k extra (irrelevant) options (Noisy oracle).
  - HRL-FULL: Selects the entire library of options O.
- In each case, once the options are selected, an HRL is constructed using the fetched options. HRL-N and HRL-N+K uses privileged information about the actual recipe of the goal task variant, something that is not available to OI-HRL.
- We test our model in AI2THOR Kitchen environment.

Results

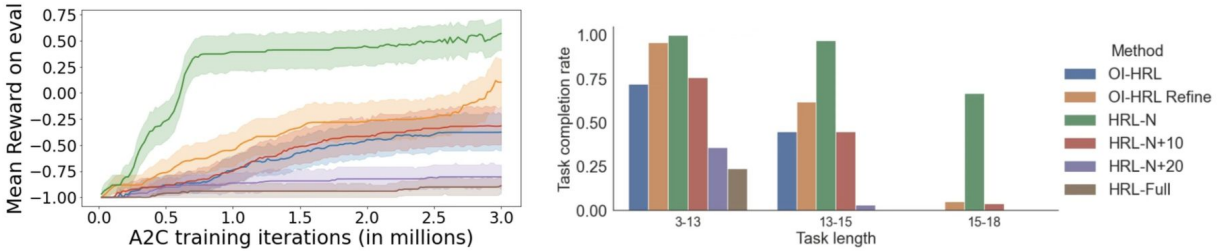


Figure 5: A2C policy training results using the Pretrained Index. Options are retrieved using OI-HRL and other baselines (a) Mean reward vs A2C training iterations plot demonstrating the gains in sample efficiency. (b) Asymptotic task completion rates of the A2C policy for different task lengths.

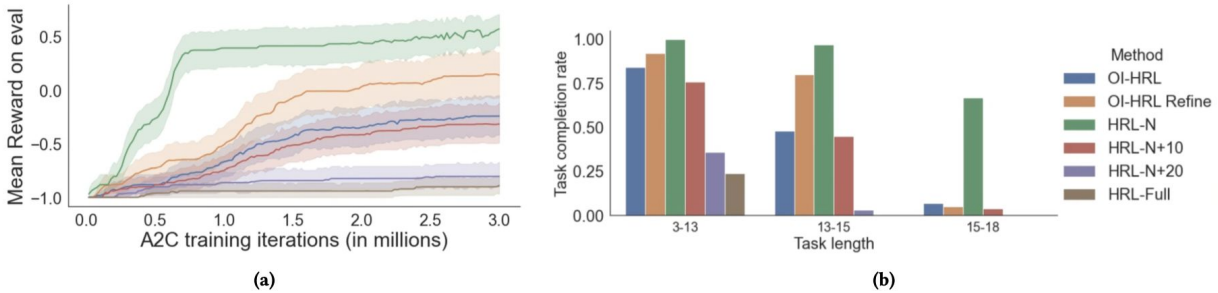


Figure 6: A2C policy training results with the Learned Index. Conventions are the same as Fig. 6

Thank You