Reinforcement Learning

- Sarath Chandar

12. Hierarchical RL

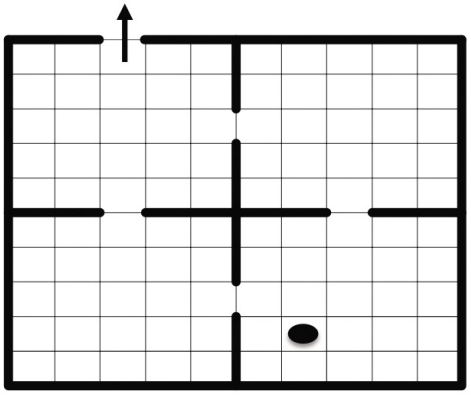
12. Hierarchical Reinforcement Learning

Hierarchical decomposition tackles complex problems by reducing them to smaller set of inter-related problems.

Hierarchical RL : aims to find good re-usable temporally extended actions that may also provide opportunities for state abstraction.

Advantages :-
1. Scalability
2. reusability / transfer learning
3. long-term credit assignment
4. Structured exploration.

Motivating example :- 4-room task



Reward −1 at each time step

actions − Stochastic

80% - intended action

20% - Stay in same place.

HRL solution: Policy to leave a room → Sub problem.

We can have a higher level RL agent that uses only 4 room-states.

Abstract actions:-

- Temporally extended actions.
- hide multi-step state transition and reward details from the time they are invoked until termination.

Ex: Room-leaving action

- Similar to 'macros' in programming.
- However, abstract action may be modelled with Stochastic transition fns and use stochastic policies.
- Sources of stochasticity:
  - Stoch. transition fn will make the sequence of states visited non-deterministic.
  - Sequence of rewards may vary even if the reward fn is deterministic.

- time taken to complete an abstract action may vary.

- ordinary actions are "primitive" actions. They terminate in one time step.

Semi-MDP:-

SMDP → MDP that includes abstract actions.

$N \geq 1$ is a random variable that denotes the number of time steps that an abstract action a takes to complete, starting in state 's' and terminating in state s'.

Transition fn: $T: S \times A \times S \times N \rightarrow [0,1]$

$$T(s, a, s', N) = Pr\{S_{t+N} = s' \mid S_t = s, a_t = a\}$$

The reward fn accumulates single-step rewards.

$$R : S \times A \times S \times N \to \mathbb{R}$$

$$R(s,a,s',N) = \mathbb{E}\left[\sum_{n=0}^{N-1} \gamma^n r_{t+n} \,\middle|\, s_t = s, a_t = a, s_{t+N} = s' \right]$$

Value of a state

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, s_t = s \right]$$

If the abstract action taken in state s persists for N steps, then

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\left(r_t + \gamma r_{t+1} + \cdots + \gamma^{N-1} r_{t+N-1} \right) + \left(\gamma^N r_{t+N} + \cdots \right) \,\middle|\, s_t = s \right]$$

$$V^{\pi}(s) = \sum_{s',N} T(s, \pi(s), s', N) \left[R(s, \pi(s), s', N) + \gamma^N V^{\pi}(s') \right]$$

Optimal value fn:

$$V^{*}(s) = \max_a \sum_{s',N} T(s,a,s',N) \left[R(s,a,s',N) + \gamma^N V^{*}(s') \right]$$

Q - state action value fn:

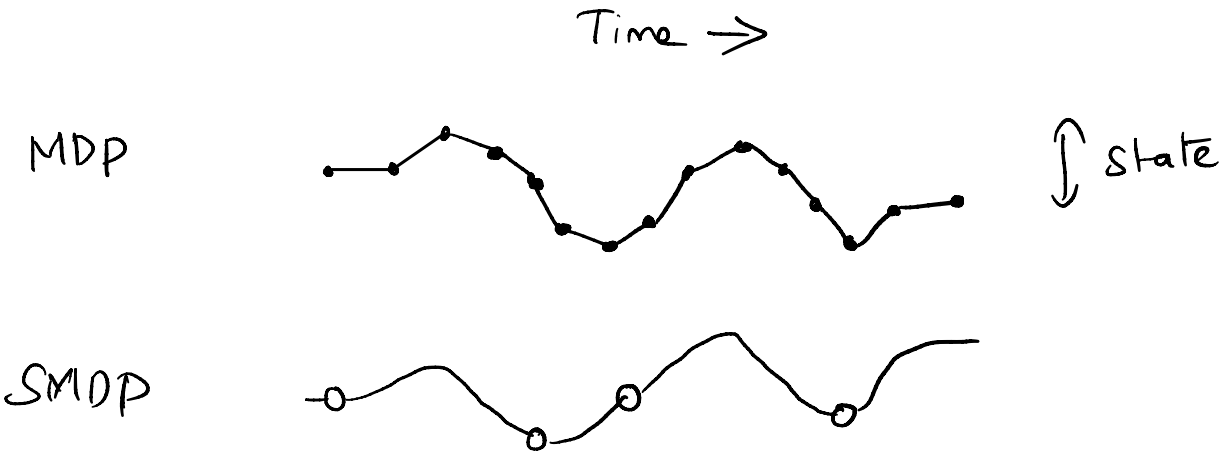
$$Q^{\pi}(s,a) = \sum_{s',N} T(s,a,s',N) \left[R(s,a,s',N) + \gamma^N Q^{\pi}(s', \pi(s')) \right]$$

Optimal Q fn:

$$Q^*(s,a) = \sum_{s',N} T(s,a,s',N)\left[R(s,a,s',N) + \gamma^N V^*(s')\right]$$

where $V^*(s') = \max_{a'} Q^*(s',a')$

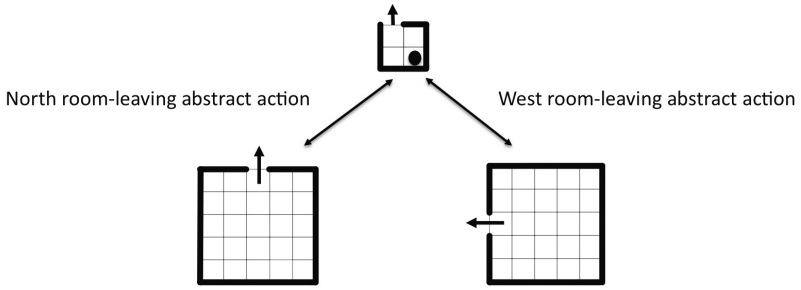
- All methods we saw for solving MDPs can be extended to SMDPs.



Structure:-

Abstract actions and SMDPs naturally lead to hierarchical structure. Abstract actions themselves can be policies from smaller SMDPs (or MDPs).

Task hierarchies :-



State abstraction :-

Benefit of decomposing a large MDP is that it will hopefully lead to state abstraction opportunities. An abstrated state space is smaller than the state space of an original MDP.

State-abstractions can be introduced

- to eliminate irrelevant variables.
- when abstract actions "funnel" the agent to a small subset of states.

Optimality:-

HRL cannot guarantee in general that a decomposed problem will necessarily yield the optimal soln. It depends on the problem, the quality of decomposition and the structure of the task hierarchy.

→ **hierarchically optimal**

Policies that are hierarchically optimal are ones that maximize the overall value fn. Consistent with the constraints imposed by the task hierarchy. Task's policy depends not only on its children's policies, but also on its context.

→ **Recursively optimal**

The policy for a parent task is optimal given the learnt policies of its children (context-free task's policy)

Recursively optimal policies cannot be better than hierarchically optimal policies which in turn cannot be better than flat optimal policies.

However, context free policies offer state abstractions / transfer learning better, which provides common macro actions to many other tasks.

Approaches to HRL :-

1) Options (Sutton et al. 1999) :

An option (in relation to an MDP $\langle S, A, T, R \rangle$) is a triple $\langle I, \pi, \beta \rangle$ in which $I \subseteq S$ is an initiation set, $\pi : S \times A \to [0,1]$ is a policy, and $\beta : S^+ \to [0,1]$ is a termination condition.

An option can be taken in state $s \in S$ of the MDP if $s \in I$. Once invoked, the option takes actions as determined by the stochastic policy π.

$\Rightarrow$ When option policies and termination depend on only the current state s, options are called Markov options.

$\Rightarrow$ If option policies and termination depend on the entire history sequence of states, actions, rewards since the option is initiated, it is called Semi-Markov options.

Note: Primitive action is an option.

a: $\langle I = S, \pi(s,a) = 1.0, \beta(s) = 1 \rangle$ for all $s \in S$.

It is possible to unify - the set of options and primitive actions!

Since options select actions, and actions are

just special kinds of options, it is possible for options to select other options.

SMDP Q-learning :-

$$Q(s,o) \leftarrow Q(s,o) + \alpha\left[r + \gamma^k \max_{a \in O} Q(s',a) - Q(s,o)\right]$$

where o is the option

k - denotes the number of time steps elapsing between s and s'.

r - Cumulative discounted reward over this time.

Intra-option value learning :-

→ SMDP Q-learning need to execute an option to termination before they can learan about it.

→ they can only be applied to one option at a time. the option that is executing at that time.

Intra-option methods

- off-policy learning methods that can learn about the consequences of one policy while actually behaving according to another. They can learn about values of executing certain options without ever executing those options.

Let $\tilde{Q}_o^*(s,o) = (1-\beta(s))\, Q_o^*(s,o) + \beta(s) \max_{o' \in O} Q_o^*(s,o')$

$$Q_o^*(s,o) = \sum_{a \in A_s} \pi(s,a)\, E\left[r + \gamma \tilde{Q}_o^*(s',o) \mid s,a\right]$$

$$= \sum_{a \in A_s} \pi(s,a) \left[r_s^a + \sum_{s'} P_{ss'}^a \tilde{Q}_o^*(s',o)\right]$$

one-step TD update:

$$Q(s_t,o) \leftarrow Q(s_t,o) + \alpha\left[\left(r_{t+1} + \gamma \tilde{Q}_o^*(s_{t+1},o)\right) - Q(s_t,o)\right]$$

— one step intra-option Q-learning applies this update rule to every option o consistent

with every action taken a_t.