Reinforcement Learning

Hierarchical Reinforcement Learning

Action hierarchy, hierarchical RL, semi-MDP

Vien Ngo
MLR, University of Stuttgart

Outline

- Hierarchical reinforcement learning
- Learning subgoals/hierarchy

Accelerating Reinforcement learning

- Temporal abstraction
- Goal/State Abstraction

Accelerating Reinforcement learning: Abstraction

- Temporal abstraction
- Goal/State Abstraction

Temporal Abstraction

- Dealing with multiple-time step ”macro” actions.
- Advantages:

  – Only exploring/computing values for interesting states (e.i. subgoals, . . .)

  – Transfer learning across problems/regions.

Hierarchical reinforcement learning

Three approaches to HRL

- Options: Sutton (temporal + state abstraction)
- Finite state controller: Parr & Russel (temporal abstraction)
- Given an action hierarchy: MAXQ (temporal + state abstraction)

Semi-Markov decision process

Semi-Markov decision process

SMDP = $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}\}$,

- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Transition function $\mathcal{T}(s, a, s', t) = p(s', t|s, a)$
- State space $\mathcal{R}(s, a)$

SMDP

Semi-Markov decision processes (SMDPs) generalize MDPs by

- allowing the decision maker to choose actions whenever the system state changes
- modeling the system evolution in continuous time
- allowing the time spent in a particular state to follow an arbitrary probability distribution

The system state may change several times between decision epochs; only the state at a decision epoch is relevant to the decision maker.

Semi-Markov Decision Process (SMDP)

- $\mathcal{T}(s, a, s', t) = P(s', t|s, a)$ defines the joint probability of a next state, and terminal time.

Bellman equations for SMDP

- Consider discrete-time SMDP:

$$V^*(s) = \max_a \left[R(s,a) + \sum_{s',\tau} \gamma^\tau p(s',\tau|s,a) V^*(s') \right]$$

$$Q^*(s,a) = R(s,a) + \sum_{s',\tau} \gamma^\tau p(s',\tau|s,a) \max_b Q^*(s',b)$$

Bellman equations for SMDP

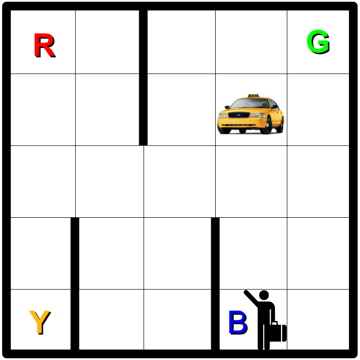
- Consider discrete-time SMDP:

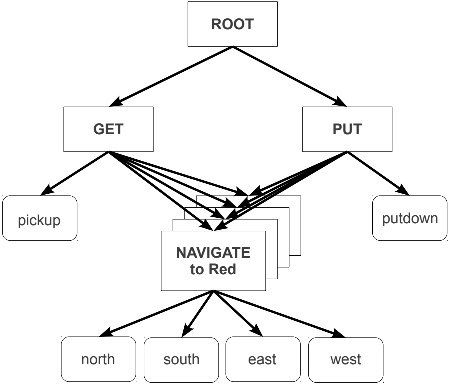
$$V^*(s) = \max_a \left[R(s,a) + \sum_{s',\tau} \gamma^\tau p(s',\tau|s,a) V^*(s') \right]$$

$$Q^*(s,a) = R(s,a) + \sum_{s',\tau} \gamma^\tau p(s',\tau|s,a) \max_b Q^*(s',b)$$

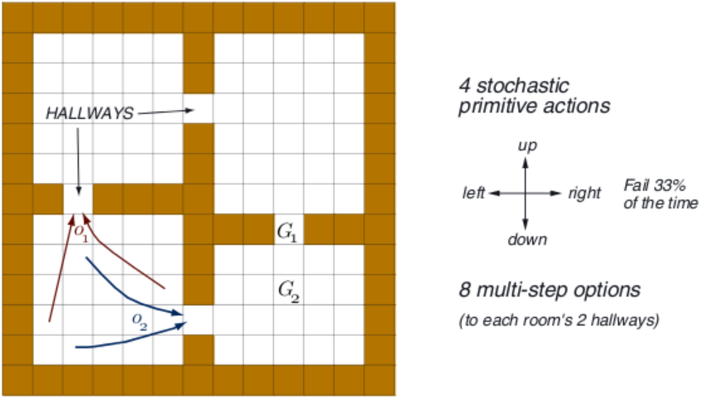
- Dynamic Programming algorithms are correspondingly extended to SMDPs (Howard, 1971; Puterman, 1994)

Example: Taxi Problem



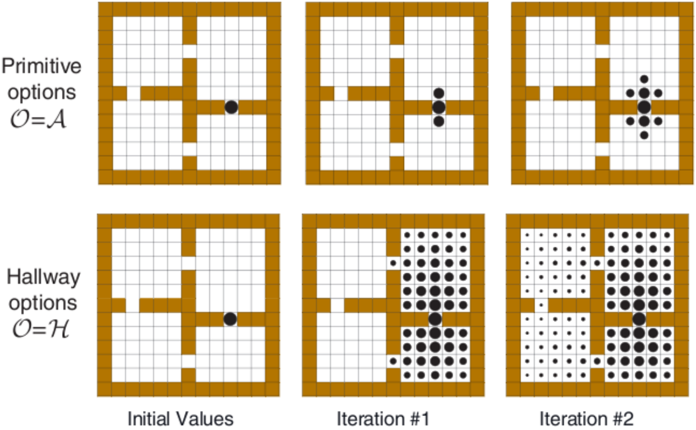


Example 2: SMDP



Sutton, Precup, Singh, 1999

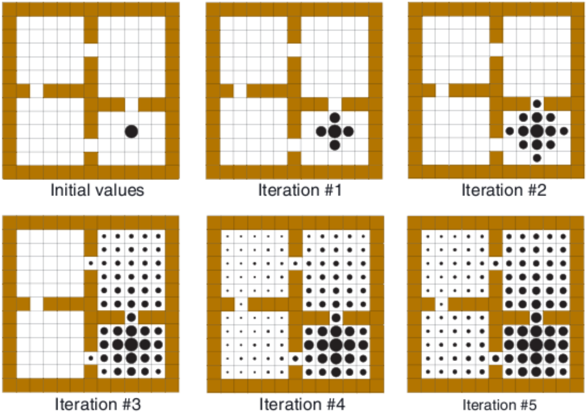
Example 2: SMDP



Sutton, Precup, Singh, 1999

Example 2: SMDP

Primitive and hallway options $\mathcal{O}=\mathcal{A}\cup\mathcal{H}$



Sutton, Precup, Singh, 1999

Options

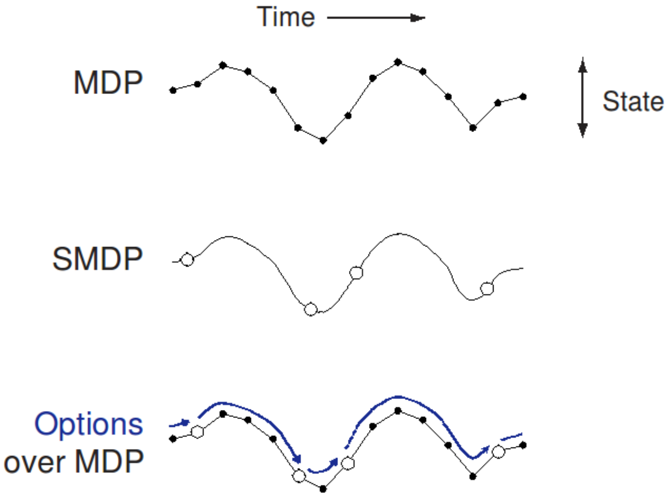
Sutton, Precup, Singh, 1999

Options

An option is a triple $o =< \mathcal{I}, \pi, \beta >$

- $\mathcal{I}$: initiation set.
- $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$: option's policy
- $\beta : \mathcal{S} \mapsto \Big[0, 1\Big]$: termination condition

Options



Value Functions for Options

option's policy: π_i; global policy: μ

- Denote
  - reward part of option:

$$r(s,o) = \mathsf{E}\Big\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^k r_{t+k} | o, s_t = s\Big\}$$

  - prediction-state part:

$$p(s'|s,o) = \sum_{k=1}^{\infty} p(s', k|s,o)\gamma^k$$

- Global policy's value function

$$\begin{aligned} V^\mu(s) &= \mathsf{E}\Big\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} ....|\mu, s_t = s\Big\} \\ &= \mathsf{E}\Big\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{k-1} r_{t+k} + \gamma^k V^\mu(s_{t+k})|\mu, s_t = s\Big\} \\ &= \mathsf{E}\Big[r(s,o) + \sum_{s_{t+k}} p(s_{t+k}|s,o) V^\mu(s')|\mu, s_t = s\Big] \end{aligned}$$

$$
\begin{aligned}
Q^{\mu}(s,o) &= \mathsf{E}\Big\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2}....|o\mu, s_t = s\Big\} \\
&= \mathsf{E}\Big\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{k-1} r_{t+k} + \gamma^k V^{\mu}(s_{t+k})|\mu, s_t = s\Big\} \\
&= \mathsf{E}\Big\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{k-1} r_{t+k} \\
&+ \max_{o'} \mu(s_{t+k}, o') Q^{\mu}(s_{t+k}, o')|o\mu, s_t = s\Big\} \\
&= \mathsf{E}\Big\{r(s,o) + \sum p(s_{t+k}|s,o) \max_{o'} \mu(s_{t+k}, o') Q^{\mu}(s_{t+k}, o')\Big\}
\end{aligned}
$$

Options: Learning

- SMDP Q-learning: given the set of defined options.
  – execute the current selected option (e.g use epsilon greedy $Q(s, o)$) to termination.
  – compute $r(s_t, o)$, then update $Q(s_t, o)$ as Q-learning/SARSA.

Options: Learning

- SMDP Q-learning: given the set of defined options.
  – execute the current selected option (e.g use epsilon greedy $Q(s, o)$) to termination.
  – compute $r(s_t, o)$, then update $Q(s_t, o)$ as Q-learning/SARSA.
- Intra-option Q-learning: partially defined options
  – after each primitive action, update all the options (off-policy learning).
  – converge to correct values, ”under same assumptions as 1-step Q-learning” (Sutton)

MAXQ

T. G. Dietterich (2000) ”Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition”, JAIR.

MAXQ

- The underlying MDP $\mathcal{M}$ is decomposed into a set of substask $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n$.
- $\mathcal{M}_0$ is the root subtask. (solving $\mathcal{M}_0$ solves $\mathcal{M}$).
- Each substask might consist of either primitive actions or other substasks.

example: TAXI problem.

MAXQ: Value Decomposition

- Consider all descendents a of a subtask $\mathcal{M}_i$ (or option $\mathcal{M}_i$)

$$
\begin{aligned}
V^\mu(i,s) &= \mathsf{E}\Big\{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2}....|\mu, s_t = s\Big\} \\
&\quad \text{(until } M_i \text{ terminates)} \\
&= \mathsf{E}\Big\{r_{t+1} + \gamma r_{t+2} + ... + \gamma^{k-1} r_{t+k} + \gamma^k V^\mu(i, s_{t+k})|\mu, s_t = s\Big\} \\
&= E\Big[r(s, \pi_i(s)) + + \gamma^{k-1} r_{t+k} + \gamma^k V^\mu(i, s_{t+k})|\mu, s_t = s\Big] \\
&= \underbrace{V^\mu(\pi_i(s), s)}_{\text{reward term}} + \underbrace{\sum_{s', N} p(s', N|s, \pi_i(s)) \gamma^N V^\mu(i, s')}_{C^\mu(i,s,\pi_i(s))(\text{completion term})}
\end{aligned}
$$

$$Q^\mu(i,s,a) = V^\mu(a,s) + C^\mu(i,s,a)$$

- The reward term:

$$
V^\mu(i,s) = \begin{cases} Q^\mu(i,s,\pi_i(s)) & \text{If } i \text{ is a macro action} \\ \sum_{s'} P(s'|s,a) r(s'|s,a) & \text{If } i \text{ is an primitive action} \end{cases}
$$

- The completion term $C^{\mu}(i, s, a)$ is the expected discounted cummulative reward of completing subtask $\mathcal{M}_i$ after taking subroutine $\mathcal{M}_a$ in state s.

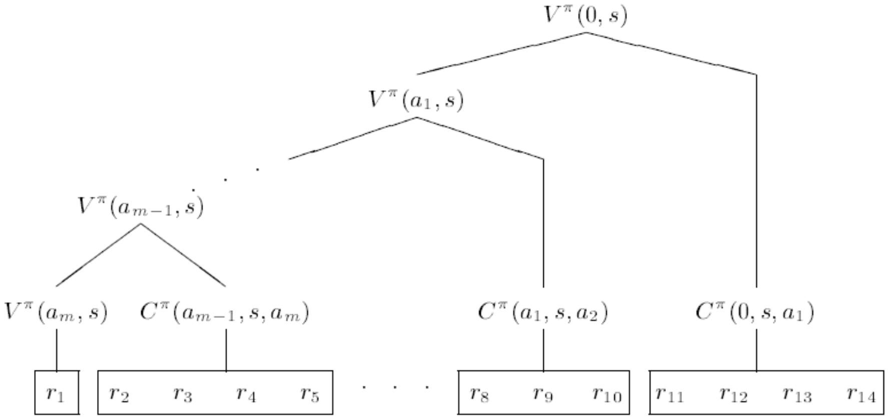
- The completion term $C^\mu(i, s, a)$ is the expected discounted cummulative reward of completing subtask $\mathcal{M}_i$ after taking subroutine $\mathcal{M}_a$ in state s.
- It recursively decompose $V^\mu(0, s)$ into value functions for $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n$.

- The completion term $C^\mu(i,s,a)$ is the expected discounted cummulative reward of completing subtask $\mathcal{M}_i$ after taking subroutine $\mathcal{M}_a$ in state s.
- It recursively decompose $V^\mu(0,s)$ into value functions for $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n$.
- In general:

$$V^\mu(0,s) = V^\mu(a_m, s) + C^\mu(a_{m-1}, s, a_m) + \ldots + C^\mu(a_1, s, a_2) + C^\mu(0, s, a_1)$$

  where $a_0, a_1, \ldots, a_m$ is a sequence of taken substasks by a hierarchical policy going from Root $\mathcal{M}_0$.
- For learning: only need to store C functions for non-primitive actions, and V for primitive actions.

Example of MAXQ value decomposition



$r_1, r_2, \ldots, r_{14}$ is a sequence of rewards w.r.t primitve actions at times $1, 2, \ldots, 14$.

MAXQ: Learning Algorithm

MAXQ-0 learning algorithm

- Given action hierarchy.
- Each subtask has *zero* pseudo terminal reward.

MAXQ-0 Learning Algorithm

Initialize $V(i,s)$ (for all primitive i) and $C(i,s,j)$ (for all non-primitive i, and descendents j of i) arbitrarily.

MAXQ-0(Node i, State s)

if i is primitve **then**
    execute i, receive r, s'
    $V_{t+1}(i,s) = (1-\alpha)V_t(i,s) + \alpha r_t$
    return 1
else
    $steps$ = 0
    while i not terminates **do**
        Choose $a \sim \pi_i(s)$ (e.g. $\arg\max_b Q(i,s,b)$))
        call $N =$ MAXQ-0(a,s) (recursive call)
        observe s'
        $C_{t+1}(i,s,a) = (1-\alpha)C_t(i,s,a) + \alpha.\gamma^N.V_t(i,s')$
        $steps = steps + N$
        $s = s'$
    return steps

MAXQ-0 Learning Algorithm

- Compute $V_t(i, s')$ if i is non-primitive?

MAXQ-0 Learning Algorithm

- Compute $V_t(i, s')$ if i is non-primitive?

$$V_t(i,s) = \begin{cases} \max_a Q_t(i,s,a) & \text{If } i \text{ is a macro action} \\ V_t(i,s) & \text{If } i \text{ is an primitive action} \end{cases}$$

$$Q_t(i,s,a) = V_t(a,s) + C_t(i,s,a)$$

MAXQ: Learning Algorithm

MAXQ-Q learning algorithm

- Given action hierarchy.
- When Each subtask has arbitrary *non-zero* pseudo reward $\tilde{R}_i$.
- MAXQ-Q introduces one more completion function for each subtask to use inside itself.

MAXQ-Q Learning Algorithm

Initialize $V(i,s)$ (for all primitive i) and $C(i,s,j)$ and $\tilde{C}(i,s,j)$ (for all non-primitive i, and descendents j of i) arbitrarily.

MAXQ-Q(Node i, State s)

```
if i is primitve then
    execute i, receive r, s'
    V_{t+1}(i,s) = (1 - α)V_t(i,s) + α r_t
else
    steps = 0
    while i not terminates do
        Choose a ~ π_i(s) (arg max_{a'} [C̃(i,s',a') + V(i,s')])
        call N = MAXQ-Q(a, s) (recursive call)
        observe s'
        a* = arg max_{a'} [C̃(i,s',a') + V(i,s')]
        C̃_{t+1}(i,s,a) = (1 - α)C̃_t(i,s,a) + α.γ^N.(R̃_i(s') + C̃_t(i,s',a*) + V_t(a*,s'))
        C_{t+1}(i,s,a) = (1 - α)C_t(i,s,a) + α.γ^N.(C_t(i,s',a*) + V_t(a*,s'))
        steps = steps + N
        s = s'
```

if i is primitve **then**
    execute i, receive r, s'
    $V_{t+1}(i,s) = (1-\alpha)V_t(i,s) + \alpha r_t$
else
    $steps$ = 0
    while i not terminates **do**
        Choose $a \sim \pi_i(s)$ $(\arg\max_{a'}[\tilde{C}(i,s',a') + V(i,s')])$
        call $N =$ MAXQ-Q(a,s) (recursive call)
        observe s'
        $a^* = \arg\max_{a'}[\tilde{C}(i,s',a') + V(i,s')]$
        $\tilde{C}_{t+1}(i,s,a) = (1-\alpha)\tilde{C}_t(i,s,a) + \alpha.\gamma^N.\Big(\tilde{R}_i(s') + \tilde{C}_t(i,s',a^*) + V_t(a^*,s')\Big)$
        $C_{t+1}(i,s,a) = (1-\alpha)C_t(i,s,a) + \alpha.\gamma^N.\Big(C_t(i,s',a^*) + V_t(a^*,s')\Big)$
        $steps = steps + N$
        $s = s'$

Optimality in HRL?

Optimality in HRL?

hierarchically optimal vs. recursively optimal

- Hierarchical optimality: The learnt policy is the best policy consistent with the given hierarchy. Task's policy depends not only on its children's policies, but also on its context.
- Recursive optimality: The policy for a parent task is optimal given the learnt policies of its children. (Context-free task's policy).

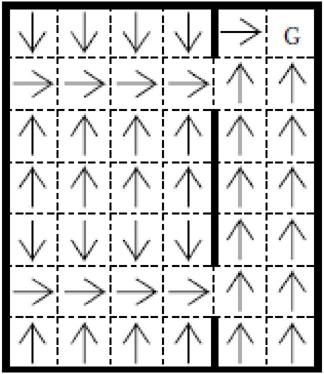
Optimality in HRL?

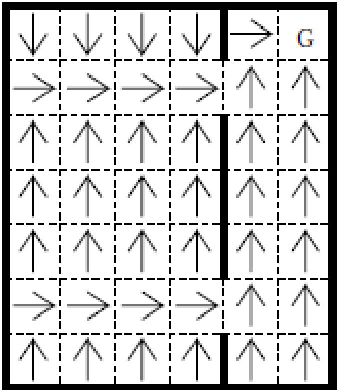
hierarchically optimal vs. recursively optimal

- Hierarchical optimality: The learnt policy is the best policy consistent with the given hierarchy. Task's policy depends not only on its children's policies, but also on its context.
- Recursive optimality: The policy for a parent task is optimal given the learnt policies of its children. (Context-free task's policy).

  – The context-free policies offer state abstraction/transfer learning better, which provides common macro actions to many other tasks.

Optimality in HRL

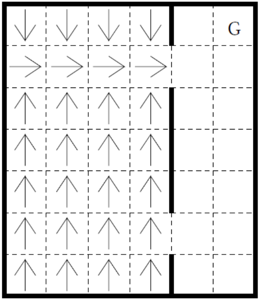




(an example from a tutorial of Dietterich).

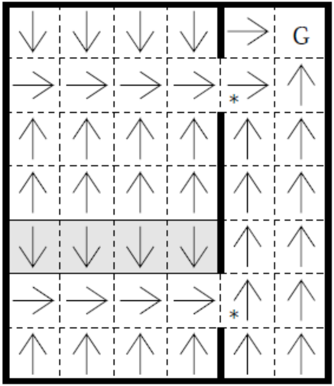
Optimality: in Options

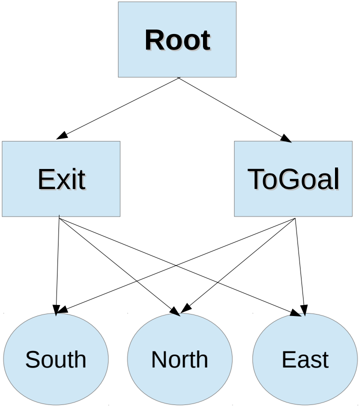
- If action space consists of both primitive actions and options, then it converges to an optimal policy.
- Otherwise, options with SMDP learning was proved to converge to a hierarchically optimal policy.



(an example from a tutorial of Dietterich).

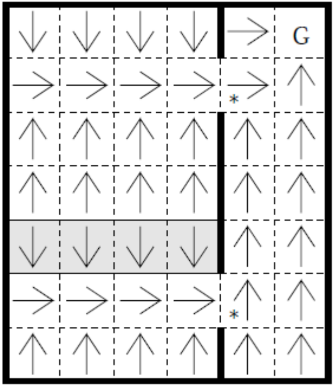
Optimality: in MAXQ

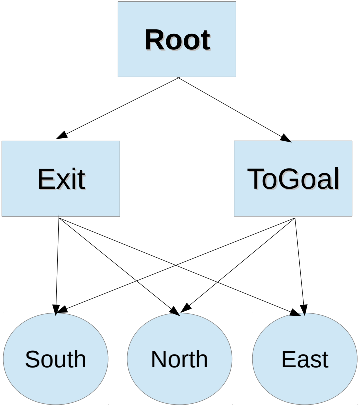




(an example from a tutorial of Dietterich).

Optimality: in MAXQ





(an example from a tutorial of Dietterich).

- MAXQ0 is recursively optimal, but MAXQ is hierarchically optimal

Optimality in HRL?

- Options learns a hierarchically optimal policy.
- MAXQ learns a recursively optimal policy.
  – MAXQ can obtain a policy which has hierarchical optimality with good design of subtask or with pseudo-reward learning.

Hierarchy/subgoal learning

Subgoal learning

- Creating usefull options randomly/heuristically, then adding gradually.

Subgoal learning

- Creating usefull options randomly/heuristically, then adding gradually.
- Creating an option/subgoal w.r.t a bottltneck (commonalities across multiple paths to a solution).

Hierarchy/subgoal learning

Amy McGovern, Andrew G. Barto, 2001.

Barto et. al. (2004, intrinsically motivated learning)

Hengst, 2002. (also use bottleneck)

Neville Mehta et. al. 2008 (using DBN)

etc.

Human hierarchical decision making

JJF Ribas-Fernandes, A Solway, C Diuk, JT McGuire, AG Barto, Y Niv & MM Botvinick (2011) - A neural signature of hierarchical reinforcement learning - Neuron 71:370-379