Suboptimality in Hierarchical RL

March 6th, 2013
CS394R Reinforcement Learning
Ruohan Zhang
Outline

• Source of suboptimality in
  – Recursively optimal policy
  – Hierarchically optimal policy

• Solutions have been developed
Recall: Recursively vs. hierarchically optimal policy

- **Hierarchical optimality**: the final policy is the best policy consistent with given hierarchy.
- **Recursive optimality**: the final policy is optimal given the policies learned by its children.
- Source of suboptimality for each type?
Domain (Dietterich)

- Grid world, start in the room on the left side, the Goal is located in the upper right corner.
- Actions: → ↓ ↑
- 2 doors
- Each action costs -1, goal gives reward 0.
Source of suboptimality

• What if we have the subtask as “exit by the nearest door?”
• What is the optimal policy, for the subtask?
Source of suboptimality

- What if we have the subtask as “exit by the nearest door?”
- What is the optimal policy, for the subtask?
Source of suboptimality

• From the optimal policies of our subtask, we achieve this final policy.
• Is it recursively optimal?
• Is it hierarchically optimal?
• Is it globally optimal?
Source of suboptimality

- From the optimal policies of our subtask, we achieve this final policy.
- Is it recursively optimal?
- Is it hierarchically optimal?
- Is it optimal?
- This is a *recursively* optimal policy, but not *hierarchically* optimal nor *globally* optimal.
Source of suboptimality

- What would be a hierarchically optimal policy?
- We can always exit by upper door.
- Is it recursively optimal?
- Is it globally optimal?
Source of suboptimality

• One question we may ask is, is hierarchically optimal policy always optimal? What about in our example?
Source of suboptimality

• One question we may ask is, is hierarchically optimal policy always optimal? What about in our example?

• If we put a “landmark” at the lower door, and we always exit by the lower door.

• The result is clearly hierarchically optimal, but not globally optimal.
Summary: source of suboptimality

- *Hierarchical optimality*: the imposed hierarchy constrains our policy.

- *Recursive optimality*: the policies learned from the subtasks are locally optimal, but we may have better policies for parent task.
Next...

- How do we deal with this problem?
  - Ideas?
  - There are helpful thoughts from our readings.
Solutions

• How do we deal with this problem?

• Approaches
  • Extend option set $O$ to include $A$ (primitive actions)
  • Redefine the reward of completing subtasks
  • Non-hierarchical execution
1. Extending $O$ to include $A$

- Introduce primitive actions as special cases of options
  - Recall the hallway example and experimental results

- What is the cost?
1. Extending $O$ to include $A$

- Introduce primitive actions as special cases of options
  - Recall the hallway example and experimental results

- What is the cost?
  - Could it be even slower than non-hierarchical learning?
2. Redefine the subtasks

• What is the difference between subtask and option?
  – *Option*: \(<I, \pi, \beta]\)
  – *Subtask*: \(<I, R, \beta]\)

• \(R\): *pseudo reward function.*
2. Dynamically redefine the subtasks

• Denote the subgoal states for task $i$ as $B(i)$

• Initialize $V'(s)$ for all states in $B(i)$

• Repeat:
  – Define a Pseudo-reward functions
    • $R'(s) = V'(s)$, for $s$ that are in $B(i)$
    • $0$, elsewhere
  – Apply hierarchical SMDP learning method to learn recursive optimal policy
  – Update $V'(s)$
3. Non-hierarchical execution

- $Q^\pi(s,a)$: function $Q$ for learned hierarchical policy $\pi$, $a$ is an option.
- At each time step, compute $a=\arg\max_a Q^\pi(s,a)$, then execute one primitive action according to $a$.
- We might terminate an option early.
- Similar to policy improvement in policy iteration, it always improves the policy.
3. Non-hierarchical execution

• An extreme case: what if we interrupt at every step (polling execution)? Do we still have advantage over non-hierarchical algorithms?
Thank you!