Approximate/Reinforcement Learning Approaches to POMDPs

Ron Parr
CompSci 590.11
Duke University

Overview

• Value function based methods
• Policy search
• Augmented state methods
Review

- Finite horizon POMDP value function is piecewise linear and convex (for arbitrary horizon lengths)
  - Max over a set of “alpha” vectors
  - Each vector corresponds to a conditional plan

- Number of pieces can grow exponentially

- Hard to solve problems with more than high 1’s or low 10’s of states

Point based methods

- Idea: Instead of generating ALL $\alpha$ vectors at each iteration, what if we generated just a subset?
- Every alpha vector would still be a valid conditional plan
- Value function would lower-bound the true value function

- Point based algorithms generate $\alpha$ vectors that are optimal for only a finite set of points, rather than for the entire belief space
Visualizing PBVI (figure from Pineau et al.)

Questions

• How do we pick the points?

• How do we find the optimal $\alpha$ vector for each point?
Picking Points

• Typically done heuristically

• Exploration from initial dist. finds a set of reachable belief states

• Reasonable if start dist. is known and/or entire belief space is not reachable (exact POMDP algorithms may be working too hard)

Compute new a vectors (figure from Ji et al.)

<table>
<thead>
<tr>
<th>Algorithm 1. Point-based backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>function $\Gamma' = \text{backup}(\Gamma, B)$</td>
</tr>
<tr>
<td>$% \Gamma$ is a set of $\alpha$-vectors representing value function</td>
</tr>
<tr>
<td>$\Gamma' = \emptyset$</td>
</tr>
<tr>
<td>for each $b \in B$</td>
</tr>
<tr>
<td>$\alpha^z_a = \arg \max_{\alpha \in \Gamma} \alpha \cdot b^z_a$, for every $a \in A$, $z \in Z$</td>
</tr>
<tr>
<td>$\alpha_a(s) = R(s, a) + \gamma \sum_{z, s'} p(s'</td>
</tr>
<tr>
<td>$\alpha' = \arg \max_{\alpha_a \in A} \alpha_a \cdot b$</td>
</tr>
<tr>
<td>if $\alpha' \not\in \Gamma'$, then $\Gamma' \leftarrow \Gamma' + \alpha'$, end</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>$%$ The following two lines are added for modified backup</td>
</tr>
<tr>
<td>$\Lambda = \text{all } \alpha \in \Gamma$ that are dominant at least at one $b^z_\pi(b)$</td>
</tr>
<tr>
<td>$\Gamma' = \Gamma' + \Lambda$</td>
</tr>
</tbody>
</table>
PBVI performance (figure from Pineau et al.)

**Figure 3**: PBVI performance for four problems: Tag(left), Maze33(center-left), Hallway(center-right) and Hallway2(right)

QMDP is an approximation method that uses 1 a vector per action at all iterations. Incremental Pruning was one best exact methods at the time.

PBVI limitations

- Fixed representation size was not adaptive to complexity value fn.

- Only as fast as value iteration, which converges asymptotically

- Not monotonic: can’t guarantee that values of all belief states increase at every iteration

- Is there a way to get the benefits of policy iteration?
Point Based Policy Iteration

- Combines policy iteration with point based methods
- Main idea:
  - Maintain a finite state machine (FSC) as the policy
  - Evaluate the FSC
  - Do one step of PBVI
  - Use output of PBVI to improve the FSC
  - Repeat

Limitations of Point Based Methods

- Still can be slow

- Assumes a known model:
  - At planning time
  - At execution time
Value function-based RL for POMDPs

• Since a POMDP is a continuous state MDP, why not use value function approximation on the continuous state?
• Many early efforts did this, e.g., RP’s second publication from grad school
• Problems:
  • Still requires a model to update the belief state
  • Problems with huge state spaces have huge belief states
• Solution(?): Use a compressed belief state, e.g., Bayes net, but this still requires a model, and an efficient way of updating the belief state

Policy Search for POMDPs
Policy Search

• Recall policy gradient (figure from Sutton & Barto):

\[
\theta_{t+1} = \theta_t + \alpha \gamma^t G_{\theta} \log \pi(A_t|S_t, \theta)
\]

This still works even if Markov property is violated, but...

Naïve policy search in POMDPs

• Policy search “works”, but policy is limited to a mapping from observations to actions

• Doesn’t directly address the partial observability issue

• At best can randomize actions to avoid losses from state confusion

• Not obvious how to combine with a critic for actor-critic methods because critic value function must be defined over states
Policy search with FSCs

• Create a random FSC
• Make transition probabilities and action probabilities tunable parameters
• Use policy gradient methods to tune both of these

• Cool idea that has been rediscovered many times over the years
• Hard to make work in practice:
  • Lots of local optima
  • Hard to find good controllers with more than a few hidden states

Policy Search in POMPs summary

• Advantages:
  • Does not require knowledge of the model
  • Does not need to maintain a belief state

• Disadvantages:
  • Belief states tell you a lot – hard to do well without them
  • Many of the challenges of policy gradient methods:
    • Local optima
    • Variance in the gradient estimate
    • Slow
  • Many fixes for policy gradient methods that are intended to reduce variance through use of a value function are not available
Augmented State Methods

Augmented state

• POMDPs are tricky b/c process is not Markovian in the observation
• Rather than change the algorithm, why not change the state?

• Advantage: Get to run regular MDP algorithms on the new state

• Challenge: How to do this
Finite History Window

- Problem might not be Markovian in current observation, but
- Perhaps it is Markovian if we augment the state to include a k-step window of previous states – see, e.g., DQN for Atari
- Advantages:
  - Obviously the right thing to do if you can afford to do it
  - Simple
- Disadvantages:
  - For n states, d step history, state space grows with $n^d$
  - Not always obvious how large to make d

History Trees

- Long history windows probably waste a lot of effort tracking irrelevant distinctions:
  - Many states may have unique/unambiguous observations
  - No need to remember history when we see these
- History trees define state as a variable length vector of previous states and actions sufficient to ensure Markov property
- In practice:
  - Collect statistics on histories
  - When violations of Markov property are detected, extend history
History tree example

- Robot going through maze
- Suppose two intersections look alike
- History tree can be used to remember how the robot \textit{got to} the intersection, to help distinguish between similar states

How to discover this:
- Need to collect statistics on all possible extensions of current histories
- When next states or next utilities diverge based upon different extensions of the history, grow the history

History tree Pros and Cons

- Works very well in some problems where short(ish) histories are sufficient to recover the Markov property
- More efficient than finite window methods

Limitations:
- Needs to collect a lot of data
- Can be hard to determine when to augment histories if there is a lot of noise
- Myopic/greedy (will miss if you need to remember something from 20 steps in the past, and remembering something 1...19 steps in the past doesn’t help.)
Augmented state with Function Approximation

- Idea: Use function approximation to learn how to augment the state “automatically” with a recurrent neural network (RNN)
- Old idea (at least as far back as Lin in the 90’s)

Learned, Augmented State

- Cool idea
- Creates self-reference in the function approximation
- Agent is essentially learning a belief state and method for updating the belief state at once
- Historically, such efforts were plagued by the difficulties associated with RNNs in general:
  - Convergence concerns
  - Difficulty with long term memory
- More recent efforts to address difficulties in recurrent neural networks, e.g., LSTMs, have shown promise in RL applications!
POMDP approximation summary

- Known model of moderate size: Use point based methods, or value function approximation on a compressed state

- Modest history dependence: Augment state, possibly using a learning method to discover required augmentation (e.g., history trees)

- Unknown model, unknown history dependence:
  - This is hard!
  - Learn model?
  - Deep learning with LSTM or similar methods to learn representation