MDP and Reinforcement Learning

Hao Cui
Hao.Cui@Tufts.edu

Three Questions

If you were in the AlphaGo team, and you wanted to build a Go (or any other board game) program that can beat the world champion, what to think about?

• What do we know?
• What do we not know?
• What do we want to achieve?

MDP – Modeling Decision Problems

Answer to Q1:
• Rules (what is the consequence of a move? How do we win/lose a game? Given the state, what are the legal actions?)
• Current position

Answer to Q3:
• Goal is to find the best move in current position, or to find a function that maps each position to a move

Answer to Q2???
We are going to talk about it.

Markov Decision Process and Reinforcement Learning

Reinforcement Learning (RL): Let your program learn to take actions!

Complete Information and problem size is feasible

Incomplete Information or problem size is too large

MDP Planning

RL

Evaluation and optimization

Trial and error

Know what to do

Policy Iteration (PI)

• Property of policy:

Define: \( \pi'(s) = \arg \max_a Q(s, a) \)

Theorem [Howard, 1960]: For any non-optimal policy \( \pi \) the policy \( \pi' \) a strict improvement over \( \pi \).

* This slide is from Alen Fern (OSU)

MDP – Modeling Decision Problems

* Problem Modeling
\( S \) is a finite state space
\( A \) is a finite action space
\( T(s, a, s') = p(s'|s, a) \) specifies transition of states
\( R(s, a) \) is the immediate reward of taking an action in state
\( \gamma \) is a discount factor

* Solutions
\( \pi : S \rightarrow A \) is a function maps states to actions
\( V^\pi(s) \) is defined as \( E[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) | \pi] \)
\( Q^\pi : S \times A \rightarrow \mathbb{R} \) same as \( V^\pi(s) \) but taking \( Q \) at first step
Policy Iteration (PI)

- Pick an arbitrary policy \( \pi \).
- Iterate:
  1. Policy evaluation: solve the linear system
     \[
     V(s) = \sum_{s' \in S} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V(s')] , \forall s \in S
     \]
  2. Policy improvement: for each \( s \in S \):
     \[
     \pi(s) = \arg \max_{a} \sum_{s' \in S} T(s, a, s') [R(s, a, s') + \gamma V(s')] 
     \]
     until \( \pi \) is unchanged.

* This slide is from Frazzoli (MIT)

Value Iteration (VI)

- Set \( V(s) \leftarrow 0 \), for all \( s \in S \)
- Iterate, for all \( s \in S \):
  \[
  V_{k+1}(s) = \max_{a} E \left[ R(s, a, s') + \gamma V_k(s') \right] 
  = \max_{a} \sum_{s' \in S} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right] , 
  \]
  until \( \max_s |V_{k+1}(s) - V_k(s)| < \epsilon \).

* This slide is from Frazzoli (MIT)

Motivation of RL

What if we don’t know some of the MDP’s components, for example, transition, reward? Or the state/action spaces are too large to explore?

Robots can learn how to act by “trial and error” even if we don’t know how!

Example of RL

- Recall
  \[
  Q^*(s) : S \times A \rightarrow \mathbb{R} \text{ same as } V^*(s) \text{ but taking } Q \text{ at first step} 
  \]
- The idea is simply to learn \( Q \) and optimize policy at the same time.

* Appearance of treasure and danger is stochastic

Hope and Questions for RL

Hope:
No matter if it is good or not, the robot can follow the current policy, get reward from the environment, and improve the policy.

Q1: How do you update the Q values?

Q2: What would happen if the robot always follows the current policy?
### Approaches to update Q values

\[ V^\pi(s) = \frac{1}{\sum \gamma^t} R_t \]

where \( R_t = \sum_{n=0}^{\infty} \gamma^n r(s_n, \pi(s_n)), s_0 = s \)

- **Monte-Carlo (MC)**
  \[ Q^\tau(s, a) \leftarrow Q^\tau(s, a) + \alpha [R - Q^\tau(s, a)] \]

- **Temporal Difference (TD):**
  \[ Q^\tau(s, a) \leftarrow Q^\tau(s, a) + \alpha [R + \gamma Q^\tau(s', \pi(a')) - Q^\tau(s, a)] \]

What is the difference?

### Exploration V.S. Exploitation

- Now think of a question. If you move to a new school and you don’t know how to get back home from the school. You would like to just give each path a try and you believe that you can find the shortest path returning home. What would you do?
  - If you already find a path and persist with it, you might end up with finding a suboptimal path
  - If you always try to make different turn when you arrive a intersection that you have already visited, you might end up with not even finding a path
  - You need make balance between exploration and exploitation

- Given current Q values, you can get the current policy. You need to follow the policy to some extent so that you can explore more about how good it is. You also need to make some randomness when you choose actions, by which you can improve the policy.

- **Epsilon-greedy**
  With epsilon probability choose action randomly, withn 1-epsilon choose it greedily

- **SoftMax**

*check out more about exploration and exploitation in Bandits if you have interest*

### On Line Optimization (SARSA)

**Repeat:**

- [in state s] take action a; observe r, s’
- choose next action a’ using policy P

\[ Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', \pi(a')) - Q(s, a)] \]

\[ P = \text{epsilon-greedy w.r.t. } Q \]

\[ s \rightarrow s', a \rightarrow a' \]

### On Line Optimization (Q learning)

**Repeat:**

- [in state s] take exploration policy action a; observe r, s’

\[ Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \]