CSIT5900
Lecture 4: Markov Decision Process and Reinforcement Learning

Fangzhen Lin

Department of Computer Science and Engineering
Hong Kong University of Science and Technology

Nondeterministic Actions

Games of chances: backgammon, blackjack, Russian roulette, and many more.

- **One shot decision**: You are given a choice of (1) take \$100; (2) flip a coin, if it's head then you get \$400 and if it's tail then you lose \$200. What would you do?
- **Money, time, and deadline**: You can take a taxi to the airport which takes 40 minutes and costs \$300.00 normally (.9 probability) but could be delayed up to 30 minutes (0.09 prob) or never get there (0.01 prob). You can also take public transportation which takes 80 minutes and costs \$50 normally (.99 prob) but could be delayed up to 30 minutes (0.009 prob) or never get there (0.001 prob). You need to get to the airport by 3pm and it's now 1pm. Which one would you take?
- **Any difference between uncertainty about your own action and undertainty about the environment**: Jump over a puddle of water vs take a step forward.

Planning under Uncertainty

- You are at floor 0 and your goal is to get to floor 6. You have available the following actions: Walk up one floor, which always succeeds, or roll a dice and it will magically take you up k floors with 0.5 probability, where k is the number on the six-faced dice that shows up after you rolled it, but with 0.5 probability you'll stay where you are.
- If you want to minimize time, what is your best plan?

Search Problems

Recall a search problem consists of:

- A set of states.
- A starting state.
- A set of actions that map states to states.
- Goal condition.
- Cost function: cost of doing an action.

Markov Decision Process (MDP): actions are indeterminate, the cost function depends on the starting and ending states of the action.

MDP

A MDP consists of:

- A set of states.
- A starting state.
- A set of actions modeled by a global probabilistic transition relation: $T(s, a, s')$ - the probability of reaching state s' from s by taking action a.
- Reward function: $Reward(s, a, s')$ - the reward for ending in s' when taking action a in s.
- Goal (terminating) condition: $End(s)$.
- A discount factor $0 \leq \gamma \leq 1$.

A Dice Game

A dice game from Percy Liang: at each round, you choose either:

- *quit* - you get \$10.
- *stay* - you get \$4 and a 6 sided dice is rolled and if it comes up 1 or 2, then end of the game otherwise continue the game to the next round.

A Dice Game - MDP Formulation

- States: *in* (start of the game) and *out* (end of the game).
- Starting state: *in*.
- $End(s)$ iff $s = out$.
- Actions: *quit* and *stay*.
- $T(in, quit, out) = 1$, $T(in, stay, out) = 2/6$, $T(in, stay, in) = 4/6$, $T(out, x, out) = 1$.
- $Reward(in, quit, out) = 10$, $Reward(in, stay, in) = 4$, $Reward(in, stay, out) = 4$.

Transitions

- The transition probabilities $T(s, a, s')$ specifies the probability of ending up in state s' when doing action a in s.
- If $T(s, a, s') \neq 0$, then s' is a possible successor.
- Since these are probabilities, they sum up to 1:

$$\sum_{s'} T(s, a, s') = 1.$$

Policies

What is a solution for MDP? A solution needs to tell which action to take in every possible situation.
A **policy** π is a mapping from states to actions. For example, here is a policy for playing the dice game is: $\pi(s) = \mathit{stay}$ if $s = \mathit{in}$, and nil otherwise. Here nil means no action, and is added so that policies are functions: $T(s, \mathit{nil}, s) = 1$ and $\mathit{Reward}(s, \mathit{nil}, s') = 0$.

Evaluating a policy

There are many policies. How do you compare them? The usual suspect: expected utilities.
Consider the above policy π for the dice game. If it yields the following run:

$$in, stay, 4, in, stay, 4, out$$

then its utility is the the sum of the discounted rewards along the sequence:

$$Reward(in, stay, in) + \gamma Reward(in, stay, out) = 4 + \gamma \times 4.$$

But the policy can yield many other runs. In fact it yield infinite number of runs.

Evaluating a policy

In discrete math, an effective way of dealing with possibly infinite sequence is to set up recurrences:

$$f(n) = \sum_{i=0}^{i=n} i \Rightarrow f(0) = 0, f(n) = n + f(n-1).$$

Given a policy π, let $V_\pi(s)$ be the expected utility of following π in state s:

$$V_\pi(s) = \begin{cases} 0, & \text{if } End(s) \\ \sum_{s'} T(s, \pi(s), s')[Reward(s, \pi(s), s') + \gamma V_\pi(s')], & \text{otherwise} \end{cases}$$

Evaluating a policy

It's more common to write the recurrences using a mutual recursion on what's called **Q-value**: Given a policy π

- Value: $V_\pi(s)$ - the expected utility of following π in state s.
- Q-value: $Q_\pi(s, a)$ - the expected utility of doing a in s and then following π.

$$V_\pi(s) = \begin{cases} 0, & \text{if } End(s) \\ Q_\pi(s, \pi(s)), & \text{otherwise} \end{cases}$$

$$Q_\pi(s, a) = \sum_{s'} T(s, a, s')[Reward(s, a, s') + \gamma V_\pi(s')].$$

Policy Evaluation - Example

Now consider the policy of stay: $\pi(in) = stay$, for the dice game. Assuming γ (discount factor) is 1.

$$
\begin{aligned}
V_\pi(out) &= 0, \\
V_\pi(in) &= Q_\pi(in, stay) \\
&= \sum_{s'} T(in, stay, s')[Reward(in, stay, s') + \gamma V_\pi(s')] \\
&= T(in, stay, in)[Reward(in, stay, in) + \gamma V_\pi(in)] + \\
&\quad T(in, stay, out)[Reward(in, stay, out) + \gamma V_\pi(out)] \\
&= 2/3(4 + V_\pi(in)) + 1/3(4 + 0) \\
&= 2/3(4 + V_\pi(in)) + 4/3
\end{aligned}
$$

Thus $V_\pi(in) = 12$.

Policy Evaluation - An Iterative Procedure

In the general case, there may not be a closed form solution. But we can use an iterative procedure to compute it

Policy Evaluation

Initialize $V_\pi^0(s) = 0$ for all s.

For each t=1,...,Max:

For each state s:

$$V_\pi^t(s) = \sum_{s'} T(s, \pi(s), s')[Reward(s, \pi(s), s') + \gamma V_\pi^{t-1}(s')].$$

How many iterations (Max) to try? Stop when V^t doesn't change much. Try it on the dice game:

$$V_\pi^t(in) = 2/3(4 + V_\pi^{t-1}(in)) + 4/3.$$

Optimal Policy and Value Function

- Policy evaluation computes the value of a state when executing the given policy.
- Given a MDP, we want to know the optimal policy and the optimal value function (maximal values of states)
- A policy π_{opt} is optimal if for every policy π', $V_{\pi_{opt}}(s) \geq V_{\pi'}(s)$ for all s.
- A value function V_{opt} is optimal if

$$V_{opt}(s) = max_{\pi} V_{\pi}(s) = V_{\pi_{opt}}(s).$$

- We first look at how to find an optimal policy by a hill climbing algorithm (policy iteration), then we look at how to compute the optimal value using a value iteration procedure.

Policy Iterations

Policy iteration:

Initialize π to a random policy.

For $i = 1, ..., MAX$:

- $\pi = successor(\pi)$

The key is of course $successor(\pi)$: improve a given policy π to a better one. This is called *policy improvement*.

Policy Improvement

Given a policy π, how to make it better?

- Better means that states have better values under the new policy.
- Recall that $V_\pi(s) = Q_\pi(s, \pi(s))$.
- Now if we can find a policy π' such that $V_{\pi'}(s) = max_{a \in Actions(s)} Q_\pi(s, a)$, then $V_{\pi'}(s) \geq V_\pi(s)$. Thus π' will be as good as π, and likely better.

Policy improvement: Given a policy π, improve it to π_{new}:

- Compute $V_\pi(s)$ for every s.
- Compute $Q_\pi(s, a)$ for every s and a in $Actions(s)$.
- Let π_{new} be: for each s

$$\pi_{new}(s) = arg\ max_{a \in Actions(s)} Q_\pi(s, a).$$

So let $successor(\pi)$ in policy iteration be π_{new}.

Policy Improvement: Example

Consider again the dice example. Let the current policy $\pi(in) = quit$ and let discount factor $\gamma = 1$.

$$
\begin{aligned}
& V_\pi(in) = 10, \;\; V_\pi(out) = 0, \\
& Q_\pi(in, quit) = \sum_s T(in, quit, s)(Reward(in, quit, s) + \gamma V_\pi(s)) \\
& \qquad = T(in, quit, out)(Reward(in, quit, out) + \gamma V_\pi(out)) \\
& \qquad = 10, \\
& Q_\pi(in, stay) = \sum_s T(in, stay, s)(Reward(in, stay, s) + \gamma V_\pi(s)) \\
& \qquad = 1/3(Reward(in, stay, out) + V_\pi(out)) + \\
& \qquad \quad 2/3(Reward(in, stay, in) + V_\pi(in)) \\
& \qquad = 4 + 2/3(10) > 10
\end{aligned}
$$

Thus $\pi_{new}(in) = stay$.

Value Iterations

If we have an optimal policy, we can compute optimal value $V_{opt}(s)$ in every state s. However, it would be good if we can compute optimal values directly:

$$V_{opt}(s) = 0, \text{ if } End(s),$$
$$V_{opt}(s) = max_{a \in Actions(s)} Q_{opt}(s, a),$$
$$Q_{opt}(s, a) = \sum_{s'} T(s, a, s')[Reward(s, a, s') + \gamma V_{opt}(s')]$$

With these recurrences, we can then have an iterative procedure to compute it.

Value Iterations

Value iteration (Bellman 1957):

Initialize $V_{opt}^0(s) = 0$ for all s.

For t = 1,..., MAX:

For each state s,

$$V_{opt}^t(s) = max_{a \in Actions(s)} \sum_{s'} T(s, a, s')[Reward(s, a, s') + \gamma V_{opt}^{t-1}(s')]$$

Convergence

Convergence Theorem: If either the discount factor $\gamma < 1$ or the MDP graph is acyclic, then both value iteration and policy iteration converge to the correct answer.

Reinforcement Learning

- Given a MDP, we can use either policy iteration or value iteration to compute optimal policy.
- What if some of the components are unknown, like don't know the transition distribution?
- Maybe you know the transition distribution, but it is too hard for you to write it down.

Reinforcement learning!

RL

In class discussion:

- Can an optimal controller be learned using RL for our boundary-following robot?
- Examples of RL?
- Blackjack game: `https://www.wikihow.com/Play-Blackjack` and `https://www.youtube.com/watch?v=eyoh-Ku9TCI`

RL - General Considerations

- Goal: Learn a policy to maximize the rewards.
- Given: a set of actions to do in a state.
- Assuming: know the states and have unlimited memory and computational resource;
- Question 1: how is a policy represented?
- Question 2: which action to try in the present state?
- Question 3: once an action is done and a reward received, how to update the current policy?

RL - General Framework

Initialization.

Loop:

1. Choose an action.
2. Observe the current state and record the rewards.
3. Update the parameters (transition model, expected values, expected Q-values, etc.).

Exploration vs Exploitation

Given the current knowledge about the model (how the world works), which action to take:

- Do the best action given what's known about the world now (*exploitation, go by the book*).
- Try an action that hasn't been tried before or did not work too well last time (*exploration, try something new*).

It's clear that unless we have a perfect model of the world, we need to do both.

Epsilon-Greedy

Epsilon-Greedy:

$$\pi(s) = \begin{cases} arg\ max_{a \in Actions} \hat{Q}(s, a) & \text{with } 1 - \epsilon \text{ probability} \\ \text{random } a \in Actions & \text{with } \epsilon \text{ probability} \end{cases}$$

Representation

How to update parameters (the model learned so far) depends on how they are represented:

- Tabular representation: transition relations, rewards, values, Q-values, policies are all represented by tables.
- Approximation: approximated by efficient functions such as neural networks - "deep reinforcement learning".

Tabular representation: Monte Carlo Method

Main idea of Monte Carlo Method: taking average of the sample. Samples:

$$D = [s_1, a_1, r_1, s_2, a_2, r_2, s_3, ...]$$

Estimating a model:

$$\begin{aligned} \hat{T}(s, a, s') &= \frac{\#(s, a, s') \in D}{\sum_{s''} \#(s, a, s'') \in D}, \\ \hat{Reward}(s, a, s') &= \text{average of } \{r : (s, a, r, s') \in D\} \end{aligned}$$

Example: $D = [S_1, A, 10, S_2, B, 8, S_1, A, 5, S_1, B, 2, S_1, A, 0, S_1]$.
$\hat{T}(S_1, A, S_1) = 2/3$ and $\hat{Reward}(S_1, A) = (10 + 5 + 0)/3$.
Given a model, optimal policy can then be computed using policy or value iterations.
Question: Why estimating model when only want to compute optimal policy? Why not estimate policy directly? Why not just estimate $Q(s, a)$?

Monte Carlo Method - Q values

$Q_\pi(s, a)$ - the expected utility in s when taking action a and then following π.
Samples:

$$D = [s_1, a_1, r_1, s_2, a_2, r_2, s_3, ...]$$

Utility at s_i:

$$u_i = r_i + \gamma r_{i+1} + \gamma^2 r_{i+2} + ...$$

Estimate:

$$\hat{Q}_\pi(s, a) = \text{average of } \{u_t \mid s_t = s, a_t = a\}$$

Example: $D = [S_1, A, 10, S_2, B, 8, S_1, A, 5, S_1, B, 2, S_1, A, 0, S_1]$. Let $\gamma = 1$.
$u_1 = 10 + 8 + 5 + 2 + 0 = 25, u_2 = 15, u_3 = 7, u_4 = 2, u_5 = 0$.
$\hat{Q}(S_1, A) = (u_1 + u_3 + u_5)/3$.

Tabular Representation: Temporal Differences

The Monte Carlo methods require an entire episode (epoch), which is a sequence of state-action-rewards transitions to a terminal state, in order to update the model/Q-values.
Temporal differences take into account current knowledge about the model/Q-value, and update them in every step. A prominent method here is Watkin's Q-learning.

Q-Learning

Assume we have the current estimate of Q-value, $\hat{Q}(s, a)$ for all state s and action a. Now given an observed transition (S, A, r, S'), update the estimated Q-value of (S, A) as follows:

$$\hat{Q}(S, A) = (1 - \mu)\hat{Q}(S, A) + \mu(r + \gamma \hat{V}(S')),$$

where

- μ is the learning rate;
- γ the discount factor;
- $\hat{V}(S') = max_{a \in Actions(S')} \hat{Q}(S', a)$;
- $\hat{Q}(S, A)$ in the right side is the old estimate, and $r + \gamma \hat{V}(S')$ is the new learned value;
- the update formula strives for a balance between the old value and the learned value, controlled by the learning rate μ.

Deep Q-Learning (DQN)

When the number of states are too large, one needs an efficient way of representing the Q-value (Q-function). The first work on using a deep neural network to represent a Q-function is done in Deepmind, in the context of playing Atari games like breakout:

Playing Atari with Deep Reinforcement Learning
By Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller
DeepMind Technologies
2013 `https://arxiv.org/pdf/1312.5602.pdf`

Policy Gradients

- So far we represent and learn a policy through Q-function, which is a value function.
- Are there any other way to represent policies, especially ones that are easy to learn (i.e. have good learning algorithm)?
- The obvious idea in the age of DL: deep neural network.
- Reference:
  - Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3), 229–256.
  - Sergey Levine (Berkeley) slides: `http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_4_policy_gradient.pdf`;
  - Jan Peters (Max-Planck Institute, Germany) Scholarpedia article: `http://www.scholarpedia.org/article/Policy_gradient_methods#Likelihood_Ratio_Methods_and_REINFORCE`.