Advanced Topics in Reinforcement Learning

Lecture 4: Dynamic Programming

Josiah Hanna
University of Wisconsin — Madison

Announcements

- Thanks for completing the background survey!
- Sign-up for a presentation:https://docs.google.com/spreadsheets/d/1PMI8XO9lP84GW5jYFJi1qPo6E19ZKacw5nRKxY7YTu8/edit?gid=0#gid=0
  - Many of the latter slots have been taken
- Final reminder: join Piazza to stay in-the-know on key class information.

Learning Outcomes

After today's class, you will be able to:

1. Be able to define and explain the relationship between state-values, action-values, and optimal policies.

2. Be able to translate Bellman equations into dynamic programming methods for computing value functions.

Value functions

- State transitions and rewards are stochastic so we must maximize **expected return.**

- Expected return is only well-defined with respect to a particular policy. (Why?)

- State-value and action-value functions are always defined in terms of some policy.

$$v_\pi(s) = \mathbb{E}_\pi[G_t \,|\, S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|\, S_t = s]$$

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \,|\, S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|\, S_t = s, A_t = a]$$

Recursive Relationship of State Values

$$v_\pi(s) := \mathbb{E}_\pi[G_t \,|\, S_t = s]$$

Definition of return

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s]$$

Definition of expectation

$$= \sum_a \pi(a \,|\, s) \sum_{s'} \sum_r p(s', r \,|\, s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1} \,|\, S_{t+1} = s']]$$

Definition of state-value

$$= \sum_a \pi(a \,|\, s) \sum_{s'} \sum_r p(s', r \,|\, s, a)[r + \gamma v_\pi(s')]$$

Page 59 of the course textbook.

Action Values

Write action-values in terms of environment dynamics and state-values:

$$q_\pi(s,a) := \mathbb{E}_\pi[G_t \,|\, S_t = s, A_t = a]$$

Definition of return

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s, A_t = a]$$

Definition of expectation

$$= \sum_{s'} \sum_{r} p(s', r \,|\, s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1} \,|\, S_{t+1} = s']]$$

Definition of state-value

$$= \sum_{s'} \sum_{r} p(s', r \,|\, s, a)[r + \gamma v_\pi(s')]$$

Exercise 3.13, page 58.

Practice: Action Values

Write state-values in terms of action-values:

Hint:

$$q_\pi(s, a) = \sum_{s'} \sum_{r} p(s', r \,|\, s, a)[r + \gamma v_\pi(s')]$$

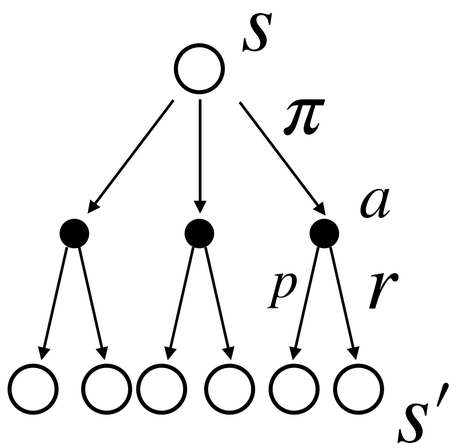
$$v_\pi(s) = ?$$

Exercise 3.12, page 58.

Bellman Equation

- The book uses the concept of a **back-up** diagram to illustrate value function computations:

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma v_\pi(s')]$$



Optimality

- Agent's objective: find policy that maximizes $v_\pi(s)$ for all s.
- The optimal policy — policy that has maximal value in all states. $\pi^\star \geq \pi$ if $v_{\pi^\star} \geq v_\pi(s)$ for all states and possible policies.
  - Does this policy always exist?
  - Is it unique?
- Possibly multiple, but always at least one optimal policies in a finite MDP.
  - Also, deterministic and Markovian, i.e., action selection only depends on current state.
- $\pi^\star(s) = \arg\max_a q_{\pi^\star}(s, a)$

$$q_{\pi^\star}(s, a) = \mathbb{E}[R_{t+1} + \gamma v_{\pi^\star}(S_{t+1}) \,|\, S_t = s, A_t = a]$$

Approximation

- The optimal policy exists but, in practice, it may not be possible to compute.
- In real world problems, we must settle for approximate optimality.
- This is an opportunity — no need to waste time finding optimal actions in states the agent rarely visits.
- Need to generalize knowledge across states — more on this in October!

Michael’s Presentation

- Link to slides.

Optimal Value Functions

- Like all policies, the optimal policy has value functions:
  - $v_{\pi^\star}(s) = \mathbb{E}[R_{t+1} + \gamma v_{\pi^\star}(S_{t+1}) \,|\, S_t = s]$
  - $q_{\pi^\star}(s, a) = \mathbb{E}[R_{t+1} + \gamma v_{\pi^\star}(S_{t+1}) \,|\, S_t = s, A_t = a]$
- The optimal policy is greedy with respect to the action-values, i.e., $\pi^\star(s) = \arg\max_a q_{\pi^\star}(s, a)$

Bellman Optimality

$$
\begin{aligned}
v_*(s) &= \mathbf{E}_{\pi^\star}[q_\star(s, A)] && \text{From last time: state-value is expected action-value.} \\
&= \sum_a \pi^\star(a \mid s) q_\star(s, a) && \text{Definition of expectation.} \\
&= \max_a q_\star(s, a) && \text{Optimal policy is greedy w.r.t } q_\star \\
&= \max_a \mathbf{E}_{\pi^\star}[G_t \mid S_t = s, A_t = a] && \text{Definition of action-value .} \\
&= \max_a \mathbf{E}_{\pi^\star}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] && \text{Recursive definition of return.} \\
&= \max_a \mathbf{E}_{\pi^\star}[R_{t+1} + \gamma v_\star(S_{t+1}) \mid S_t = s, A_t = a] && \text{Definition of state-value.} \\
&= \max_a \sum_{s', r} p(s', r \mid s, a)[r + \gamma v_\star(s')] && \text{Definition of expectation.}
\end{aligned}
$$

Dynamic Programming in RL

- Dynamic programming is a general class of algorithm that builds a solution to a problem by recursively solving sub-problems.
- In RL, dynamic programming refers to algorithms that compute values at one state using values (partially) computed for other states.
  - Not learning methods!
- “Bootstrapping”
  - Learning a guess from a guess.
  - Methods that use initial value estimates to compute new, improved value estimates.
  - From the expression “pull oneself up by your own bootstraps.”
  - Not to be confused with bootstrapping in statistics.

Dynamic Programming in RL

- Use value functions to find improved policies.
- Turn Bellman equations into value function updates.
- Bellman equation for policy value becomes policy evaluation:

$$v_{k+1}(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma v_k(s')]$$

- Bellman optimality equation becomes value iteration:

$$v_{k+1}(s) \leftarrow \max_a \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma v_k(s')]$$

Limitations of Dynamic Programming

- Require full knowledge of the environment
  - Know transitions and rewards.
- May have high computational requirements; linear in actions, states, and rewards per-update.
- We will discuss relaxing these limitations when we discuss model-based learning in a few weeks.
- What is done in practice?
  - Dynamic programming methods are applied for solving MDPs in practice.
  - Not for full RL problems; but key ideas are important!

Policy Evaluation (Prediction)

- Given a policy, compute its state- or action-value function.

$$v_{k+1}(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma v_k(s')]$$

$$q_{k+1}(s, a) \leftarrow \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma \sum_{a'} q_k(s', a')]$$

- When to stop making updates?
- Do these updates converge?
  - Yes, updates are a **contraction mapping** with respective fixed points v_π, q_π.
  - Convergence proof for value-iteration. Can you generalize it?

Policy Evaluation Demo

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Policy Iteration

- We have $v_\pi(s)$ for the current policy π. How can we improve π?
- Alternate:
  - Run policy evaluation updates to find v_π.
  - Set $\pi'(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]$
  - Why does this work?

Policy Improvement Theorem

- Suppose for π that $\exists s, a$ such that $q_\pi(s, a) \geq v_\pi(s)$.
- Let $\pi'(s) = a$ and $\pi'(\tilde{s}) = \pi(\tilde{s})$ for all other states $\tilde{s}$.
- What is true about π'? Why?
  - As good as or better than π, i.e., $v_{\pi'}(s) \geq v_\pi(s), \forall s$
- If π is sub-optimal, does there exist s, a such that $q_\pi(s, a) \geq v_\pi(s)$?
  - Yes, this follows from Bellman Optimality. Must be at least one state where π is not greedy w.r.t. its action-value function.
  - Optimal value function: $v_\star(s) = \max_a q_\star(s, a) \forall s$

Policy Iteration Demo

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Summary

- Learning value functions allow us to compute optimal policies.
- Policy Evaluation: find value function for a fixed policy.
- Policy Iteration: compute optimal policy by iterating 1) policy evaluation and 2) greedy policy improvement.
- Value Iteration: directly learn optimal value function.
- Dynamic programming methods don’t solve the full RL problem but they are the basis for most of the methods we will see in this class.

Action Items

- Read Chapter 5 of course textbook.
- Send a reading response by 12pm on Monday.
- Sign-up for a presentation: https://docs.google.com/spreadsheets/d/1PMI8XO9lP84GW5jYFJi1qPo6E19ZKacw5nRKxY7YTu8/edit?usp=sharing