Advanced Topics in Reinforcement Learning

Lecture 4: Dynamic Programming

Josiah Hanna
University of Wisconsin — Madison

Announcements

- Homework 1 released on canvas; due Thursday, September 29.
- Reading Sign-Ups: https://docs.google.com/spreadsheets/d/1-dce7-qzt8EVM4gYOLIl5WzYEGpioWM4x0VyA6QimzY/edit#gid=0
- **How important is the math?**
  - Very! Particularly Bellman equations for policy value and optimality.

Overview

- <u>Course Overview</u>
- Review Bellman Equations (wrap up Bellman optimality).
- Yuxiao's Presentation
- Policy Evaluation via Dynamic Programming
- Policy Iteration

Bellman Equation (Review)

- Bellman equation expresses state-value, $v_\pi(s)$, in terms of expected reward and state-values at next time-step.

$$v_\pi(s) = \mathbf{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+2}) \mid S_t = s]$$

$$v_\pi(s) = \mathbf{E}_\pi[R_{t+1} \mid S_t = s] + \gamma \mathbf{E}_\pi[v_\pi(S_{t+2}) \mid S_t = s]$$

Expected immediate reward

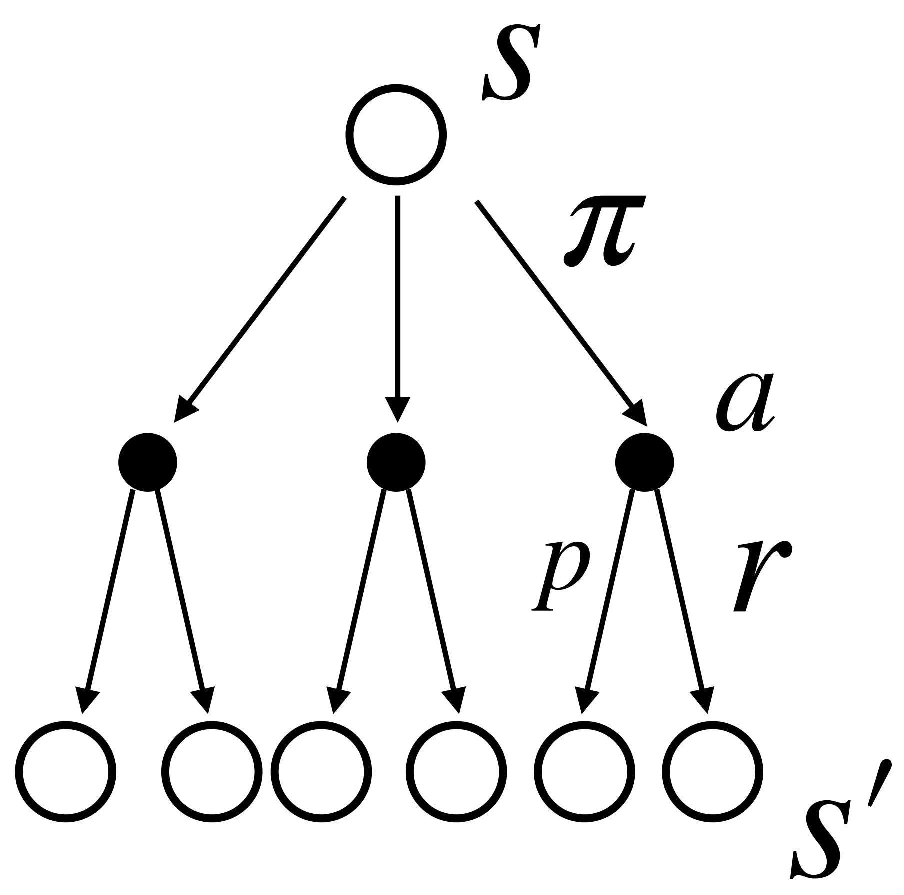
Expected future reward for t' > t+1

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

Bellman Equation

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

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



Golf Example

- State is ball location. Actions are putt (short distance, accurate) or drive ball (long distance, less accurate).
- Reward is -1 until the ball goes in the hole.
- What is value of policy that always putts?

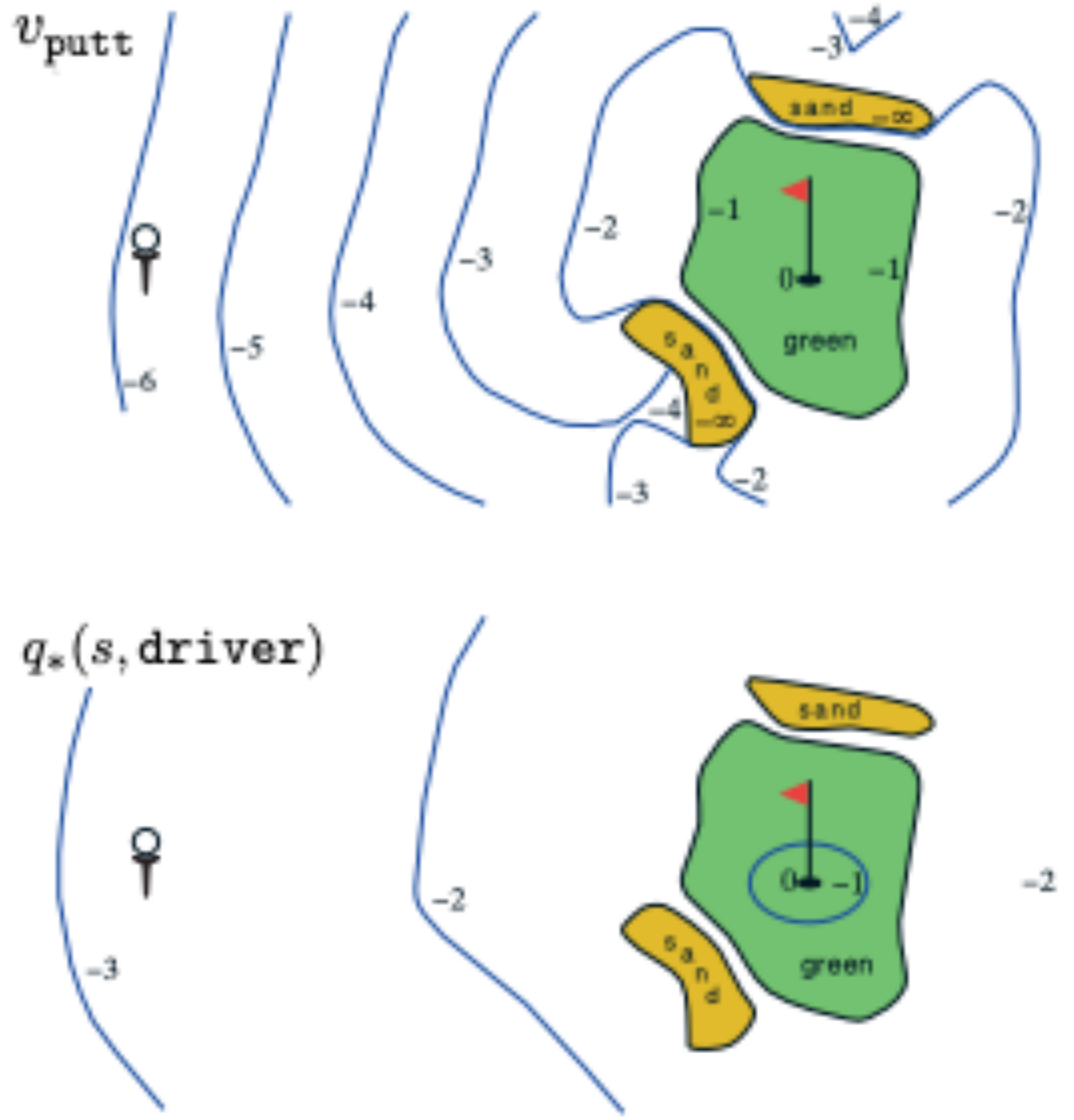


Figure 3.3: A golf example: the state-value function for putting (upper) and the optimal action-value function for using the driver (lower). ■

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.
- $\pi^\star(s) = \arg\max_a q_{\pi^\star}(s, a)$

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

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

$$v_*(s) = \mathbf{E}_{\pi^\star}[q(s, A)]$$
From last time: state-value is expected action-value.

$$= \sum_a \pi^\star(a \mid s) q_\star(s, a)$$
Definition of expectation.

$$= \max_a q_\star(s, a)$$
Optimal policy is greedy w.r.t $q_\star$

$$= \max_a \mathbf{E}_{\pi^\star}[G_t \mid S_t = s, A_t = a]$$
Definition of action-value .

$$= \max_a \mathbf{E}_{\pi^\star}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a]$$
Recursive definition of return.

$$= \max_a \mathbf{E}_{\pi^\star}[R_{t+1} + \gamma v_\star(S_{t+1}) \mid S_t = s, A_t = a]$$
Definition of state-value.

$$= \max_a \sum_{s', r} p(s', r \mid s, a)[r + \gamma v_\star(s')]$$
Definition of expectation.

Golf Example

- State is ball location. Actions are putt (short distance, accurate) or drive ball (long distance, less accurate).
- Reward is -1 until the ball goes in the hole.
- What is action-value of using driver and then following the optimal policy?

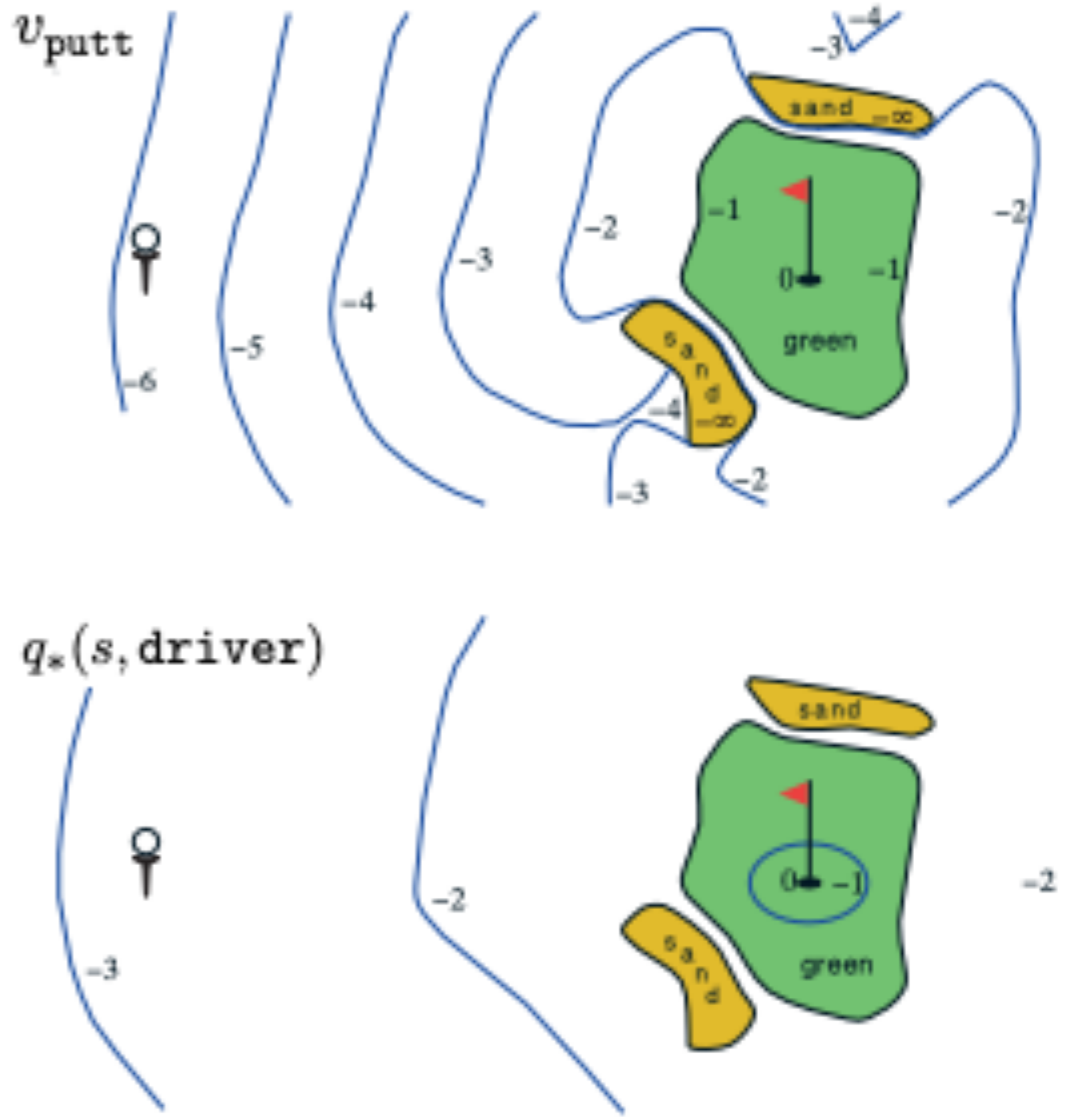


Figure 3.3: A golf example: the state-value function for putting (upper) and the optimal action-value function for using the driver (lower). ■

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!

Yuxiao’s Presentation

- Link to slides.

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, update is a **contraction mapping** with fixed point 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 Improvement (Control)

- 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.
- 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

- Bellman equations express relationships between values at one state and subsequent states.
- Dynamic programming turns Bellman equations into value function updates.
- Policy Evaluation: find value function for a fixed policy.
- Policy Iteration: compute optimal policy by iterating 1) policy evaluation and 2) greedy policy improvement.

Action Items

- Homework 1 now released. Due September 29 @ 9:29 am.
- Start reading for next week.
- Be thinking about final project — proposal due in 2.5 weeks.
    - Application of RL to a domain of your choice.
    - Or an algorithmic modification to improve an RL algorithm.
    - The more concrete your proposal is, the better guidance you will receive!