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.

• Start reading chapter 5 for next week.

• Project page: https://pages.cs.wisc.edu/~jphanna/teaching/2022fall_cs839/project.html
Overview

- The Policy Improvement Theorem
- Policy Iteration
- Value Iteration
- Asynchronous Value Iteration
- Samarth’s Presentation
Policy Evaluation (Prediction)

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

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

• We have $v_\pi(s)$ for the current policy $\pi$. How can we improve $\pi$?
  
  • Behave greedily w.r.t. \[ \sum_{s', r} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \]

• Suggests a simple algorithm. Alternate:

  • Run policy evaluation updates to find $v_\pi'$.
  
  • 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 $\pi$ that $\exists s, a$ such that $q_\pi(s, a) \geq v_\pi(s)$.

• Let $\pi'(s) = a$ and $\pi'(^s) = \pi(^s)$ for all other states.

• What is true about $\pi''$? Why?
  
  • As good as or better than $\pi$, i.e., $v_\pi(s) \geq v_{\pi'(s)}(s)$, $\forall s$

  • If $q_\pi(s, a) \geq v_\pi(s)$ then always taking action $a$ cannot decrease expected return.

• If $\pi$ is sub-optimal, does there exist $s, a$ such that $q_\pi(s, a) > v_\pi(s)$?

  • Yes, this follows from Bellman Optimality. Must be at least one state where $\pi$ 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

• First, evaluate $\pi$ to obtain $v_\pi$.

• Then, update $\pi$ to $\pi'$ such that $\pi'(s) = \arg\max_a \sum_{s',r} p(s', r | s, a) [r + v_\pi(s')]$

• Policy improvement theorem guarantees that $v_{\pi'}(s) \geq v_\pi(s) \forall s$.

• Can converge quickly in practice (in terms of policy updates).

• (Fixing the subtle bug on page 80).
Policy Iteration Demo

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

• What’s wrong with policy iteration?
  • Policy evaluation must converge between policy updates.
  • We don’t need the exact action-values — just which action has maximal action-value.

• Value iteration combines policy evaluation and iteration in one step:
  \[ v_{k+1}(s) \leftarrow \max_a \sum_{s', r} p(s', r \mid s, a)[r + \gamma v_k(s')] \]

• In-place or out-of-place updates?
  • In-place propagates value updates faster.
  • Out-of-place is more amenable to analysis.
Value Iteration

\[
V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r r(s', r) \left[ r + \gamma V(s') \right]
\]

Value Iteration

Policy Evaluation
Value Iteration Demo

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

- Regular DP methods require sweeps over the entire state space *per-iteration*!

- Infeasible if we have a large state-space.

- Actually unnecessary — can update states in any order and still converge as long as all states updated infinitely often in the limit.

- Why does this help?

\[
\begin{align*}
v_{k+1}(s) & \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a) \left[ r + \gamma v_k(s') \right]
\end{align*}
\]
Generalized Policy Iteration

- What is it?
  - We can be quite permissive in how we mix evaluation and improvement.
  - As long as $\nu$ becomes closer to $\nu_\pi$ and $\pi$ becomes greedy w.r.t. $\nu$ we will converge to $\nu_\star, \pi_\star$.

- A general framework for all algorithms we will introduce in this class.

- Do you think this holds when $\nu_\pi$ must generalize across states? I.e., increasing the value of $\nu_\pi(s)$ will also increase the value of $\nu_\pi(s')$ for $s'$ close to $s$. 

Josiah Hanna, University of Wisconsin — Madison
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

- Homework 1 released. Due September 29 @ 9:29 am.

- Start reading chapter 5 for next week. First learning methods!

- Be thinking about final project — proposal due in 2 weeks.
  - The more concrete your proposal is, the better guidance you will receive!