Reinforcement Learning: Tutorial 3

Dynamic programming

Week 2
University of Amsterdam

Alejandro García Castellanos
February 2026

Check-in

- How is it going?
- How is HW1?

Outline

1. Admin
2. Dynamic programming exercises
3. Ask anything about HW1

Admin

- Proposal: from now on, we do one hour exercises and one hour self-work on a HW with the possibility to ask questions
- Is everything going well with Codegra.de?
- Any questions?

Tutorial 3 Overview

1. Dynamic programming exercises
2. Ask anything about HW1

Tutorial 3 Overview

1. Dynamic programming exercises
   - Questions 2.1-2.2
2. Ask anything about HW1

Theory Intermezzo: Policy iteration, Value iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(\mathit{terminal}) \doteq 0$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

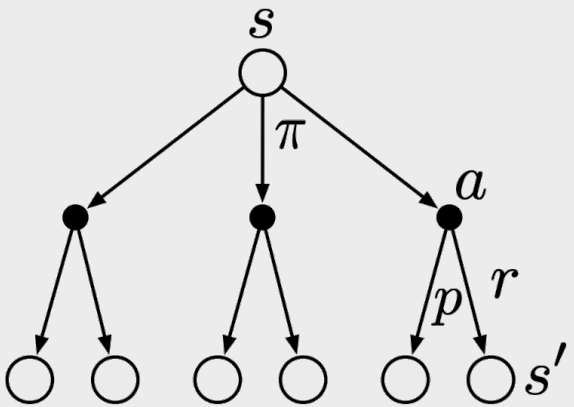
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

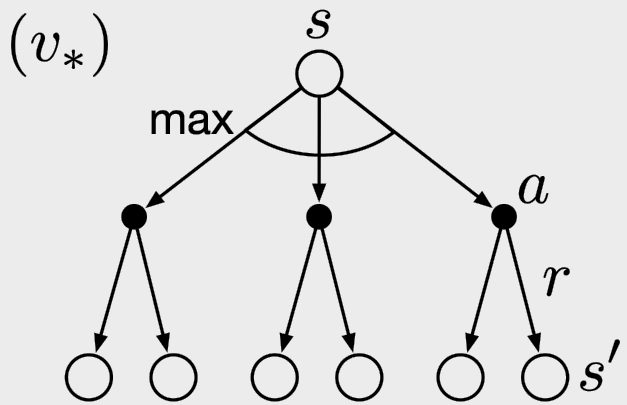
Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\mathit{terminal}) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

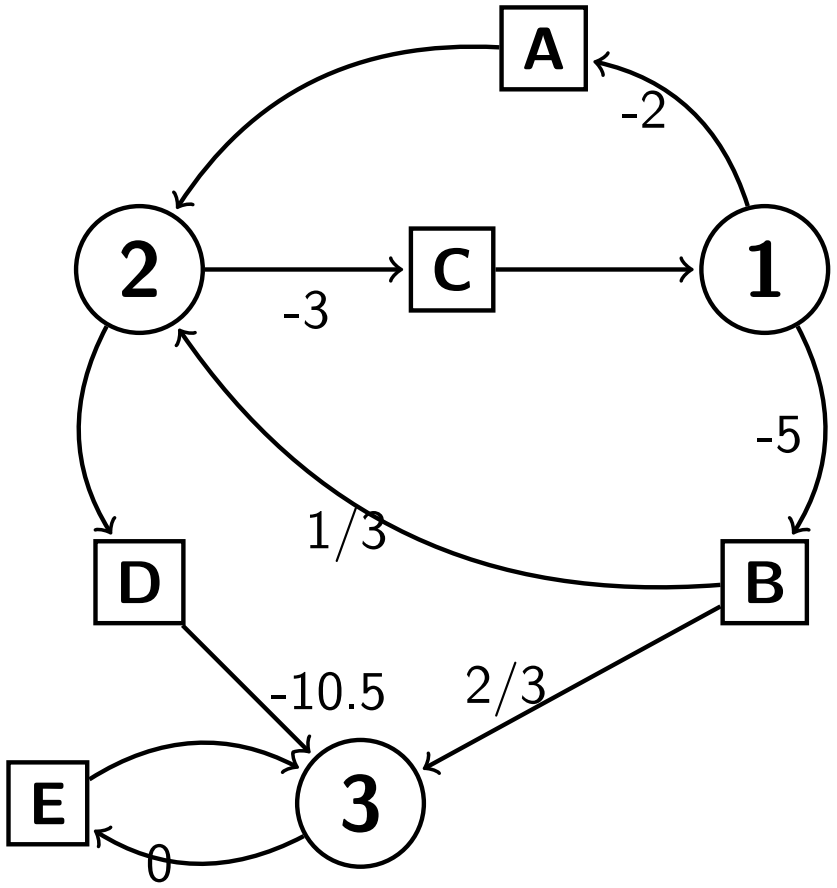
Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$





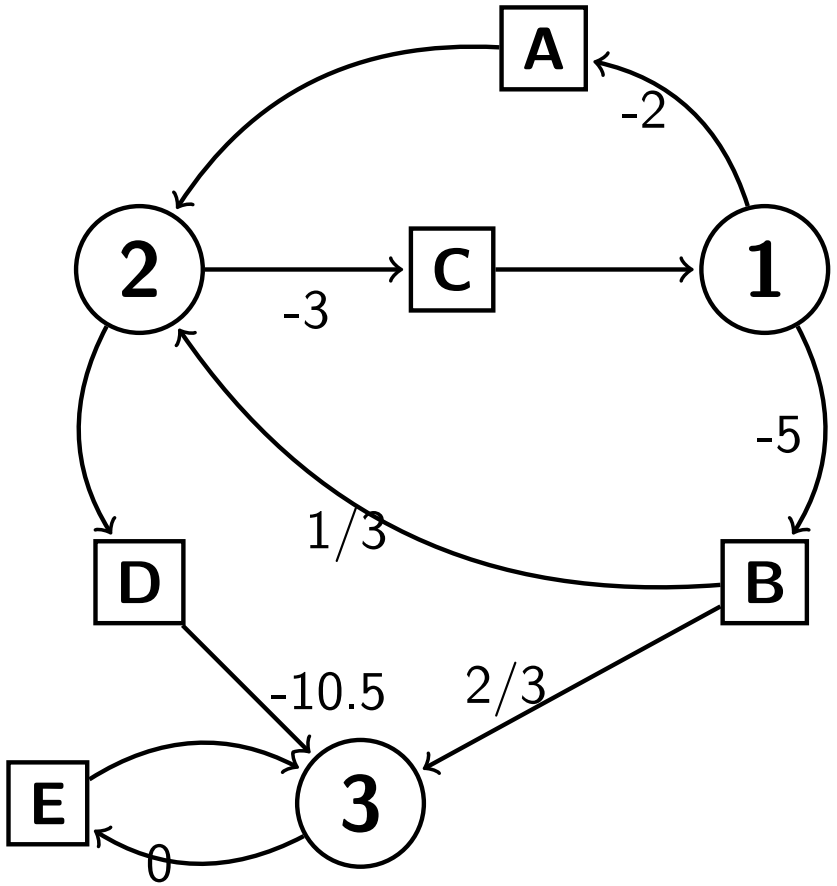
Q 2.1 Dynamic programming

MDP consisting of 3 states: 1, 2, 3. The actions (A,B,C,D,E) available at each state are deterministic apart from action B. Arrows represent actions with labels being the rewards. State 3 is a terminal state.



Q 2.1 Dynamic programming

Perform Value iteration for all states until convergence. Use a discount factor of 1 for this question. Initialise values at 0 for all states. Keep track of the maximizing actions at each iteration.



Q 2.1 Dynamic programming

$$v_0(s) = [0, 0, 0]$$

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + v_k(s')]$$

$$v_1(s_1) = \max\{-2, -5\} = -2$$

$$v_1(s_2) = \max\{-3, -10.5\} = -3$$

$$v_1(s_3) = 0$$

$$v_1(s) = [-2, -3, 0] \quad \text{(maximizing actions: A, C, E)}$$

$$v_2(s_1) = \max\{-2 + -3, -5 + -3/3\} = -5$$

$$v_2(s_2) = \max\{-3 + -2, -10.5\} = -5$$

$$v_2(s_3) = 0$$

$$v_2(s) = [-5, -5, 0] \quad \text{(maximizing actions: A, C, E)}$$

$$\vdots$$

Q 2.1 Dynamic programming

Continuing yields the results (adding the maximizing action in parentheses):

state	v_0^*	v_1^*	v_2^*	v_3^*	v_4^*	v_5^*	v_6^*	v_7^*
s_1	0	-2(A)	-5(A)	-6 $\frac{2}{3}$(B)	-7 $\frac{2}{3}$(B)	-8 $\frac{2}{9}$(B)	-8.5(B)	-8.5(B)
s_2	0	-3(C)	-5(C)	-8(C)	-9 $\frac{2}{3}$ (C)	-10.5(D)	-10.5(D)	-10.5(D)
s_3	0	0(E)	0 (E)	0(E)	0(E)	0(E)	0(E)	0(E)

Q 2.1 Dynamic programming

1. What are the state values V^* at each of the states?

Q 2.1 Dynamic programming

1. What are the state values V^* at each of the states?

   The values of each state are -8.5, -10.5, 0, respectively for states 1, 2, and 3.
   State values are the expected cumulative returns with an infinite horizon, which is what value iteration converges, too.

Q 2.1 Dynamic programming

2. When considering some finite horizon k (i.e., when caring only about the rewards in the next k time steps), how can the corresponding value V_k^* be read off from the results of value iteration?

Q 2.1 Dynamic programming

2. When considering some finite horizon k (i.e., when caring only about the rewards in the next k time steps), how can the corresponding value V_k^* be read off from the results of value iteration?

   Finite-horizon value functions v_k^* measure the expected cumulative return with a horizon of k. These can be read off directly from the intermediate results of value iteration, provided initialize at the rewards we get in terminating in each particular state, in this case: $v_0^* = 0$.
   The initialization does not matter for the final result of value iteration but it does for the intermediate steps.

Q 2.1 Dynamic programming

3. For different horizons k, different policies are optimal. What are the optimal policies for this example?

Q 2.1 Dynamic programming

3. For different horizons k, different policies are optimal. What are the optimal policies for this example?

   By looking at the maximizing actions at each step, we can look at policies that are optimal for some horizon: (A,C,E), (B,C,E), and (B,D,E). So 3 in total.
   Note that this means that in the finite horizon case we might execute different actions in the same state depending on the amount of time that is left!

Q 2.2 Exam question: Dynamic programming

1. Judge the following statements with True/False and add a short explanation.

Q 2.2 Exam question: Dynamic programming

1. Judge the following statements with True/False and add a short explanation.
   a. Value Iteration tends to converge to better policies than Policy Iteration.

Q 2.2 Exam question: Dynamic programming

1. Judge the following statements with True/False and add a short explanation.
   a. Value Iteration tends to converge to better policies than Policy Iteration.

   False, they both converge to optimal policies and therefore give the same return.

Q 2.2 Exam question: Dynamic programming

1. Judge the following statements with True/False and add a short explanation.
   b Value Iteration combines one sweep (one backup per state) of Policy Evaluation and one sweep of Policy Improvement in each sweep.

Q 2.2 Exam question: Dynamic programming

1. Judge the following statements with True/False and add a short explanation.
   b Value Iteration combines one sweep (one backup per state) of Policy Evaluation and one sweep of Policy Improvement in each sweep.

   True, in each iteration we find the greedy policy (maximizing action), which is policy improvement and do one value update under that policy (policy evaluation).

Q 2.2 Exam question: Dynamic programming

2. We consider the optimal value function in state s, $v_*(s) = \max_\pi v_\pi(s)$, where v_π is the value function for policy π. The Bellman optimality equation at this state is given by:

$$v_*(s) = \max_a \sum_{s',r} p\left(s', r \mid a, s\right)\left[r + \gamma v_*\left(s'\right)\right],$$

where $p(s', r|a, s)$ represents the transition probabilities to state s' with reward r, given that action a was taken in state s, and γ is a discount factor. Explain why this equation holds when policy iteration stabilizes (you may assume a tabular setting).

Q 2.2 Exam question: Dynamic programming

2. The Bellman optimality equation at this state is given by:

$$v_*(s) = \max_a \sum_{s',r} p\left(s', r \mid a, s\right)\left[r + \gamma v_*\left(s'\right)\right].$$

(a) The policy improvement step doesn't change the policy anymore, so the policy is given by

$$\pi(s) = \arg\max_a \sum_{s',r} p\left(s', r \mid a, s\right)\left[r + \gamma v_\pi\left(s'\right)\right]$$

(b) The policy evaluation step doesn't change the value function anymore, so

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

Since by (a) the policy returns the argmax, we have that

$$v_\pi(s) = \max_a \sum_{s',r} p\left(s', r \mid a, s\right)\left[r + \gamma v_\pi\left(s'\right)\right].$$

The Bellman optimality equation holds so the value function is v_*.


WELL DONE!

Tutorial 3 Overview

1. Dynamic programming exercises
2. Ask anything about HW1
   - Questions 2.3-2.4

Ask anything about HW1

1. 2.3: Coding
2. 2.4: Theory

That's it!

See you tomorrow