“All have their worth,” said Yavanna,

“and each contributes to the worth of the others”.

J.R.R. Tolkien, *The Silmarillion*

CMPUT 365
Introduction to RL

Marlos C. Machado

Classes 10 & 11/35

Plan

- Dynamic programming
  - Finally, a solution method (albeit limited)!

Reminder (1 of 2)

You **should be enrolled in the private session** we created in Coursera for CMPUT 365.

I **cannot** use marks from the public repository for your course marks.

You **need** to **check**, **every time**, if you are in the private session and if you are submitting quizzes and assignments to the private section.

The deadlines in the public session **do not align** with the deadlines in Coursera.

If you have any questions or concerns, **talk with the TAs** or email us `cmput365@ualberta.ca`.

Reminder (2 of 2)

The midterm is next week!

- It will be written here, in the classroom (50 minutes).
- I need you to bring some piece of ID.
- It'll have multiple-choice (see Coursera) and open-ended questions (see Worksheets)
  - We won't have a practice midterm, but it doesn't matter, if we did it, it would be a mix and match of Coursera and Worksheet questions. You don't need me for that.
- It is closed-book, no calculators are allowed.
- **There won't be a formula sheet.**
- Previous years:

  2023: Avg.: 68%, Max: 100% 2024: Avg.: 65%, Max: 100% 2025: Avg.: 60%, Max: 100%

Please, interrupt me at any time!

https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Dynamic Programming – Why?

- “DP provides an essential foundation for the understanding of the methods presented in the rest of this book”.
- … but “classical DP algorithms are of limited utility in reinforcement learning both because of their assumption of a perfect model and because of their great computational expense”.
- “all of these [RL] methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment”.

Models and Planning

- How should we think about p(s', r | s, a)?

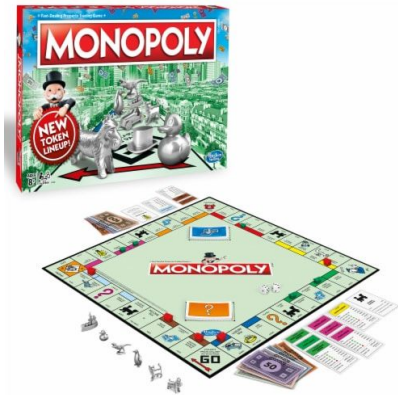
It is a model. It tells us everything that is possible and impossible to happen (and their probability!)

- Is dynamic programming different from what we did in bandits?

Figuring out how to act

Imagine the universe consists of you playing Monopoly against a computer. Your goal is to win the game.

There are two ways you can do so:



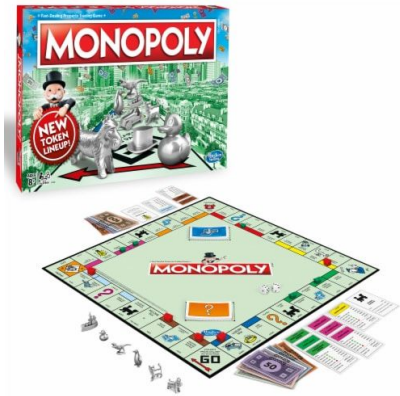
https://www.kroger.com/product/images/large/front/0019516614631

Figuring out how to act

Imagine the universe consists of you playing Monopoly against a computer. Your goal is to win the game.

There are two ways you can do so:

1. **Trial and error learning.** Play against it over and over, figure out the game rules and the computer's strategy.



https://www.kroger.com/product/images/large/front/0019516614631

Figuring out how to act

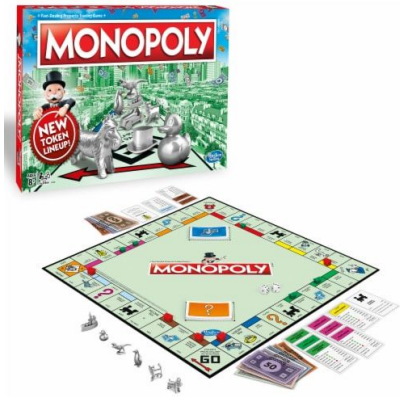
Imagine the universe consists of you playing Monopoly against a computer. Your goal is to win the game.

There are two ways you can do so:

1. **Trial and error learning.** Play against it over and over, figure out the game rules and the computer's strategy.
2. **Planning.** You could be given access to the game's rulebook as well as the code implementing the AI playing against you. You would then and there and **think** about how to win. You **reason** about the rules and the AI, and **plan** how to win.

There's no interaction!

The game's rulebook and the code implementing the AI would allow you to compute p(s', r | s, a).



https://www.kroger.com/product/images/large/front/0019516614634

Key Idea Behind Dynamic Programming

"To use value functions to organize and structure the search for good policies."

We use the same equations as before, but we replace an = by a ←, that's it (we turn Bellman equations into assignments).

There's lots to decide

- What should we compute? v_π, q_π, v_*, q_*, π^*?
- How should we select states to imagine about? And in what order?
- How much computation do we need to figure out the optimal policy, π^*, using the function p: $\mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$?
- How many times do we need to iterate over this imagining / planning process?

Obtaining value functions and π^* from π and p (with no interaction) is called ***Dynamic Programming****.*

Policy Evaluation (Prediction)

Given a policy and an MDP, what's the corresponding value function?

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) \Big[r + \gamma v_\pi(s') \Big]
\end{aligned}
$$

$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) \Big[r + \gamma v_k(s') \Big]
\end{aligned}
$$

expected update

Policy Evaluation (Prediction)

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

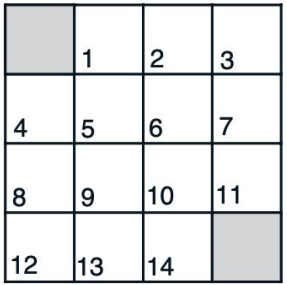
Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$ arbitrarily, for $s \in \mathcal{S}$, and $V(\textit{terminal})$ to 0

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

“in-place” update

Policy Evaluation – Example

actions



$R_t = -1$
on all transitions

v_k for the random policy

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

→

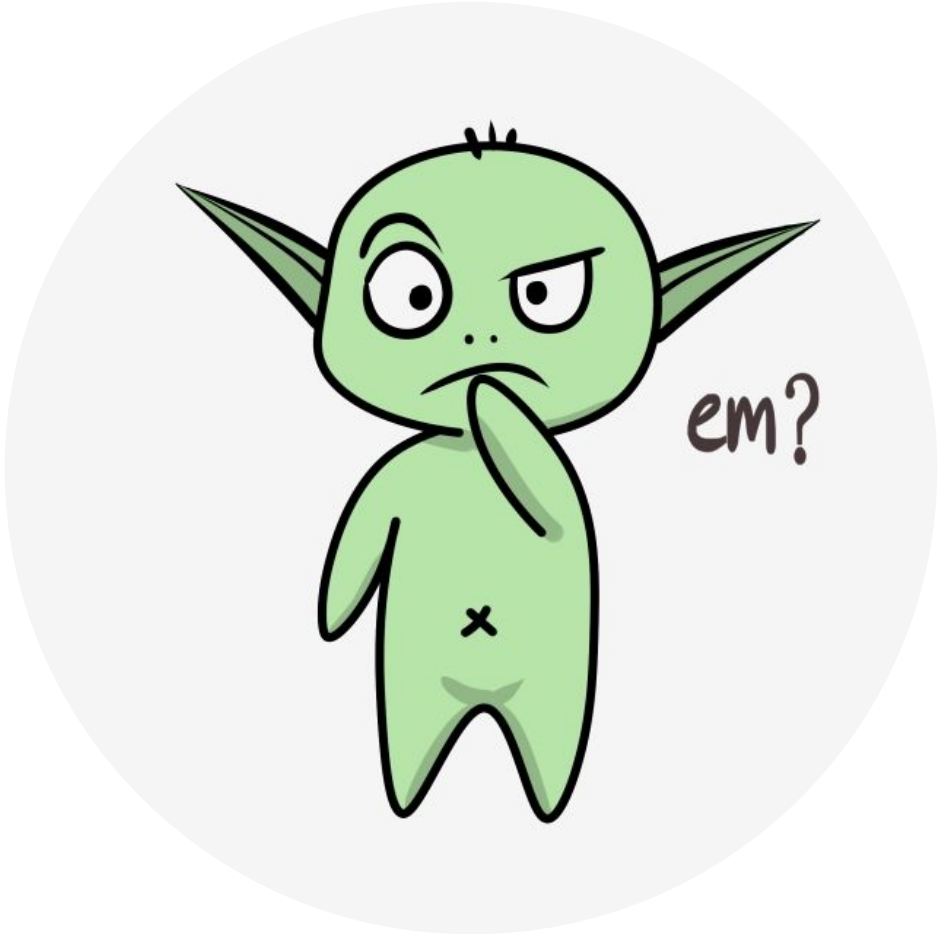
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

→

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

→ … →

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Policy Improvement

Given a value function for a policy π, how can we get a better policy π'?

We already know how good policy π is, what if we acted differently now? What if instead of selecting action π(s) we selected action a ≠ π(s), but then we followed π?

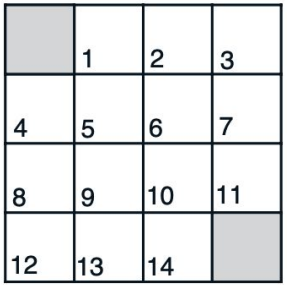
We know the value of doing that!

$$
\begin{aligned}
q_\pi(s,a) &\doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_\pi(s') \Big].
\end{aligned}
$$

If this new action is better, in general this new policy is better overall

Policy Improvement – Intuition

actions

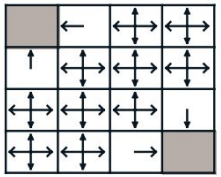


$R_t = -1$
on all transitions

v_k for the random policy

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



Policy Improvement Theorem

That this is true is a special case of a general result called the *policy improvement theorem*. Let π and π' be any pair of deterministic policies such that, for all $s \in \mathcal{S}$,

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{4.7}$$

Then the policy π' must be as good as, or better than, π. That is, it must obtain greater or equal expected return from all states $s \in \mathcal{S}$:

$$v_{\pi'}(s) \geq v_\pi(s). \tag{4.8}$$

A more aggressive update

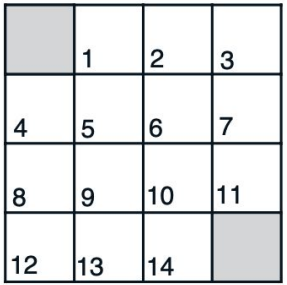
Instead of doing it for a particular action in a single state, we can consider changes at *all* states and to *all* possible actions.

$$\begin{aligned}
\pi'(s) &\doteq \arg\max_a q_\pi(s,a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \arg\max_a \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_\pi(s') \Big],
\end{aligned}$$

This is called *policy improvement*. And eventually it converges to the optimal policy.

Policy Improvement – Intuition

actions

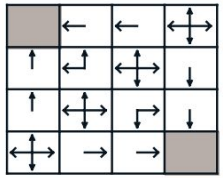


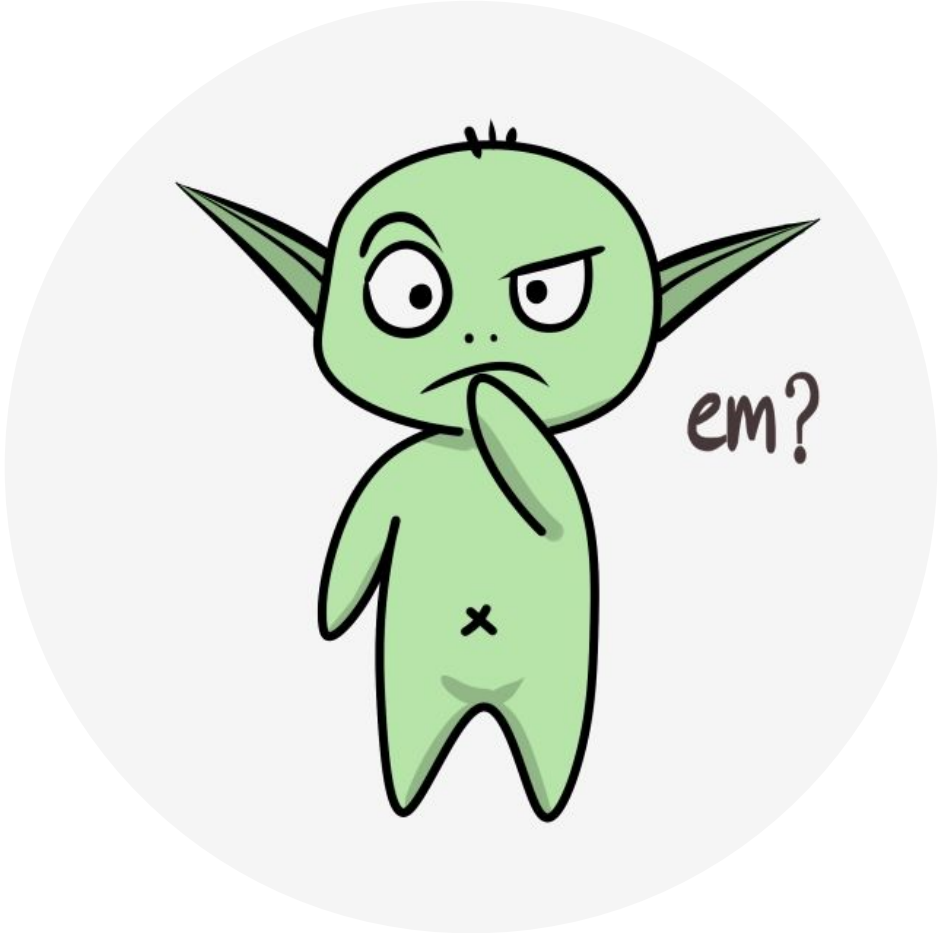
$R_t = -1$
on all transitions

$k = 2$

v_k for the random policy

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0





https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

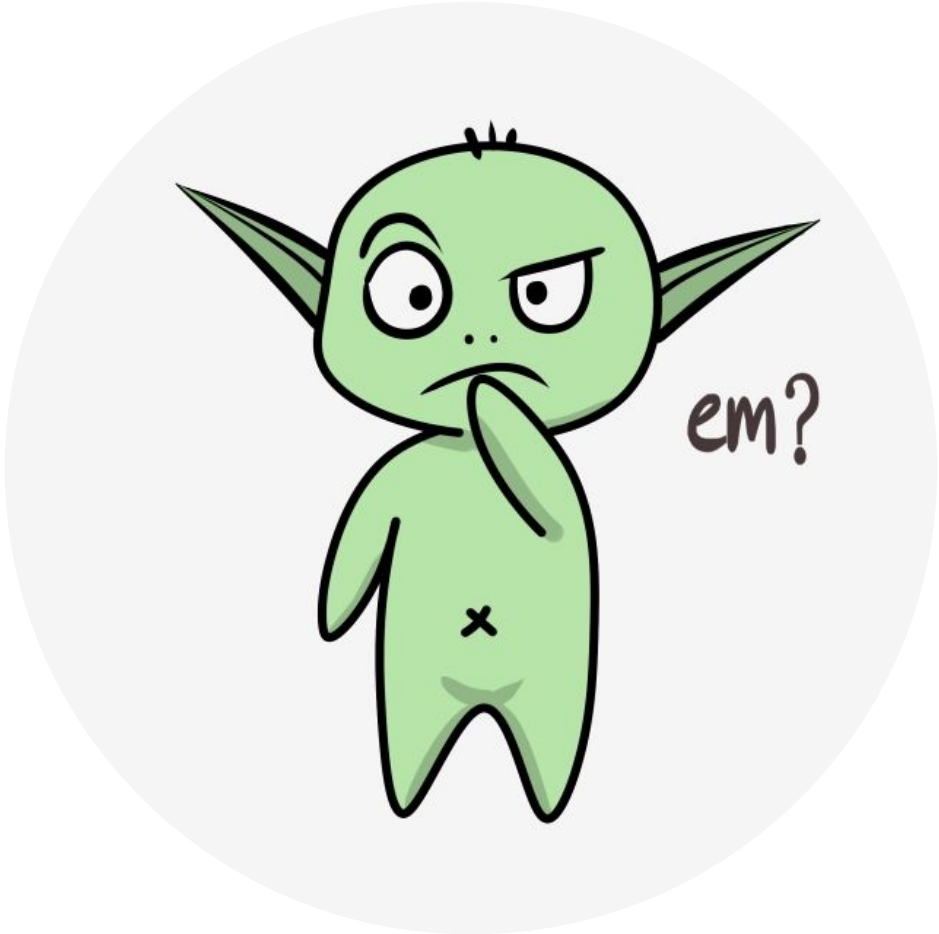
Policy Iteration: Interleaving Policy Eval. and Improvement

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 $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma V(s')\big]$
   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



https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Value Iteration

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

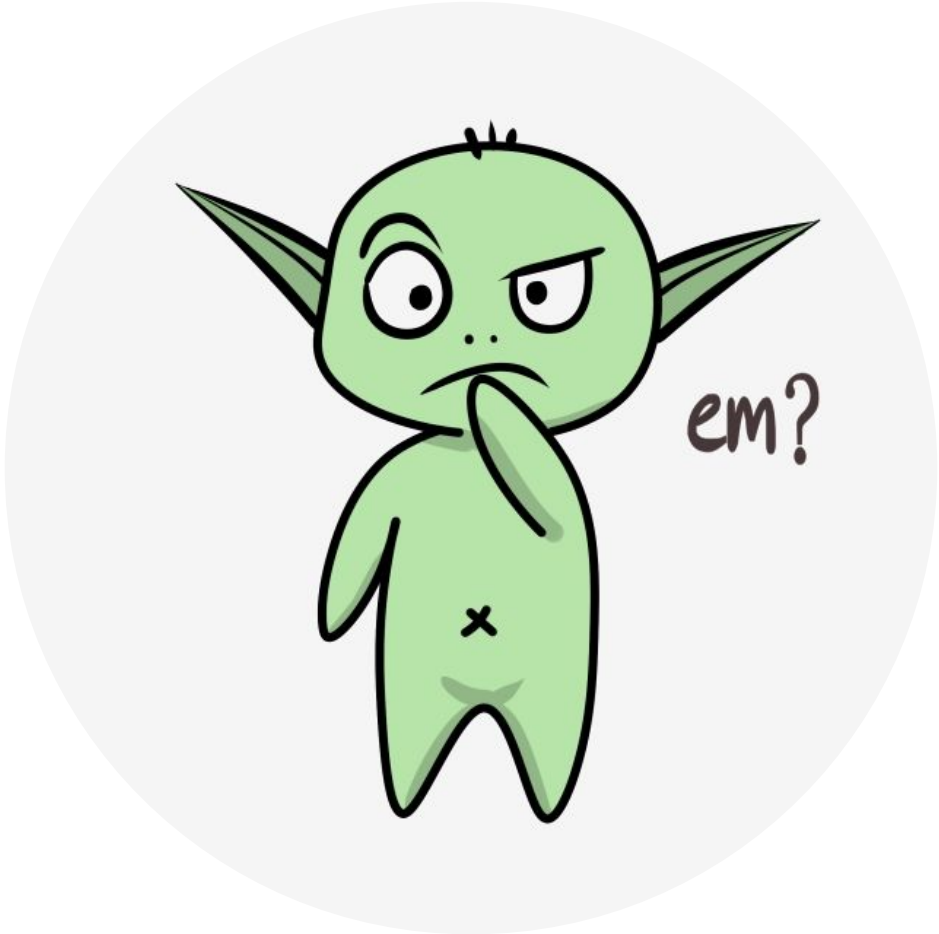
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

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

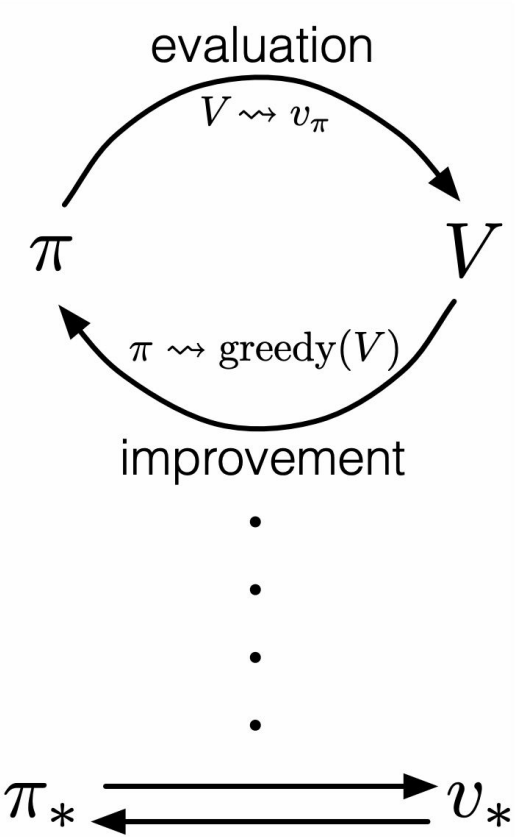
It doesn't need to be so synchronous

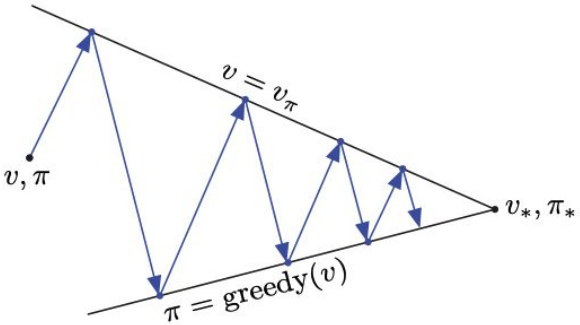
We just turned the Bellman optimality equation into an update rule!

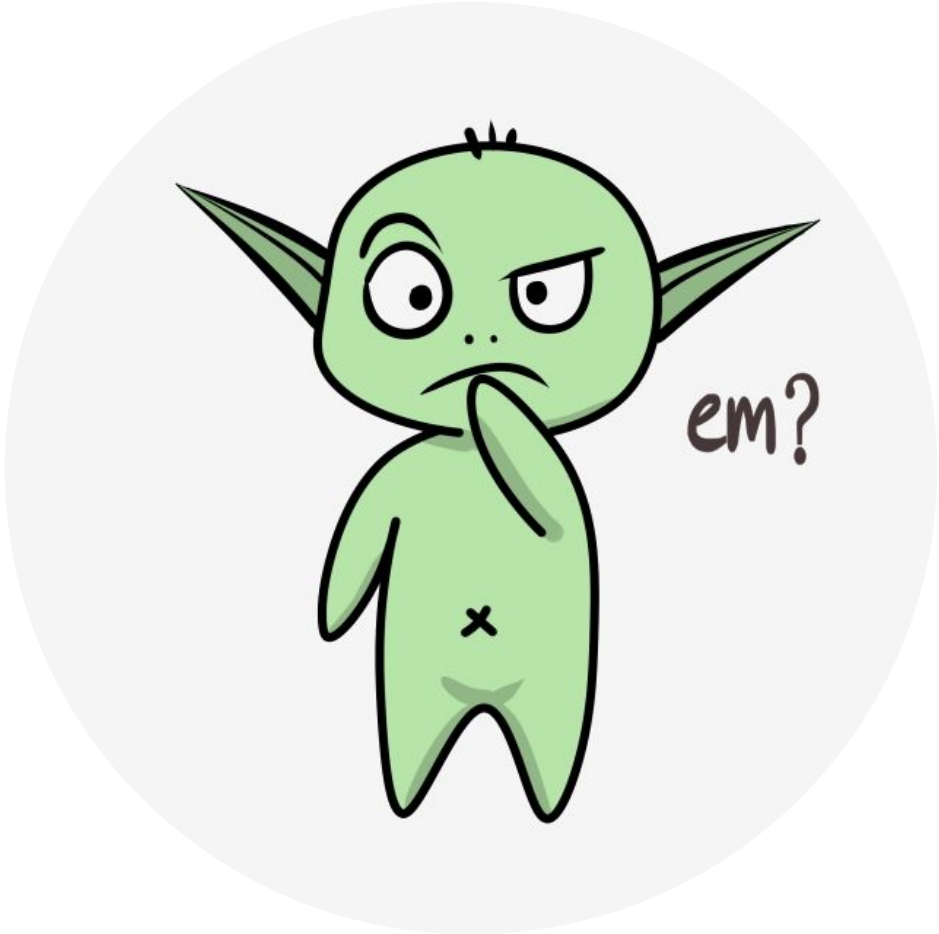


https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Generalized Policy Iteration

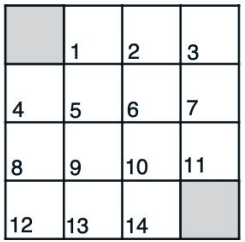






https://pngtree.com/freepng/question-expression-cartoon-illustration_4545209.ht

Example 4.1 Consider the 4×4 gridworld shown below.



$R_t = -1$
on all transitions

The nonterminal states are $\mathcal{S} = \{1, 2, \ldots, 14\}$. There are four actions possible in each state, $\mathcal{A} = \{\texttt{up}, \texttt{down}, \texttt{right}, \texttt{left}\}$, which deterministically cause the corresponding state transitions, except that actions that would take the agent off the grid in fact leave the state unchanged. Thus, for instance, $p(6, -1 | 5, \texttt{right}) = 1$, $p(7, -1 | 7, \texttt{right}) = 1$, and $p(10, r | 5, \texttt{right}) = 0$ for all $r \in \mathcal{R}$. This is an undiscounted, episodic task. The reward is -1 on all transitions until the terminal state is reached. The terminal state is shaded in the figure (although it is shown in two places, it is formally one state). The expected reward function is thus $r(s, a, s') = -1$ for all states s, s' and actions a. Suppose the agent follows the equiprobable random policy (all actions equally likely). The left side of Figure 4.1 shows the sequence of value functions $\{v_k\}$ computed by iterative policy evaluation. The final estimate is in fact v_π, which in this case gives for each state the negation of the expected number of steps from that state until termination. ■

Exercise 4.1 In Example 4.1, if π is the equiprobable random policy, what is $q_\pi(11, \texttt{down})$? What is $q_\pi(7, \texttt{down})$? □

Exercise 4.6 Suppose you are restricted to considering only policies that are ε-*soft*, meaning that the probability of selecting each action in each state, s, is at least $\varepsilon/|\mathcal{A}(s)|$. Describe qualitatively the changes that would be required in each of the steps 3, 2, and 1, in that order, of the policy iteration algorithm for v_* on page 80. □

Next class

- What **I** plan to do: Finish discussing Chapter 4 on Dynamic Programming
- What I recommend **YOU** to do for next class:
  - Complete the assigned reading: Chapter 4, §4.1-§4.4 (pp. 73-84); §4.6-§4.7 (pp. 86-89)
  - Complete Practice quiz and Progr. assignment (Optimal policies with dynamic programming)
    - It is due on Wednesday!
- Next classes:
  - Wednesday: Dynamic Programming
  - **Friday: Midterm**