Module 5: Lesson 3

Dynamic Programming: Policy iteration

Outline

- Policy iteration
- Policy iteration to solve the Gridworld problem.

Policy iteration

An alternative way to analyze the optimal course of action for the agent in a DP problem is to exploit the methodology of policy iteration.

This strategy features two nested loops:

- The inner loop fixes a policy $\pi(s)$ and finds the corresponding value function attached to that policy v_π.
- The outer loop iterates over policies using the value-maximization update, which yields a "greedy" action with respect to the current value function.

Policy iteration can be understood as a value iteration method that is divided into two steps:

- Value iteration "bootstraps" by updating the value of a state based on the optimal actions that arise from the previous estimates of the function.
- In contrast, policy iteration "waits" until having a full estimate of the value function for each policy before updating the optimal course of action.

Policy iteration: Pseudo-code

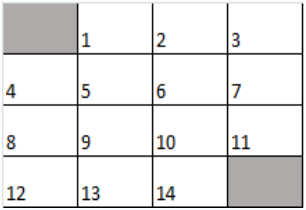
The algorithm can be described as follows:

Begin with some flat guess, say $v^{(0)}(s) = 0$, for all s, and an arbitrary policy $\pi^{(0)}(s)$.

- Policy evaluation step:
  1. Compute $v^{(1)}(s) = \sum_{s'} \mathbb{P}(s'|s, \pi^{(0)}(s)) \left[r(s, \pi^{(0)}(s)) + \gamma \, v^{(0)}(s') \right]$.
  2. Stop if $v^{(0)}$ and $v^{(1)}$ are close enough. Otherwise, set $v^{(0)} = v^{(1)}$ and return to step 1.
- Policy improvement step:
  1. Compute $\pi^{(1)}(s) = \arg\max_a \sum_{s'} \mathbb{P}(s'|s, a) \left[r(s, a) + \gamma \, v^{(1)}(s') \right]$.
  2. Stop if $\pi^{(0)}$ and $\pi^{(1)}$ are close enough. Otherwise, set $v^{(0)} = v^{(1)}$, $\pi^{(0)} = \pi^{(1)}$, and return to the Policy evaluation step.

Application: Gridworld

A useful setup to study the implementation of DP and RL algorithms is the gridworld.



The agent is located in one of the cells (states) in the grid. The agent must choose to move up, down, right, or left to exit the grid by reaching the gray cells ("terminal" states).

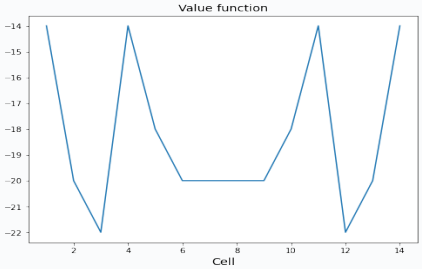
Each transition has a reward of -1 while reaching the terminal states has a value of 0. Thus, the value function captures the (negative) number of expected steps that are needed to exit the grid from a given cell.

The agent remains in the same state if any movement brings them "outside" of the grid.

Application: Policy evaluation in the first iteration

We begin the algorithm by posing a random policy that sets $\pi(s, a) = 0.25$ for all actions.

- Notice how we can determine stochastic policies $\pi(s, a)$ instead of deterministic policies $\pi(s)$.



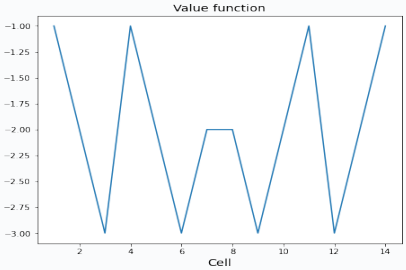
The random action yields, trivially, greater values to those cells (states) that are closer to the terminal cells.

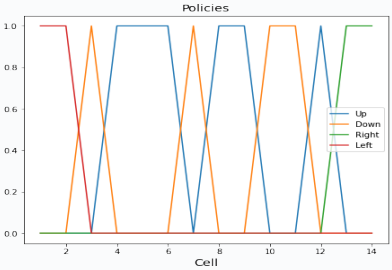
However, nodes that are one step away from the terminal cells have values of -14. This policy can be greatly improved!

Application: Optimal values and policies

The iteration scheme uncovers the neighborhood of each cell (state) to the terminal states and the shortest paths.

The optimal policies are not unique; the agent may reach the terminal states in the same number of steps on different paths.





Summary of Lesson 3

In Lesson 3, we have looked at:

- The policy iteration algorithm
- Application of policy iteration in the gridworld

⇒ **References for this Lesson:**

Sutton, Richard S., and Barto, Andrew G. *Reinforcement Learning: An Introduction.* MIT Press, 2018. (see Chapters 3 & 4)

TO DO NEXT: Now, please go to the associated Jupyter Notebook for this lesson to study the programming implementation of policy iteration to the gridworld.

In the next lesson, we will study an extension of the gridworld that includes random perturbations and asynchronous Dynamic Programming methods that reduce the computational burden of these techniques.