National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Faculty of Computing

CS 368: Reinforcement Learning

Lab 04: Markov Decision Processes - II

(Policy Iteration and Value Iteration)

Date: 11 October 2024

Time: 02:00 PM – 5:00 PM

Instructor: Dr. Zuhair Zafar

Lab 04: Markov Decision Processes - II

Objectives

Given a real-world environment, find out an optimal policy for each state using Policy iteration and Value Iteration Algorithms.

Tools/Software Requirement:

Google Colab, Python

Introduction

A **Markov decision process (MDP)** refers to a stochastic decision-making process that uses a mathematical framework to model the decision-making of a dynamic system. It is used in scenarios where the results are either random or controlled by a decision maker, which makes sequential decisions over time. MDPs evaluate which actions the decision maker should take considering the current state and environment of the system.

Policy Iteration

The common way that MDPs are solved is using **policy iteration** – an approach that is similar to value iteration which we have studied in theory class. While value iteration iterates over value functions, policy iteration iterates over policies themselves, creating a strictly improved policy in each iteration (except if the iterated policy is already optimal). Policy Iteration has the following two steps.

1) Policy iteration first starts with some (non-optimal) policy, such as a random policy, and then calculates the value of each state of the MDP given that policy — this step is called **policy evaluation**. It then updates the policy itself for every state by calculating the expected reward of each action applicable from that state.
   The basic idea here is that policy evaluation is easier to compute than value iteration because the set of actions to consider is fixed by the policy that we have so far. The important concept in policy iteration is **policy evaluation**, which is an evaluation of the expected reward of a policy. The expected reward of policy π from s, $V_\pi(s)$, is the weighted average of reward of the possible state sequences defined by that policy times their probability given π.

Definition – Policy evaluation

Policy evaluation can be characterised as $V^{\pi}(s)$ as defined by the following equation:

$$V^{\pi}(s) = \sum_{s' \in S} P_{\pi(s)}(s' \mid s)\,[r(s,a,s') + \gamma\, V^{\pi}(s')]$$

Iterative Policy Evaluation applies the expected updates for each state, iteratively, until the convergence to the true value function is met. Note that solving for $\boldsymbol{\mathcal{V}}$, is equivalent to solving a system of linear equations in $|\boldsymbol{\mathcal{S}}|$ unknowns, the $\boldsymbol{\mathcal{V}}$(s), with $|\boldsymbol{\mathcal{S}}|$ expected updates equations. The solution to the system of equations is the value function for the specified policy.

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 $V(s)$, for all $s \in \mathcal{S}^{+}$, arbitrarily except that $V(terminal) = 0$

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

2) The second step in policy iteration deals with **Policy improvement.** If we have a policy and we want to improve it, we can use policy improvement to change the policy (that is, change the actions recommended for states) by updating the actions it recommends based on $V_{\pi}(s)$ that we receive from the policy evaluation.

Let $Q_{\pi}(s,a)$ be the expected reward from state $\boldsymbol{s}$ when doing action $\boldsymbol{a}$ first and then following the policy π. Recall from the theory lectures on Markov Decision Processes that we define this as:

$$q_{\pi}(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{\pi}(s')$$

In this case, $V_{\pi}(s')$ is the value function from the policy evaluation.

If there is an action $\boldsymbol{a}$ such that $Q_\pi(s,a) > Q_\pi(s,\pi(s))$, then the policy π can be strictly improved by setting $\pi(s) \leftarrow a$. This will improve the overall policy.

Policy Iteration Algorithm

Pulling together policy evaluation and policy improvement, we can define **policy iteration**, which computes an optimal π by performing a sequence of interleaved policy evaluations and improvements:

Algorithm (Policy iteration)

Input : MDP $M = \langle S, s_0, A, P_a(s' \mid s), r(s,a,s') \rangle$
Output : *Policy* π

Set V^π to arbitrary value function; e.g., $V^\pi(s) = 0$ for all s
Set π to arbitrary policy; e.g. $\pi(s) = a$ for all s, where $a \in A$ is an arbitrary action

repeat
    Compute $V^\pi(s)$ for all s using policy evaluation
    for each $s \in S$
        $\pi(s) \leftarrow \text{argmax}_{a \in A(s)} Q^\pi(s,a)$
until π does not change

Value Iteration

One drawback to policy iteration is that each of its iterations involves policy evaluation, which may itself be a protracted iterative computation requiring multiple sweeps through the state set. This process can be expedited by using the value iteration algorithm.

Value Iteration is an algorithm used to solve RL problems where we have full knowledge of all components of the MDP. It works by iteratively improving its estimate of the 'value' of being in each state. It does this by considering the immediate rewards and expected future rewards when taking different available actions. These values are tracked using a value table, which updates at each step. Eventually, this sequence of improvements converges, yielding an optimal policy of state → action mappings that the agent can follow to make the best decisions in the given

environment.

Value Iteration leverages the concept of dynamic programming, where solving a big problem is broken down into smaller subproblems. To achieve this, the Bellman equation is used to guide the process of iteratively updating value estimates for each state, providing a recursive relationship that expresses the value of a state in terms of the values of its neighboring states.

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

for all $s \in \mathcal{S}$. For arbitrary v_0, the sequence $\{v_k\}$ can be shown to converge to v_* under the same conditions that guarantee the existence of v_*.

Value iteration formally requires an infinite number of iterations to converge exactly to v_*. In practice, we stop once the value function changes by only a small amount in a sweep. The box below shows a complete algorithm with this kind of termination condition.

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

Lab Tasks

Consider an autonomous backhoe loader (similar to an excavator, it has a digging bucket but additionally, it also has an attached blade for pushing earth and building debris for coarse surface grading). The backhoe loader is working in an off-road environment.

By using its time-of-flight cameras and laser scanners, the backhoe loader can recognize which type of terrain it is on. The sensors can recognize 2 types of terrains accurately: rocky tracks and ridges. The backhoe loader can perform the following tasks: (1) If there are rocks or ridges nearby, it can use its driller to drill. (2) It can dig the ground using its digging bucket. However, it cannot dig when it is on the ridges. (3) It can push the debris using its blade.

Whenever the backhoe loader is drilling on a rocky track, there is a 30 percent chance that the terrain does not change, and the reward, in that case, is +5. With a probability of 0.7, the drilling of a rocky track changes the terrain and forms a ridge with a reward of +1. If the backhoe loader drills on a ridge, there is a 40 percent chance that the terrain never changes with a reward of +2 and a 60 percent chance that the ridges are converted into a rocky track with a reward of +6.

If the backhoe loader decides to dig the ground on a rocky track, there is a 75 percent chance that the terrain does not change with a reward of +7. It is possible with 0.25 probability, that digging the ground eventually forms a ridge, and the reward, in this case, is +1.

If the backhoe loader pushes the rocky debris on a rocky track, there is a 45 percent chance that the terrain does not change with a reward of +9. There is a 55 percent chance that pushing the rocky debris using its blade can form a ridge and the reward, in this case, is +5. However, if the backhoe loader pushes the debris on a ridge, there is an 80 percent chance that the terrain does not change with a reward of +2. With a 0.2 probability and reward of +10, pushing the debris on a ridge can change the terrain and form a rocky track.

1. **You are initially provided with the following deterministic policy. Implement the policy iteration algorithm to determine the optimal value for each state and the corresponding optimal policy. Assume a discount factor, $\gamma = 0.9$.**

State (s)	$\pi(s)$
Rocky Track	Drill
Ridge	Push

2. **You are provided with a uniform random stochastic policy. Apply the policy iteration algorithm to compute the optimal value for each state and the optimal policy. Assume a discount factor, $\gamma = 0.9$).**

3. **Using a discount factor of 0.9, solve the MDP using Value Iteration (until the values have become reasonably stable). You should start with the values set to zero. You should show both the optimal policy and the optimal values.**

Deliverable:

Please submit your notebook on LMS before the deadline **(14th October 2024, 11:59pm).**

Lab Rubrics

Assessment	**Does not meet expectation** **(1/2 marks)**	**Meets expectation** **(3/4 marks)**	**Exceeds expectation** **(5 marks)**
Software Problem Realization **(CLO1, PLO1)**	The student struggles to formulate the problem as MDP and does not apply policy / value iteration algorithm to solve it. There is a lack of demonstration of the problem's data requirements and no attempt to model the given environment effectively.	The student formulates the problem as MDP with some guidance, applies policy & value iteration algorithms with hints, and shows it's working. However, the approach might not be fully optimized or lacks a thorough justification.	The student independently formulates the given problem as MDP, applies policy iteration & value iteration algorithms without guidance, and effectively models the environment. The approach is fully optimized and can be applied to different similar problems.
Software Tool Usage **(CLO4, PLO5)**	Code has syntax errors, and the implementation of the Policy iteration / Value Iteration algorithm is incorrect or incomplete. The code is not modular and lacks comments for readability and reuse. The student	The student has implemented the Policy & Value iteration algorithms correctly for the given problem with minor mistakes. The code is mostly correct in terms of syntax and functionality but might not be optimized or well-structured for reuse.	The student has implemented the Policy & Value iteration algorithms efficiently and correctly. The code is clean, modular, well-documented, and follows best practices for reproducibility and reuse. The student demonstrates

	shows no ability to use relevant libraries where required.	Some documentation is provided. The student also shows working knowledge of relevant libraries where required.	full command of libraries and tools.