First Mini-Project, Dynamic Programming

Arya Ebrahimi
Reinforcement Learning Spring2023

Abstract – **In this mini-project, we investigate DP methods to compute optimal policies and compare their performance in different problem settings. First, the policy and value iteration algorithms are implemented and tested using the default OpenAi gym FrozenLake environment, and in the next part, a comparison is made using different reward systems and parameters.**

Index Terms – Dynamic Programming, Policy Iteration, Value Iteration, GPI

Introduction

DP methods are feasible only when a perfect model of the environment as an MDP is available. This perfect model (dynamics of MDP) can be represented as a set of probabilities $p(s', r|s, a)$, for all $s \in S$, $a \in A$, $r \in R$ and $s' \in S^+$. DP allows agents to quickly explore the state-action space of an MDP and discover optimal policies. DP techniques in RL can be divided into two basic categories: policy evaluation and policy improvement. While policy improvement algorithms employ the value function to enhance the current policy, policy evaluation algorithms compute the value function of a given policy.

Policy Evaluation

The goal of policy evaluation is to determine each state's contribution to a certain policy. This can be done by repeatedly updating a value function estimate based on the Bellman equation, which links a state's value to the values of its succeeding states.

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

Policy evaluation enables the agent to evaluate the effectiveness of the policy and determine whether it needs to be improved by computing the value function for a specific policy. Several dynamic programming methods, such as value iteration and policy iteration, depend on it as a key component.

Policy Improvement

The goal of policy improvement is to create a new policy that is greedily derived from the estimated value function of the current policy. The new policy is created by choosing the course of action that, based on the most recent estimate of the value function, maximizes the expected return in each state.
The policy improvement algorithm can be summarized as follows:

1. Given a value function estimate V, for each state m in S, select the action a that maximizes the expected return:

$$a = argmax_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$$

2. Construct a new policy that selects the actions calculated in step 1.
3. Repeat steps 1-2 until convergence

By maximizing the expected return at each state, the policy improvement method iteratively improves the policy. The agent can obtain a series of better policies, each with a larger expected return than the last one, by continually performing the policy improvement method.

Policy Iteration

Policy iteration combines policy evaluation and policy improvement to Find the best policy in an MDP that maximizes expected return.
The policy iteration repeats both of these steps until there is no longer any change in the policy, which denotes convergence to the optimal policy.

GPI & value iteration

In contrast to policy iteration, generalized policy iteration integrates policy evaluation and policy improvement procedures in a more comprehensive and flexible way. The goal of GPI is to execute policy evaluation and improvement iteratively but in any sequence and using appropriate sub-algorithms for the issue at hand.

The fundamental notion underlying GPI is that we can do policy evaluation and policy improvement iteratively in a flexible way to utilize any method that performs best for the situation instead of utilizing a fixed policy for many iterations.

In the framework of GPI, a particular dynamic programming approach called value iteration is often used. Value iteration is a sort of iterative policy evaluation in which the Bellman optimality equation is continually applied until convergence, updating the value function estimate for each state.
The value iteration algorithm can be summarized as follows:

1. Initialize the value function V for all states s.
2. Repeat until convergence:
   - For each state s in state space update its value function estimate as follows:
     $$V(s) \leftarrow max_a \sum_{s',r} p(s',r|s,a)[r+\gamma V(s')]$$
3. Output a deterministic policy such that:
   $$\pi(s) = argmax_a \sum_{s',r} p(s',r|s,a)[r+\gamma V(s')]$$

Value iteration is just policy iteration when policy evaluation is stopped after just one sweep.

Experiments

Table 1: Different results of the first run of DP algorithms.

Mode	Gamma	Slippery	Step reward	Hole reward	Steps	Reward
PI	0	False	0	0	100	0
VI	0	False	0	0	100	0
PI	0.9	False	0	0	6	1
VI	0.9	False	0	0	6	1
PI	1	False	0	0	100	0
VI	1	False	0	0	100	0
PI	0.9	True	0	0	11	1
VI	0.9	True	0	0	73	0
PI	0.9	False	-0.05	0	6	0.75
VI	0.9	False	-0.05	0	6	0.75
PI	0.9	True	-0.05	0	28	-0.35
VI	0.9	True	-0.05	0	29	-1.4
PI	0.9	False	0	-2	6	1
VI	0.9	False	0	-2	6	1
PI	0.9	True	0	-2	26	1
VI	0.9	True	0	-2	100	0
PI	0.9	False	-0.05	-2	6	0.75
VI	0.9	False	-0.05	-2	6	0.75
PI	0.9	True	-0.05	-2	8	0.65
VI	0.9	True	-0.05	-2	42	-1.05

By setting the gamma value to 0, state values will only depend on the immediate reward; Thus, only the value of the previous state of the goal state would be one, and all the others would be 0. Taking the argmax results in staying in the current state. (or by selecting random argmax, it will act stochastically in states with a value of 0) As can be seen in Table1, the number of steps for PI and VI is reported as 100, which is the cutoff value for the FrozenLake environment, and the episode is truncated for $\gamma = 0$.

On the other hand, by choosing $\gamma = 1$, state values will take the total value of the goal state, which makes all the non-terminal states have a value of 1. A similar explanation to the case of $\gamma = 0$ can be considered for this situation.
A feasible discount factor could be 0.9, which results in an optimal policy, and we consider this as a default gamma for the other experiments.

Since, in other experiments, the environment would be slippery in some cases(stochastic), another run is made to become more confident about the reported values.

Table 2: Different results of the second run of DP algorithms.

Mode	Gamma	Slippery	Step reward	Hole reward	Steps	Reward
PI	0	False	0	0	100	0
VI	0	False	0	0	100	0
PI	0.9	False	0	0	6	1
VI	0.9	False	0	0	6	1
PI	1	False	0	0	100	0
VI	1	False	0	0	100	0
PI	0.9	True	0	0	79	1
VI	0.9	True	0	0	39	0
PI	0.9	False	-0.05	0	6	0.75
VI	0.9	False	-0.05	0	6	0.75
PI	0.9	True	-0.05	0	36	-1.75
VI	0.9	True	-0.05	0	87	-4.3
PI	0.9	False	0	-2	6	1
VI	0.9	False	0	-2	6	1
PI	0.9	True	0	-2	19	1
VI	0.9	True	0	-2	21	1
PI	0.9	False	-0.05	-2	6	0.75
VI	0.9	False	-0.05	-2	6	0.75
PI	0.9	True	-0.05	-2	97	-3.8
VI	0.9	True	-0.05	-2	69	-2.4

We consider both stochastic and deterministic environments for the next experiments. As can be seen in table1 and table2, in stochastic environments, we might choose an action to take, but different action could be taken in real. Because of this, episodes are longer, and there is no guarantee to reach the goal state because the randomness of the environment could cause us to slip into the holes no matter what the rewards are. In contrast, in a deterministic environment, all the episodes end in 6 steps following an optimal policy.