MDPs and RL - CS 4641

Omar Shaikh

April 16, 2019

Abstract

In this report, I explore three RL methods of interest. The first two are Policy and Value iteration, both rooted in Dynamic Programming Techniques. The last method, Q-learning, utilizes TD learning unlike the first two. I compare and contrast both value and policy iteration in two MDPs; finally, I conclude with an exploration of Q-learning strategies and a general comparative analysis.

1 Markov Decision Processes

A state S_t is Markov if and only if

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t] \tag{1}$$

A future is independent of the past given the present. We use state transition matrix P to define transition probabilities from all states s to all successor states s'

$$P_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s] \tag{2}$$

The following MDP follows Markov assumption.

1.1 Frozen Cliff MDP

The general MDP used in this project is a slightly altered version of the Frozen Cliff MDP taken from OpenAI's gym.

> Winter is here. You and your friends were tossing around a frisbee at the park when you made a wild throw that left the frisbee out in the middle of the lake. The water is mostly frozen, but there are a few holes where the ice has melted. If you step into one of those holes, you'll fall into the freezing water. At this time, there's an international frisbee shortage, so it's absolutely imperative that you navigate across the lake and retrieve the disc. However, the ice is slippery, so you won't always move in the direction you intend. The surface is described using a grid like the following (1a)
>
> S : starting point, safe
> F : frozen surface, safe
> H : hole, fall to your doom
> G : goal, where the frisbee is located
>
> The episode ends when you reach the goal or fall in a hole. You receive a reward of 1 if you reach the goal, and zero otherwise.

The slight alteration comes in the form of a small negative living reward, to enforce a horizon on the MDP. If the agent doesn't fall into a hole, but doesn't reach it's goal-state, it receives a small pre-defined penalty. My attached code was stolen mostly from the cited GitHub repository.

1.2 Problem Instances and Interest

The following two frozen lakes highlight our MDPs. For my environments, I added a custom transition matrix defined in 3, where the intended action x is parameterized by probability p.

For both my MDPs (1b and 1a) I have a single goal state in the bottom right corner. However, with the larger MDP, there are more (terminal) states which will require more time for our RL algorithms to converge.

```
SFFF
FHFH
FFFH
HFFG
```

(a) Small MDP (4x4)

```
SFFFFFFHHHFFFFFFFFFF
FFFFFFFFFFFFFFFFHHFF
FFFHFFFFFFFHHFFFFFFF
FFFFFHFFFFFFFFFFHHFF
FFFFFHFFFFFFFFFFHHFF
FFFFFHFFFFFFFFFFHHFF
FFFFFFFFHFFFFFFFHHFF
FFFFFHFFFFHHFFFFHHFF
FFFFFHFFFFFFFFFFHHFF
FFFFFHFFFFFFFFFFHHFF
FFFFFFFFFFFHHHHHHHFF
HHHHFHFFFFFFFFFFHHFF
FFFFFHFFFFHHHFFFHHFF
FFFFFFFFFFFFFFFFHHFF
FFFFFHFFFFFFHFFFHHFF
FFFFFHFFFFFFFFFFHHFF
FFFFFFFFFFFHFFFFFFFF
FHHFFFHFFFFHFFFFFHFF
FHHFHFHFFFFFFFFFFFFF
FFFHFFFFFHFFFFHHFHFG
```

(b) Large MDP (20x20)

Figure 1: Frozen Cliff Environments

Furthermore, there are several paths we can see in the large MDP, and the solution is somewhat vague. With the small MDP, the optimal policy can be eyeballed given parameters.

$$S_{t+1}(x) = \begin{cases} x, & \text{with probability } 1-p \\ rotate(x, 90°), & \text{with probability } \frac{p}{2} \\ rotate(x, -90°), & \text{with probability } \frac{p}{2} \end{cases} \tag{3}$$

Note that this MDP is quite similar to GridWorld detailed in literature, except that the transition matrix doesn't account for the agent accidentally moving backward (we assume that the person looking for the frisbee won't mess up too badly). Because of this, we can directly rely on prior observations from GridWorld (due to the research attention this problem has received), while exploring a slightly different MDP formulation.

2 Computer Specifications

The computer used to train and test has an i7-7700HQ (clock speeds @ 3.8 GHZ), 32 GB of RAM, and an Nvidia GTX 1070 with 8 GB of VRAM. Whenever it was possible, I used all CPU cores. Runtimes of algorithms should be considered in context of these specifications.

3 Dynamic Programming RL

The following subsections cover Reinforcement Learning Algorithms that rely on Bellman's equations and Dynamic Programming techniques.

In the case of value iteration, the optimal value for each state is calculated, where the value is defined as the sum of maximum reward following the best policy. In the case of value iteration, the best policy can be defined by greedily picking a route of the maximum value at each state. Note that we are not finished until each value for every state has converged (see algorithm 1).

With policy iteration, however, we pick a random initial policy and solve a set of linear equations to justify actions at every step in our policy. When our actions stop changing (e.g all states have been "justified") we converge at our final policy. Note, however, that each iteration takes more effort on the part of policy iteration when compared to value iteration due to solving this system of equations (see algorithm 2).

For both problem one and two, the step reward (where the next state is not a hole or the goal state) is -.1. I will explore no step reward (and therefore no event horizon) in the comparative analysis.

Algorithm 1 Value Iteration: Learn function $J : \mathcal{X} \to \mathbb{R}$

```
Require:
  States $\mathcal{X} = \{1, \dots, n_x\}$
  Actions $\mathcal{A} = \{1, \dots, n_a\}$,    $A : \mathcal{X} \Rightarrow \mathcal{A}$
  Cost function $g : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$
  Transition probabilities $f_{xy}(a) = \mathbb{P}(y|x, a)$
  Discounting factor $\alpha \in (0, 1)$, typically $\alpha = 0.9$
  procedure ValueIteration($\mathcal{X}$, $A$, $g$, $f$, $\alpha$)
      Initialize $J, J' : \mathcal{X} \to \mathbb{R}_0^+$ arbitrarily
      while $J$ is not converged do
          $J' \leftarrow J$
          for $x \in \mathcal{X}$ do
              for $a \in A(x)$ do
                  $Q(x, a) \leftarrow g(x, a) + \alpha \sum_{j=1}^{n_x} f_{xj}(a) \cdot J'(j)$
          for $x \in \mathcal{X}$ do
              $J(x) \leftarrow \min_a\{Q(x, a)\}$
      return $J$
```

Algorithm 2 Policy Iteration: Learning a policy $\pi : \mathcal{X} \to \mathcal{A}$

```
Require:
  States $\mathcal{X} = \{1, \dots, n_x\}$
  Actions $\mathcal{A} = \{1, \dots, n_a\}$,    $A : \mathcal{X} \Rightarrow \mathcal{A}$
  Cost function $g : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$
  Transition probabilities $f$, $F$
  $\alpha \in (0, 1)$
  procedure PolicyIteration($\mathcal{X}$, $A$, $g$, $f$, $F$, $\alpha$)
      Initialize $\pi$ arbitrarily
      while $\pi$ is not converged do
          $J \leftarrow$ solve system of linear equations $(I - \alpha \cdot F(\pi)) \cdot J = g(\pi)$
          for $x \in \mathcal{X}$ do
              for $a \in A(x)$ do
                  $Q(x, a) \leftarrow g(x, a) + \alpha \sum_{j=1}^{n_x} f_{xj}(a) \cdot J(j)$
          for $x \in \mathcal{X}$ do
              $\pi(x) \leftarrow \arg\min_a\{Q(x, a)\}$
      return $\pi$
```

3.1 Problem 1a

3.2 Problem 1b

3.3 Comparative Analysis

4 TD Learning

4.1 Q-Learning