Reinforcement Learning

COMS4047A/COMS7053A

Lab 1 - Dynamic Programming

Tamlin Love (tamlin.love1@students.wits.ac.za)
Shahil Mawjee (shahil.mawjee@students.wits.ac.za)

Overview

The lab will focus on the dynamic programming methods used in reinforcement learning. Given the complete model of the environment, can we compute the optimal policy?

We will be using a grid-world environment developed using the OpenAI gym framework. The grid-world will be of size 5 by 5, with the goal in the bottom right corner.

Goals:

- Understand OpenAI Gym environment framework.
- Implement the collection of dynamic programming algorithms for reinforcement learning.
- Understand the role of γ in RL

Submission

Work in groups of 3 to 4 people. List group members at the top of the file. Ensure your python file is named `lab_1.py` and that the method names and parameters remain the same as in the provided skeleton.

Once your have completed the lab, zip your code file and the plot of exercise 4.1.2. Submit your zip file to Moodle.

1 OpenAI Gym

OpenAI Gym is a python library that offers a standardised, shared interface to a large collection of RL environments, including the grid-world environment

used in this lab. By abstracting away the environment, gym allows us to focus our attention on implementing the actual RL algorithms.
Gym can be installed using `pip install gym`. From there, we can set up the grid-world environment of size 5 by 5, with the goal at state 24, a terminal reward of 0 and a step reward of -1.

```python
from environments.gridworld import GridworldEnv

env = GridworldEnv(shape=[5, 5], terminal_states=[24],
    terminal_reward=0, step_reward=-1)
```

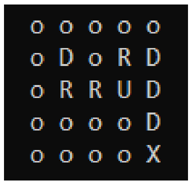
To reset the environment to its initial state, we can call `env.reset()`, which also returns the initial state. To visualise the current state of the environment, we can call `env.render()`.
For the grid-world environment, `env.P` is the transition function, where `env.P[state][action]` is the tuple `(prob,next_state,reward,done)`.

1.1 Exercise

1. Using the gym environment, generate a trajectory with a uniform random policy.

2. Print the direction of the action a_t taken at state s_t for each state visited in the trajectory, shaped as the grid.

   For example, your trajectory might look like this:



```
o o o o o
o D o R D
o R R U D
o o o o D
o o o o X
```

2 Policy Evaluation

The first step to improving a policy is to evaluate the state-value function v_π for an arbitrary policy (see Sutton & Barto 4.1 for details). The state-value according to a policy is computed as:

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

The value of v_π can be updated iteratively for all s:

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

For any policy π, your task to to implement a policy evaluation function based on the following pseudocode:

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

```
Input $\pi$, 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 $\Delta < \theta$
```

Figure 1: Source - Sutton & Barto Section 4.1, page 75

2.1 Exercise

Complete the `policy_evaluation` function. The function will take in gym environment and the policy as input and return value function.

Call the function with the grid-world environment and a random policy.

3 Policy Iteration

Now that we can evaluate the state value function for any given policy, we can use the values to improve our policy.

First consider the state-action value function (q-function), which consists of selecting a in s and then behaving according to π:

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

Let π' be a policy such that, for all $s \in S$:

$$q(s, \pi'(s)) \geq v_\pi(s)$$

Then the following holds for all $s \in S$:

$$v'_\pi(s) \geq v_\pi(s)$$

(See Sutton & Barto page 78 for proof)

Consider a simple policy improvement which consists of π selecting an action according to the maximum state-action value:

$$\begin{aligned}\pi'(s) &= \text{argmax}_a \, q_\pi(s,a) \\ &= \text{argmax}_a \, \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \,|\, S_t = s, A_t = a\right] \\ &= \text{argmax}_a \sum_{s',r} p(s', s \,|\, s, a)[r + \gamma v_\pi(s')]\end{aligned}$$

Alternating between policy evaluation and policy improvement is known as **policy iteration**. Implement the algorithm based on the following pseudocode:

```
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 $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
       $\Delta \leftarrow 0$
       Loop for each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s))\big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   $\textit{policy-stable} \leftarrow \textit{true}$
   For each $s \in \mathcal{S}$:
       $\textit{old-action} \leftarrow \pi(s)$
       $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
       If $\textit{old-action} \neq \pi(s)$, then $\textit{policy-stable} \leftarrow false$
   If $\textit{policy-stable}$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
```

Figure 2: Source - Sutton & Barto Section 4.3, page 80

3.1 Exercise

Complete the `policy_iteration` function. The function will take in a gym environment and the policy evaluation function as input and return the improved policy and value function.

Call the function with the grid-world environment and the `policy_evaluation` function.

Side thought: when breaking ties, if the values are the same, does it mean the policy is unstable?

4 Value Iteration

The issue with policy iteration is that every time we improve our policy, we must do a full sweep through all states again to evaluate the new policy. This is extremely inefficient. Value iteration combines both evaluation and improvement in the same step. It can be written as a particularly simple update operation:

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \,|\, S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p(s', r \,|, s, a)\left[r + \gamma v_\pi(s')\right] \end{aligned}$$

Value iteration based on the following pseudocode

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

Output a deterministic policy, $\pi \approx \pi_*$, such that
   $\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
```

Figure 3: Source - Sutton & Barto Section 4.4, page 83

4.1 Exercise

1. Complete the `value_iteration` function. The function will take in a gym environment as input and return the policy and value function. Call the function with the grid-world environment.

2. Plot the average running time for Policy Iteration and Value Iteration by varying the discount rate γ.

   (a) The x-axis should be the discount rate. The range of discounts should be specified by `np.logspace(-0.2, 0, num=30)`

(b) The y-axis is the average time in seconds. Average times over 10 runs.