

Exercise 2

Value Functions, Dynamic Programming and Optimal Policies

1 Policy and Value Iteration

In this exercise you will implement important dynamic programming approaches for solving MDPs. You will use them to solve a simple gridworld MDP similar to the one shown in Figure 1. For quick reference, the respective algorithms are outlined in more detail in Figures 2, 3 and 4.¹

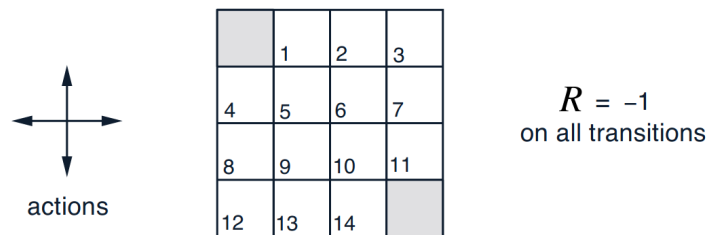


Figure 1: A gridworld MDP. Upper left and lower right states are terminal states.

Programming Tasks:

1. **Policy evaluation (one step)** Fill in code inside `policy_evaluation_one_step`, which performs exactly one step of policy evaluation (the inner part of the loop inside Figure 2).
2. **Policy evaluation** Using `policy_evaluation_one_step`, implement the method `policy_evaluation` which iteratively performs one step of policy evaluation until the changes in the value function estimate are smaller than some predefined threshold.
3. **Policy improvement** Continue with the `policy_improvement` method, which returns the optimal greedy policy w.r.t a given value function.
4. Utilizing your results from 1. - 3. implement:
 - (a) the **policy iteration** (PI) algorithm (Figure 3) inside `policy_iteration`
 - (b) the **value iteration** (VI) algorithm (Figure 4) inside `value_iteration` (*Hint*: You can either do this the *easy* way and reuse methods of 1. - 3. or do it the *efficient* way by closely following the algorithm outline of Figure 4).

¹We recommend reading the first chapters of Sutton (“Reinforcement Learning: An Introduction“ by Richard S. Sutton and Andrew G. Barto, 1998) for an excellent introduction into the topic.

```
Input  $\pi$ , the policy to be evaluated
Initialize an array  $V(s) = 0$ , for all  $s \in \mathcal{S}^+$ 
Repeat
   $\Delta \leftarrow 0$ 
  For each  $s \in \mathcal{S}$ :
     $v \leftarrow V(s)$ 
     $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$ 
     $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
until  $\Delta < \theta$  (a small positive number)
Output  $V \approx v_\pi$ 
```

Figure 2: Policy evaluation.

```
1. Initialization
   $V(s) \in \mathbb{R}$  and  $\pi(s) \in \mathcal{A}(s)$  arbitrarily for all  $s \in \mathcal{S}$ 

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

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

Figure 3: Policy iteration.

2 Optimal Policies

Try to answer the following questions concerning value functions and optimal policies.

Questions:

1. Imagine you know the optimal state-value function V of an MDP but you have no information about the state transition probabilities P . Could you derive the optimal policy from the state-value function V ?
2. What would change, if you would have access to the optimal action-value function Q instead? ²
3. Can there be multiple optimal deterministic policies for one MDP? If so, explain the conditions under which this would be the case.

²This is the fundamental idea of a very important method (or rather class of methods) that goes under the name of Q-Learning. The modern version of this utilized neural networks as an action-value function approximator, thus the name Deep Q-Learning (DQN) (“Playing Atari with Deep Reinforcement Learning“ by Mnih et al., 2013). Stay tuned!

```
Initialize array  $V$  arbitrarily (e.g.,  $V(s) = 0$  for all  $s \in \mathcal{S}^+$ )

Repeat
   $\Delta \leftarrow 0$ 
  For each  $s \in \mathcal{S}$ :
     $v \leftarrow V(s)$ 
     $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$ 
     $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
until  $\Delta < \theta$  (a small positive number)

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

Figure 4: Value iteration.