



Department of Electrical & Computer Engineering
2022SPRING - Mathematical and Biological Foundation of RL

**Project: find optimal policies of in a grid
world**

Prepared by: 20211203076

Date: May 9, 2022

Contents

1	Problem description	1
2	Assignment 1	1
2.1	Environment setup	1
2.2	Agent	1
2.3	Results	2
3	Assignment 2	2
3.1	The parameters of the Bellman equation in matrix-vector form	2
3.2	Solve the state values by using the closed-form solution	4
3.3	Solve the state values by using the iterative solution.	4
3.4	Discussion	6
4	Assignment 3: model-based algorithms	6
4.1	Value iteration	7
4.2	Policy iteration	8
5	model-free algorithms	8
5.1	Value-based methods	8
5.2	Policy-based methods	10
6	Appendix	12

1 Problem description

In a 4x4 grid world as illustrated in Fig 1.1. The red, blue, and green represent forbidden areas, the target, and the agent respectively. The agent's goal is to reach the target with the least steps within the constraints of forbidden areas.

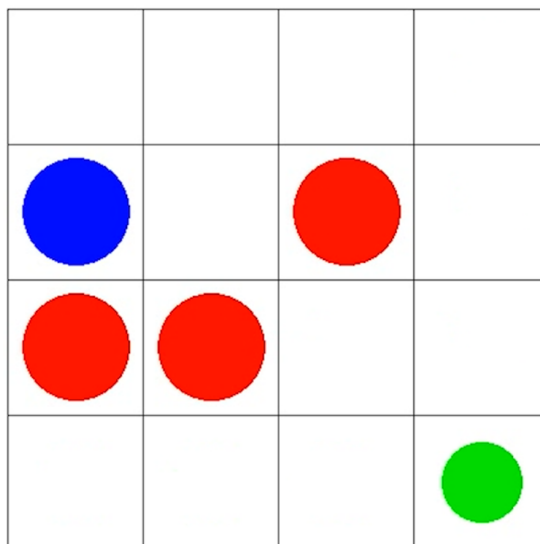


Figure 1.1: Grid world

For the agent, actions that lead to forbidden areas or boundaries are excluded. Therefore the forbidden areas do not correspond to the state.

2 Assignment 1

I build two files corresponding to the environment and agent, which will be introduced in the following subsections.

2.1 Environment setup

The environment can interact with the agent, and also visualize the grid world with the inheriting of the class `gym.Env`.

The states are set to be a list of 0-15 and remove 6, 8, 9 three states. The actions are set to be an array of `[-4, 4, -1, 1]`, which represents the agent going up, down, left, and right. The reward is set to be -1 corresponding to all pairs of actions and states, excluded 100 for the pair leading to the target.

2.2 Agent

The agent's starting state is 15. In every step, the agent observes the state of itself and the environment and takes action according to a different policy. In assignment 1,

I define additional two deterministic policies as in Fig 2.1. The agent will choose two policies in turn.

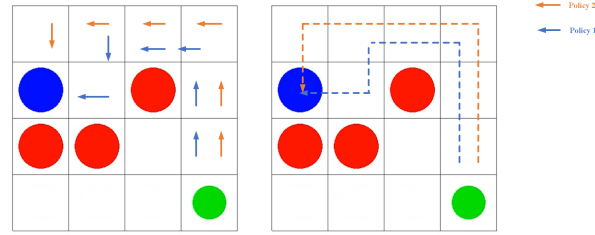


Figure 2.1: Trajectories of the two policies

2.3 Results

The discounted return of the two deterministic policies is illustrated in Fig. 2.2.

```
DiscountedReturn: 22.525120000000001
1 episode
value: [100.      79.      62.2     48.76     0.
100.      79.      38.008    0.      0.
-3.10848816 29.4064   -5.      -4.99999999 -4.99979886
22.52512   ]
```

Figure 2.2: The discounted return

3 Assignment 2

In this assignment, I append the action "Don't move" and choose the two policies as in Fig. 3.1, one of which is not the optimal policy.

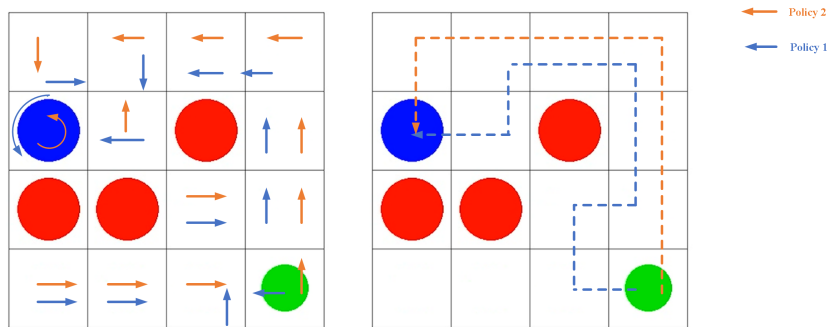


Figure 3.1: The new two policies

3.1 The parameters of the Bellman equation in matrix-vector form

The matrix-vector form of the Bellman equation can be written as :

$$v_{\pi} = r_{\pi} + \gamma P_{\pi} v_{\pi} \quad (1)$$

3.2 Solve the state values by using the closed-form solution

The closed-form solution can be written as

$$v_{\pi} = (I - \gamma P_{\pi})^{-1} r_{\pi} \quad (2)$$

For the two policies, we have:

$$v_{\pi 1} = \begin{bmatrix} 59.64 \\ 75.8 \\ 59.64 \\ 46.71 \\ -5. \\ 96. \\ 0. \\ 36.37 \\ 0. \\ 0. \\ 21.48 \\ 28.1 \\ 8.56 \\ 11.94 \\ 16.18 \\ 11.94 \end{bmatrix}, v_{\pi 2} = \begin{bmatrix} 96. \\ 75.8 \\ 59.64 \\ 46.71 \\ -5. \\ 59.64 \\ 0. \\ 36.37 \\ 0. \\ 0. \\ 21.48 \\ 28.1 \\ 8.56 \\ 11.94 \\ 16.18 \\ 21.48 \end{bmatrix} \quad (3)$$

3.3 Solve the state values by using the iterative solution.

The Bellman equation also can be written as iteration form:

$$v_s = r_{\pi} + \gamma P_{\pi} v_{s+1} \quad (4)$$

```

num_episodes = 100
max_number_of_steps=20

for p in [1, 2]:
    agent.vtab = np.zeros(len(agent.vtab))
    for i in range(num_episodes):
        env.reset() #random choose the init state , or the agent
                     → will not go through all states

        for t in range(max_number_of_steps):
            env.render()

            #chosenAction
            chosenAction=agent.deterministicPolicy(env.state,p)
            nextstate, reward, Terminal=env.step(chosenAction)

            #Value iteration
            agent.vtab[env.state]=reward+agent.gamma*agent.vtab[next
            → state] #Dynamic Programming -Value
            → iterations
            env.state=nextstate

print('Iteration_solution_Policy
    → ',p,':\n',agent.vtab.reshape(len(agent.vtab),1))

```

Listing 3.2: Code that calculates the state value in iteration solution

Do the value iteration for every step and randomly choose the initial state to ensure

the agent will go through all the states. After calculation, we have:

$$v_{\pi 1} = \begin{bmatrix} 59.64 \\ 75.8 \\ 59.64 \\ 46.71 \\ -5. \\ 96. \\ 0. \\ 36.37 \\ 0. \\ 0. \\ 21.48 \\ 28.1 \\ 8.56 \\ 11.95 \\ 16.18 \\ 11.94. \end{bmatrix} \quad v_{\pi 2} = \begin{bmatrix} 96. \\ 75.8 \\ 59.64 \\ 46.71 \\ -5. \\ 59.64 \\ 0. \\ 36.37 \\ 0. \\ 0. \\ 21.48 \\ 28.1 \\ 8.56 \\ 11.94 \\ 16.18 \\ 21.48 \end{bmatrix} \quad (5)$$

3.4 Discussion

In order to make sure that the iteration solution converges to the closed-form solution, three points need to consider:

1. The number of steps should be large enough.
2. To ensure that the times of agent passing through each state is quite enough, the initial state of the agent should be random, and episodes should also be large enough.
3. Due to the bootstrapping algorithm, the state value will converge from the terminal state to the initial state, which means the previous state value will converges slower than the later ones

4 Assignment 3: model-based algorithms

First of all, to solve this problem I first consider model-based algorithms, which means the agent knows everything about the environment including calculating the next state based on the current state and action. Dynamic programming is a common class

of model-based algorithms including value iteration and policy iteration.

4.1 Value iteration

In order to find the optimal policy, we can simply estimate the state value and define the policy as $\pi(s) = \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$. The off-policy algorithm can separate the value approximation and $\epsilon - greedy$ policy algorithm. Action policy. Additionally, value iteration is much easier to execute.

```

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  $V(\text{terminal}) = 0$ 

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

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

```

Figure 4.1: The pseudocode of value iteration

The initial state is shown in Fig 1.1. The training episode is set to be 200 times. For each episode, the game will end either the agent reach the goal or the agent has run 100 steps. Denote the parameter $\gamma = 0.8, \epsilon = 0.8$, and $decrease = 0.01$, The training procedure is recorded in the attachment DyPro-v.mp4 and the return(without discounted) is shown in Fig 4.2.

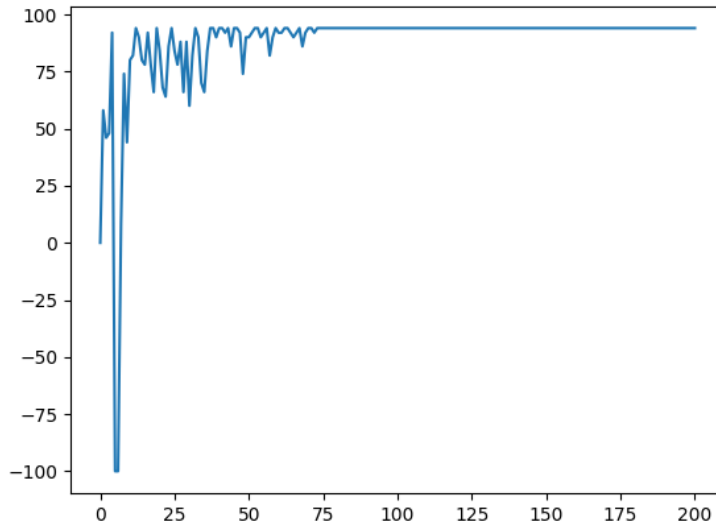


Figure 4.2: The return of each episode

After training, the state value vector is iterated and reshaped to a 4X4 matrix in order to corresponding to Fig 4.2 as follow:

$$v_s = \begin{bmatrix} 100. & 79. & 62.2 & 48.76 \\ 0. & 100. & 79. & 38.01 \\ 0. & 0. & -3.11 & 29.41 \\ -5. & -5.00 & -5.00 & 22.52 \end{bmatrix} \quad (6)$$

4.2 Policy iteration

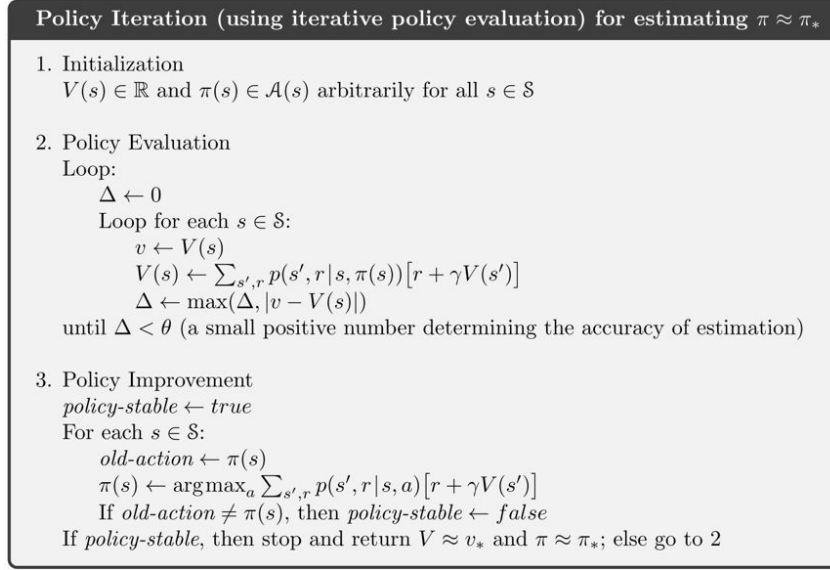


Figure 4.3: The pseudocode of policy iteration

5 model-free algorithms

Unlike the model-based algorithms, the file agent does not include any issues about the state transform matrix. To compare with the mode-based algorithm, the parameters is the same as the previous section.

5.1 Value-based methods

Q-learning and SARSA are the most famous classical value-based methods. The difference between them is that Q-learning is the off-policy while SARSA is the on-policy as shown in Fig 5.1.

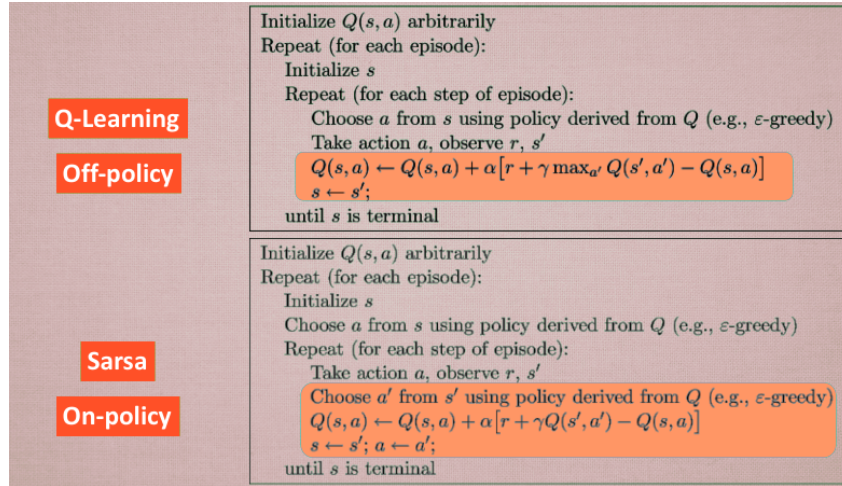


Figure 5.1: The pseudocode of two policies

The return of each episode with respect to two policies is shown in Fig. 5.2. The SARSA algorithm haven't converged in this case. On the other hand, the Q-learning algorithm behaves as good as well as the model free method in the previous section. than the SARSA algorithm.

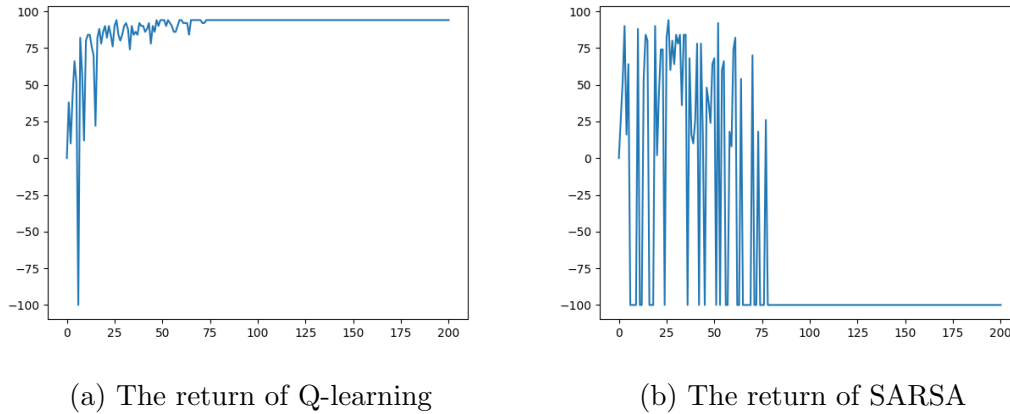


Figure 5.2: Returns of two policies in 200 episodes

The action value of Q-learning is:

$$Q_{s,A} = \begin{bmatrix} 0. & 91.41006541 & 0 & 4.75603507 \\ 0. & 79. & 53.99 & 47.86 \\ 0. & 0. & 62.2 & 34.2 \\ 0. & 27.61 & 48.76 & 0. \\ 0. & 0. & 0. & 0. \\ 56.39 & 0. & 100. & 0. \\ 0. & 0. & 0. & 0. \\ 38.01 & 18.89 & 0. & 0. \\ 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. \\ 0. & 2.68 & 0. & 21.056 \\ 29.41 & 11.8 & 13.77 & 0. \\ 0. & 0. & 0. & -0.99993 \\ 0. & 0. & -0.99993 & 1.01 \\ 6.58 & 0. & -0.99997 & 14.05 \\ 22.53 & 0. & 6.358 & 0. \end{bmatrix} \quad (7)$$

5.2 Policy-based methods

In this section, we consider methods that instead learn a parameterized policy that can select actions without consulting a value function. A value function may still be used to learn the policy parameter, but is not required for action selection. Policy-based method's advantage is that can output a continuous stochastic policy as in Fig 5.3.

```
function REINFORCE
  Initialise  $\theta$  arbitrarily
  for each episode  $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$  do
    for  $t = 1$  to  $T - 1$  do
       $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$ 
    end for
  end for
  return  $\theta$ 
end function
```

Figure 5.3: The pseudocode of policy gradient

In this case, we use a DNN to output the action as in Fig 5.4. Two linear layers, connected in the middle using Relu activation function, and finally connected to Softmax to output the probability of each action.

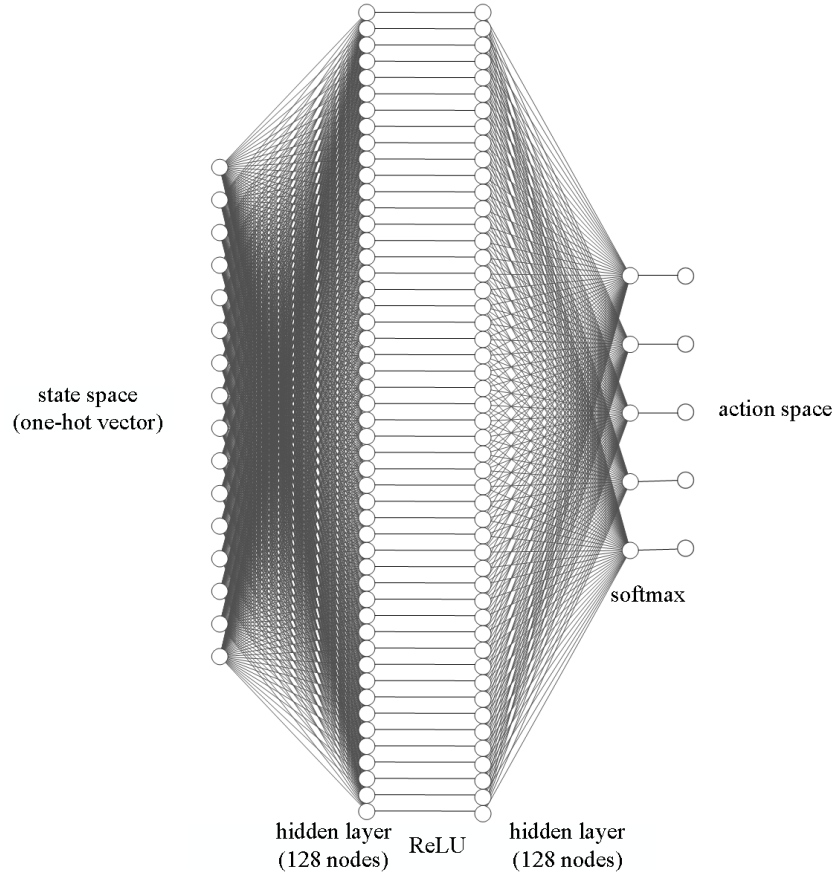
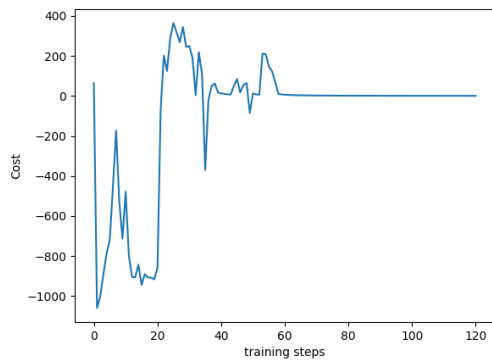
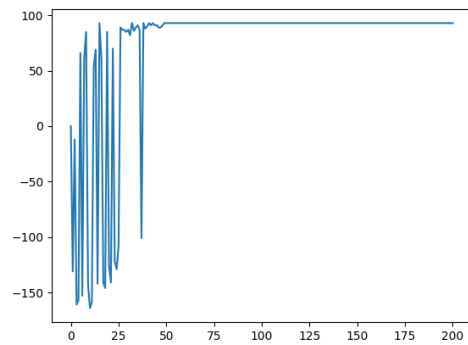


Figure 5.4: The structure of policy neural network

The training loss is $L = -q \cdot \log P(a)$, which is similar to cross entropy loss. It makes the distribution of action approximate the distribution of the action value under current policy. The training process is shown in Fig 5.5. The PG algorithm converges in 50 episodes and behaves better than all the algorithm in previous section.



(a) The training lose of policy gradient



(b) The return of policy gradient

Figure 5.5: Training process of policy gradient in 200 episodes

6 Appendix

The Dynamic programming, Sarsa, Q-learning, and Policy gradient algorithms can also be found in the attachment, which will be introduced in the following assignments. The source code will keep up to date on my Github:

https://github.com/GarrentDSTRC/RL_Project1_GridWorld

List of Figures

1.1	Grid world	1
2.1	Trajectories of the two policies	2
2.2	The discounted return	2
3.1	The new two policies	2
3.2	Matrix parameters in Bellman equation	3
4.1	The pseudocode of value iteration	7
4.2	The return of each episode	7
4.3	The pseudocode of policy iteration	8
5.1	The pseudocode of two policies	9
5.2	Returns of two policies in 200 episodes	9
5.3	The pseudocode of policy gradient	10
5.4	The structure of policy neural network	11
5.5	Training process of policy gradient in 200 episodes	12