RL Homework 2

20241202239

September 2024

Language: Python

Problem setup: **Environment:** 4*4 grid world. **Reword:** is $r_{boundary} = r_{forbidden} = -1$, and $r_{target} = 1$. **Discount rate:** is $\gamma = 0.9$

Understanding of this algorithm: There are two steps in value iteration: first is *policy update*, it aims to fin new policy that $\pi_{k+1} = arg\ max(r_\pi + \gamma P_\pi v_k)$, second is *value update*, calculate new state value from updated policy, $v_{k+1} = r_{\pi_{k+1}} + \gamma P_{\pi_{k+1}} v_k$.

Key parts of code:

```
def update_P(policy): #update martrix by policy
    P = np.zeros((16,16))
    for i in range(16):
        if i in [6,9,14]: #pass forbidden
            continue
        if policy[i] == 1:
            if ((i//4)==3)|((i+1) in [6,9,14]):
                P[i][i] = 1
            else:
                P[i][i+1] = 1
        elif policy[i] == 0:
            if (i>11)|((i+4) in [6,9,14]):
                P[i][i] = 1
            else:
                P[i][i+4] = 1
        elif policy[i] == 3:
            if ((i//4)==0)|((i-1) in [6,9,14]):
                P[i][i] = 1
```

```python
            else:
                P[i][i-1] = 1
        elif policy[i] == 2:
            if (i<4)|((i-4) in [6,9,14]):
                P[i][i] = 1
            else:
                P[i][i-4] = 1
        elif policy[i] == 4:
            P[i][i] = 1

    P = np.delete(P, [6,9,14], axis=0)
    P = np.delete(P, [6,9,14], axis=1)

    return P

if __name__ == "__main__":
    discount=0.9

    policy = np.ones(16,dtype=int)
    P = update_P(policy)

    state_value = np.zeros((16,1)) #initial values
    for i in range(61): #iteration
        env = GridWorld(env_size = (4,4),start_state = (0,0),target_state = (2,2),
                  forbidden_states = [(2,1),(1,2),(2,3)],reward_target=1,
                  reward_forbidden=-1,reward_step=0)
        state = env.reset()
        rpi = []
        for s in range(16):
            if s in [6,9,14]:
                continue
            q_table = []
            for k in range(5): #k is action
                (x, y), reward =
                    env._get_next_state_and_reward((s%4,s//4),env.action_space[k])
                q_table.append(reward+discount*(state_value[y*4+x]))
            max_value = max(q_table) #get q_table and choose firdt max valueas update
                action
```

```
        policy[s] = q_table.index(max_value)
        (x, y), reward =
            env._get_next_state_and_reward((s%4,s//4),env.action_space[policy[s]])

        rpi.append(reward) #update return
    #state_value update
    P = update_P(policy)
    state_value = np.delete(state_value, [6,9,14])
    state_value = 0.9*P@state_value + rpi
    state_value = np.insert(state_value,6,0)
    state_value = np.insert(state_value,9,0)
    state_value = np.insert(state_value,14,0)
```

Initially, set the all state value is zero, than calculate q-values for every state, choose the action with greatest q-values to update policy, than we use new policy to update state value for iteration.

Optimal policy and optimal state values: Here plot the policy:

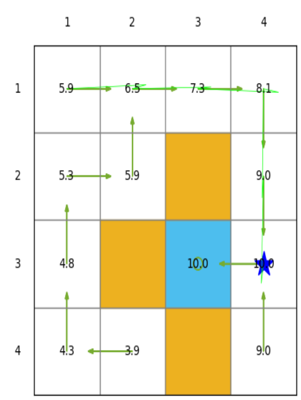

图 1: Optimal Policy

Evolvement: There are 61 iteration, first five and the last five figures are plot.

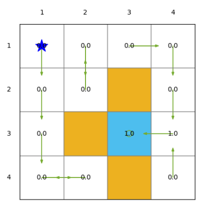
图 2: k=1

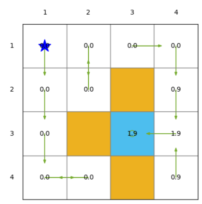
图 3: k=2

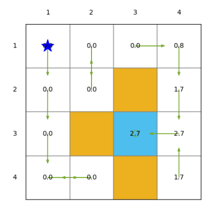
图 4: k=3

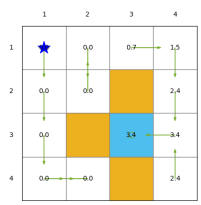
图 5: k=4

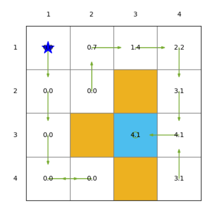
图 6: k=5

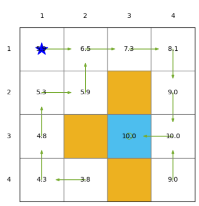
图 7: k=57

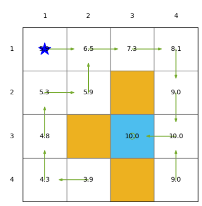
图 8: k=58

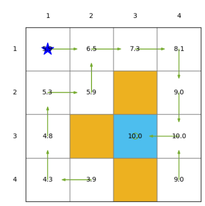
图 9: k=59

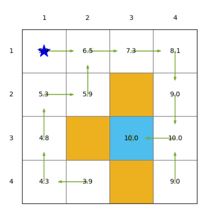
图 10: k=60

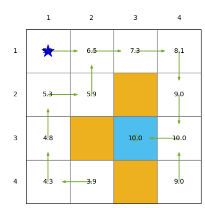
图 11: k=61

Observation The iteration firstly update the state value form the target, and then nearby state. After policy is fixed, state value are still update like use iterative solution to find state value for a policy.