Report for Gridworld

516021910212 Liuyuxi

March 2019

1 Introduction

Use dynamic programming to carry out reinforcement learning. Judge the value function of a strategy based on known models. And find the optimal strategy and optimal value function by policy evaluation and policy iteration. The DP methods have been tested in a small gridworld.

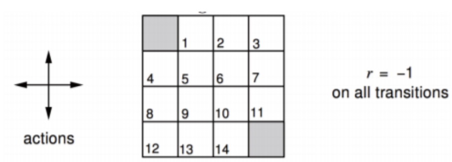

Figure 1: The gridworld

The agent is in an 4×4 gridworld, trying to get to the top left or bottom right corner as shown in 1. Each gird in the gridworld represents a certain state. S_t is the state at grid t. And the state space can be denoted as $S = \{s_t | t \in 0, ..., 15\}$. S_0 and S_{15} are terminal states, where $S_1$to $S_1$4are non-terminal states and can move one grid to north,east,south and west. Hence the action space is $A = \{N, E, S, W\}$. And the actions leading out of the grid leave state unchanged. Each movement get a reward of -1 until the terminal state is reached. Both policy evaluation and policy iteration have been used in finding the shortest way to the terminal state randomly given an initial non-terminal state.

2 Algorithms

This is a model in the Reinforcement Learning literature. In this particular case:

- State space S: GridWorld has 4×4 distinct states, The start state is S_0 and the end state is S_{15}

- Actions A: The agent can choose from up to 4 actions to move around.
- Transition probability P: The position of any attempt to leave the square world will not change, and the rest will be 100% transferred to the position pointed by the action
- Rewards R: The agent receives -1 reward when it is in the non-terminal states and receives 0 reward when it is the terminal states.
- Attenuation coefficient γ :1
- Current strategy : The individual adopts a random action strategy and has an equal probability to move in any possible direction in any non-terminating state, $\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 1/4$

In other words, this is a deterministic, finite Markov Decision Process (MDP) and as always the goal is to find an agent policy that maximizes the future discounted reward.

- The Policy Evaluation (one sweep) button turns the Bellman equation into a synchronous update and performs a single step of Value Function estimation. Intuitively, this update is diffusing the raw Rewards at each state to other nearby states through 1. the the dynamics of the environment and 2. the current policy.

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')\right]$$

- The Policy Update button iterates over all states and updates the policy at each state to take the action that leads to the state with the best Value (integrating over the next state distribution of the environment for each action).

  Here, we are computing the action value function at each state $Q(s,a)$, which measures how much expected reward we would get by taking the action a in the state s. The function has the form:

$$Q^{\pi}(s,a) = E_{\pi}\{r_t + \gamma V^{\pi}(s_{t+1}) \mid s_t = s, a_t = a\} = \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')\right]$$

  In our Gridworld example, we are looping over all states and evaluating the Q function for each of the (up to) four possible actions. Then we update the policy to take the argmaxy actions at each state. That is,

$$\pi'(s) = \arg\max_{a} Q(s,a)$$

3 Setup

3.1 Policy Evaluation

```
#declare the state
states=[i for i in range(16)]
#declare the value and the initialization is 0
values=[0 for _ in range(16) ]
actions=["N","E","S","W"]
#move north then the current state -4
ds_actions={"N":-4,"E":1,"S":4,"W":-1}
gramma=1.00#Attenuation coefficient
#determin the next state
def nextState(s,a):
    next_state=s
    #boundary
    if(s%4==0 and a=="W")or (s<4 and a=="N") \
    or ((s+1)%4==0 and a=="E")or (s>11 and a=="S"):
        pass
    else:
        ds=ds_actions[a]
        next_state=s+ds
    return next_state

#get the reward when leaving the state
def rewardOf(s):
    return 0 if s in [0,15] else -1

#check whether the terminatestate
def isTerminateState(s):
    return s in [0,15]

#get all the next states
def getAllState(s):
    next=[]
    if isTerminateState(s):
        return next
    for a in actions:
        next_state=nextState(s,a)
        next.append(next_state)#include not move
    return next

#update the current value
def updateValue(s):
    AllState=getAllState(s)
    newValue=0
    num=4
    reward=rewardOf(s)#leaving reward
    for next_state in AllState:
```

```
            newValue+=1.00/num*(reward+\
            gramma*values[next_state])
        return newValue


def printValue(v):
    for i in range(16):
        print('{0:>6.2f}'.format(v[i]), end=" ")
        if (i + 1) % 4 == 0:
            print("")
    print()


def performOneIteration():
    newValues=[0 for _ in range(16)]
    for s in states:
        newValues[s]=updateValue(s)
    global values
    values=newValues
    printValue(values)

def main():
    max_iterate_times=160
    cur_iterate_times=0
    while cur_iterate_times<=max_iterate_times:
        print("Iterate No.{0}".format\
        (cur_iterate_times))
        performOneIteration()
        cur_iterate_times+=1
    printValue(values)

if __name__=='__main__':
    main()
```

3.2 Policy Iteration

```
# the state
states = [i for i in range(16)]
# the value
values = [0 for _ in range(16)]
actions = ["N", "E", "S", "W"]
# move north then the current state -4
ds_actions = {"N": -4, "E": 1, "S": 4, "W": -1}
gramma = 1.00  # Attenuation coefficient
```

```
a = []
for i in range(16):
    a.append(actions)
#determin the next state
def nextState(s,a):
    next_state=s
    #boundary
    if(s%4==0 and a=="W")or (s<4 and a=="N") or \
    ((s+1)%4==0 and a=="E")or (s>11 and a=="S"):
        pass
    else:
        ds=ds_actions[a]
        next_state=s+ds
    return next_state

#get the reward when leaving the state
def rewardOf(s):
    return 0 if s in [0,15] else -1



#check whether the terminatestate
def isTerminateState(s):
    return s in [0,15]


def printValue(v):
    for i in range(16):
        print('{0:>6.2f}'.format(v[i]),end = " ")
        if(i+1)%4==0:
            print("")
    print()


#strategy
def allowedActions(s):
    allow=[]
    candidate=[]
    if isTerminateState(s):
        return allow
    for a in actions:
        next_state=nextState(s,a)
        candidate.append(values[next_state])
    b=max(candidate)
    for a in actions:
        next_state=nextState(s,a)
```

```
            if(values[next_state]==b):
                allow.append(a)
        return allow

#get the possible state on allowed actions
def allowedStates(s):
    allowStates=[]
    if isTerminateState(s):
        return allowStates
    for a in allowedActions(s):
        next_state=nextState(s,a)
        allowStates.append(next_state)
    return allowStates

def allowedUpdateValue(s):
    next_state=allowedStates(s)
    newValue=0
    num=len(next_state)
    reward=rewardOf(s)
    for state in next_state:
        newValue+=1.00/num*(reward+\
        gramma*values[state])
    return newValue

#iteration and update the strategy
def allowedPerformOneIteration():
    newValues=[0 for _ in range(16)]
    newa=[]
    for s  in states:
        allowedActions(s)
        newa.append(allowedActions(s))
    global a
    a=newa
    #the strategy is updated
    for s  in states:
        newValues[s]=allowedUpdateValue(s)
    global values
    values=newValues
    printValue(values)

def main():
    max_iterate_times=10
    cur_iterate_times=0
    while cur_iterate_times<=max_iterate_times:
        print("Iterate No.{0}".format\
        (cur_iterate_times))
```

```
            allowedPerformOneIteration()
            #performOneIteration()
            cur_iterate_times+=1
        printValue(values)
        for i in range(16):
            print(allowedActions(i))


if __name__=='__main__':
    main()
```

4 Result

4.1 Policy Evalution

With the same action stratety, the algorithm will converge around 153 times. We set the maximum number of iterations to 160. The resulting value function is shown in Table1 and Figure2.

Table 1: Policy Evaluation

Iterate No.152			
0.00	-14.00	-20.00	-21.99
-14.00	-18.00	-20.00	-20.00
-20.00	-20.00	-18.00	-14.00
-21.99	-20.00	-14.00	0.00
Iterate No.153			
0.00	-14.00	-20.00	-22.00
-14.00	-18.00	-20.00	-20.00
-20.00	-20.00	-18.00	-14.00
-22.00	-20.00	-14.00	0.00
Iterate No.154			
0.00	-14.00	-20.00	-22.00
-14.00	-18.00	-20.00	-20.00
-20.00	-20.00	-18.00	-14.00
-22.00	-20.00	-14.00	0.00

4.2 Policy Iteration

the algorithm will converge around 4 times. We set the maximum number of iterations to 10. The result is shown in Table2 and Figure3.

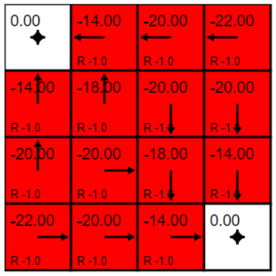

Figure 2: Policy evaluation

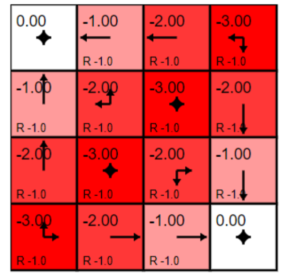

Figure 3: Policy Iteration

Table 2: Policy iteration

Iterate No.2			
0.00	-1.00	-2.00	-2.00
-1.00	-2.00	-2.00	-2.00
-2.00	-2.00	-2.00	-1.00
-2.00	-2.00	-1.00	0.00
Iterate No.3			
0.00	-1.00	-2.00	-3.00
-1.00	-2.00	-3.00	-2.00
-2.00	-3.00	-2.00	-1.00
-3.00	-2.00	-1.00	0.00
Iterate No.4			
0.00	-1.00	-2.00	-3.00
-1.00	-2.00	-3.00	-2.00
-2.00	-3.00	-2.00	-1.00
-3.00	-2.00	-1.00	0.00

5 Conclusion

Thanks to this project that I have a deep understanding of reinforcement learning. I also meet with some obstacles during the period. At the beginning, I choose the environment gridworld. However, after some investigation I turn to establish the environment myself. Thanks for the guidance of the teacher and our teaching assistants.