Introduction to Reinforcement Learning

Syed Mustafa
Habib University
sm06554@st.habib.edu.pk
Submitted to Shahid Shaikh

March 27, 2023

1 Planning

Topics tested in this assignment:

1. Optimal Policy and Optimal function.
2. Chapter 3, 4 and 5.

Explanation of GridWorld

This section is to explain the MDP for question 1 and 2.
This is a grid representation of a finite Markov Decision Process (MDP). Each cell is a state of the MDP. Four actions are possible at a given state, i.e. **north, south, east, west**. Actions that would cause the agent to get of the grid leave its location unchanged.

Question 1

The optimal value function v∗ for the GridWorld is generated by the code GridWorld 3 5.py which is available on Canvas. Write a program which takes the value function generated by the above code as input and generates the corresponding optimal policy.

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

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

Figure 1: Policy Iteration Psuedocode

The given code has already implemented the policy evaluation. Now to obtain a optimal policy we must implement policy improvement.
The code for this code is provided below:

Code for Question 1

Listing 1: The above piece of code remained unchanged

```
import numpy as np
WORLD_SIZE = 5
A_POS = [0, 1]
A_PRIME_POS = [4, 1]
B_POS = [0, 3]
B_PRIME_POS = [2, 3]
DISCOUNT = 0.9
# left, up, right, down
ACTIONS = [np.array([0, -1]),
           np.array([-1, 0]),
           np.array([0, 1]),
           np.array([1, 0])]

def step(state, action):
    if state == A_POS:
        return A_PRIME_POS, 10
    if state == B_POS:
        return B_PRIME_POS, 5



    next_state = (np.array(state) + action).tolist()
    # print(f'The next_state = (np.array(state) + action).tolist() outputs= \n
        {next_state} ')
    x, y = next_state
    if x < 0 or x >= WORLD_SIZE or y < 0 or y >= WORLD_SIZE:
        reward = -1.0
        next_state = state
    else:
        reward = 0
    return next_state, reward
```

Listing 2: This was part of the policy improvement code. This function would return a new pocily.

```
def optimal_policy(value):
    policy = np.zeros((WORLD_SIZE, WORLD_SIZE), dtype=object)
    for i in range(WORLD_SIZE):
        for j in range(WORLD_SIZE):
            values = []
            for action_idx, action in enumerate(ACTIONS):
                (next_i, next_j), reward = step([i, j], action)
                values.append(reward + DISCOUNT * value[next_i, next_j])

            best_actions = np.argwhere(values == np.max(values)).flatten().tolist()
            policy[i, j] = np.array(best_actions)
    return policy
```

Listing 3: Changes to given code

```
def figure_3_5():
    value = np.zeros((WORLD_SIZE, WORLD_SIZE))
    policy = np.random.randint(0,4,(5,5,1))
```

```
    epoch = 0
    while True:
        it = 0
        # it = 0
        while True:
            # keep iteration until convergence
            new_value = np.zeros_like(value)
            for i in range(WORLD_SIZE): #nested loop to loop over each state
                for j in range(WORLD_SIZE):
                    values = [] #values is different from value.
                    for x in policy[i, j]:
                        action = ACTIONS[x]
                        # print('The Action = ', action)

                        (next_i, next_j), reward = step([i, j], action)
                        # value iteration
                        values.append(reward + DISCOUNT * value[next_i, next_j]) #V(s)
                            = Sum over all s', r: p(Rt+1 + DISCOUNT * value[next_i,
                            next_j])
                    new_value[i, j] = np.max(values)

            if np.sum(np.abs(new_value - value)) < 1e-2:
                np.set_printoptions(precision=2)
                print(f'The value function at epoch {epoch} coonverged at iteration
                    {it} is \n {value} \n With the policy: \n {policy}')
                print()
                break
            value = new_value
            it += 1



        stable = True
        new_policy = optimal_policy(value)
        for i in range(len(new_policy)):
            for j in range(len(new_policy[i])):
                if not (new_policy[i, j].tolist() == policy[i,j].tolist()):
                    stable = False
        if stable == True:
            optimalPolicy = policy
            optimal_value = value
            break

        # print(f'And the new_policy is in epoct {epoch}= \n{new_policy}')

        epoch += 1


        policy = new_policy
    return optimalPolicy, optimal_value
    # print(f'And the optimal policy is = \n{policy}')
```

Listing 4: Code to make policy easier to view

```
policy, value = figure_3_5()
print(value)
```

```python
left, up, right, down = 0,1,2,3
arrow_dic = dict([(0 , "left"), (1 ,"up"), (2 , "right"), (3, "down")])

Arrow = np.zeros_like(policy, dtype='object')
# print(Arrow)
for i in range(len(policy)):
    for j in range(len(policy[i])):
        temp = []
        for index in range(len(policy[i,j])):
            x= arrow_dic[policy[i,j][index]]
            temp.append(x)
        Arrow[i,j] = temp
# print(Arrow.tolist())

for i in range(len(policy)):
    for j in range(len(policy[i])):
        # for index in range(len(policy[i,j])):
        print(Arrow[i, j], end = ' , ')
    print()

#  (0 = left, 1 = up, 2 = right, 3 = down)
```

Output

```python
policy, value = figure_3_5()
print(value)
left, up, right, down = 0,1,2,3
arrow_dic = dict([(0 , "left"), (1 ,"up"), (2 , "right"), (3, "down")])

Arrow = np.zeros_like(policy, dtype='object')
# print(Arrow)
for i in range(len(policy)):
    for j in range(len(policy[i])):
        temp = []
        for index in range(len(policy[i,j])):
            x= arrow_dic[policy[i,j][index]]
            temp.append(x)
        Arrow[i,j] = temp
# print(Arrow.tolist())

for i in range(len(policy)):
    for j in range(len(policy[i])):
        # for index in range(len(policy[i,j])):
        print(Arrow[i, j], end = ' , ')
    print()

#  (0 = left, 1 = up, 2 = right, 3 = down)
```

```
[[21.98 24.42 21.98 19.42 17.48]
 [19.78 21.98 19.78 17.8  16.02]
 [17.8  19.78 17.8  16.02 14.42]
 [16.02 17.8  16.02 14.42 12.98]
 [14.42 16.02 14.42 12.98 11.68]]
['right'] , ['left', 'up', 'right', 'down'] , ['left'] , ['left', 'up', 'right', 'down'] , ['left'] ,
['up', 'right'] , ['up'] , ['left', 'up'] , ['left'] , ['left'] ,
['up', 'right'] , ['up'] , ['left', 'up'] , ['left', 'up'] , ['left', 'up'] ,
['up', 'right'] , ['up'] , ['left', 'up'] , ['left', 'up'] , ['left', 'up'] ,
['up', 'right'] , ['up'] , ['left', 'up'] , ['left', 'up'] , ['left', 'up'] ,
```

Figure 2: Output of question 1

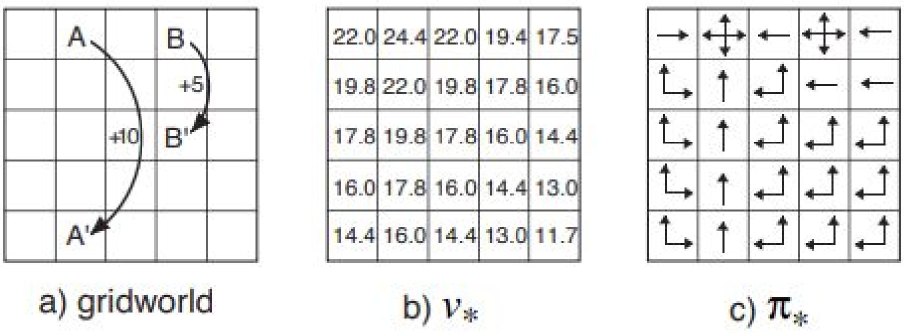

Figure 3: The output matches the given answer in book

Question 2

Starting from the code GridWorld 3 2.py, which is available on Canvas, implement the complete value iteration algorithm to generate the optimal value function $v*$ and an optimal policy $\pi*$.

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

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

until $\Delta < \theta$ (a small positive number)

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

Figure 4.5: Value iteration.

Figure 4: Policy Iteration Psuedocode

Code for Question 2

Listing 5: Preliminary

```
#######################################################################
# Copyright (C)                                                       #
# 2016-2018 Shangtong Zhang(zhangshangtong.cpp@gmail.com)             #
# 2016 Kenta Shimada(hyperkentakun@gmail.com)                         #
# Permission given to modify the code as long as you keep this        #
# declaration at the top                                              #
#######################################################################

import numpy as np

WORLD_SIZE = 5
A_POS = [0, 1]
A_PRIME_POS = [4, 1]
B_POS = [0, 3]
B_PRIME_POS = [2, 3]
DISCOUNT = 0.9

# left, up, right, down
ACTIONS = [np.array([0, -1]),
          np.array([-1, 0]),
          np.array([0, 1]),
          np.array([1, 0])]
ACTION_PROB = 0.25

def step(state, action):
    if state == A_POS:
```

```
        return A_PRIME_POS, 10
    if state == B_POS:
        return B_PRIME_POS, 5

    next_state = (np.array(state) + action).tolist()
    x, y = next_state
    if x < 0 or x >= WORLD_SIZE or y < 0 or y >= WORLD_SIZE:
        reward = -1.0
        next_state = state
    else:
        reward = 0
    return next_state, reward
```

Listing 6: Updated GridWorld3_2() code

```
def figure_3_2():
    value = np.zeros((WORLD_SIZE, WORLD_SIZE))
    it = 0
    while True:
        # keep iteration until convergence
        new_value = np.zeros_like(value)
        for i in range(WORLD_SIZE):
            for j in range(WORLD_SIZE):
                temp = []
                for action in ACTIONS:

                    (next_i, next_j), reward = step([i, j], action)
                    temp.append(ACTION_PROB * (reward + DISCOUNT * value[next_i,
                        next_j]))
                    # bellman equation
                new_value[i, j] = max(temp)
        if np.sum(np.abs(value - new_value)) < 1e-2:
            break
        value = new_value
        it += 1
        # input("Press Enter to continue...")
        np.set_printoptions(precision=2)
        print(value)
        print()
    print("Converges in {} iterations".format(it))


    ## Greedy Policy ##

    policy = greedy_policy(value)
    print("The Policy is",policy, sep = ' = n')

    return policy, value
if __name__ == '__main__':
    policy, value = figure_3_2()
    print(value)
    left, up, right, down = 0,1,2,3
    arrow_dic = dict([(0 , "left"), (1 ,"up"), (2 , "right"), (3, "down")])

    Arrow = np.zeros_like(policy, dtype='object')
    # print(Arrow)
    for i in range(len(policy)):
```

```
        for j in range(len(policy[i])):
            temp = []
            for index in range(len(policy[i,j])):
                x= arrow_dic[policy[i,j][index]]
                temp.append(x)
            Arrow[i,j] = temp
    # print(Arrow.tolist())

    for i in range(len(policy)):
        for j in range(len(policy[i])):
            # for index in range(len(policy[i,j])):
            print(Arrow[i, j], end = ' , ')
        print()

#  (0 = left, 1 = up, 2 = right, 3 = down)
```

Listing 7: Greedy Policy

```
def greedy_policy(value):
    policy = np.zeros((WORLD_SIZE, WORLD_SIZE), dtype=object)
    for i in range(WORLD_SIZE):
        for j in range(WORLD_SIZE):
            values = []
            for action_idx, action in enumerate(ACTIONS):
                (next_i, next_j), reward = step([i, j], action)
                values.append(reward + DISCOUNT * value[next_i, next_j])

            best_actions = np.argwhere(values == np.max(values)).flatten().tolist()
            policy[i, j] = np.array(best_actions)
    return policy
```

Question 3

This question was completed with Omema Rizvi (or06360)

Listing 8: Question 3

```
# Right now conser
import numpy as np
from statistics import mean


WORLD_SIZE = 5
A_POS = [0, 1]
A_PRIME_POS = [4, 1]
B_POS = [0, 3]
B_PRIME_POS = [2, 3]
DISCOUNT = 0.9
TERMINAL = [WORLD_SIZE-1, WORLD_SIZE-1]
# left, up, right, down
ACTIONS = [np.array([0, -1]),
          np.array([-1, 0]),
          np.array([0, 1]),
          np.array([1, 0])]





def step(state, action):
    terminal = False
    if state == A_POS:
        return A_PRIME_POS, 10
    if state == B_POS:
        return B_PRIME_POS, 5
    if state == TERMINAL:
        print(f"the terminal state is reached")
        return None, 0, #next_state == None --> state <-- Terminal

    next_state = (np.array(state) + action).tolist()
    x, y = next_state
    if x < 0 or x >= WORLD_SIZE or y < 0 or y >= WORLD_SIZE:
        reward = -1.0
        next_state = state
    else:
        reward = 0
    return next_state, reward


def greedy_policy(value):
    '''
    Reused from Question 2
    '''

    policy = np.zeros((WORLD_SIZE, WORLD_SIZE), dtype=object)
    for i in range(WORLD_SIZE):
        for j in range(WORLD_SIZE):
            values = []
            for action_idx, action in enumerate(ACTIONS):
                if i == WORLD_SIZE -1 and j == WORLD_SIZE -1:
```

```python
                values.append(0)
                continue
            (next_i, next_j), reward = step([i, j], action)
            values.append(reward + DISCOUNT * value[next_i, next_j])

        best_actions = np.argwhere(values == np.max(values)).flatten().tolist()
        policy[i, j] = np.array(best_actions)
    return policy


def moving_avg(v, n, G):
    '''
    Args:
    G: Return Value
    v: value of of state s belonging to S
    n: number of times s has been visited before, in previous episodes
    Return:

    '''
    v = n/(n+ 1)* (v+1/n * G)
    return v


def generates_episode(Policy):
    '''
    Args:
    Policy: Policy pi
    Return:
    EP: Episode list: [()]
    '''

    too_long = False
    random = np.random.randint(25)
    i = random//5
    j = random%5
    curr_state = [i,j] # State initialization
    EP = list()
    terminal = False
    while not terminal:
        action_index = np.random.choice(Policy[i, j])
        action = ACTIONS[action_index]
        next_state, reward = step(curr_state, action)
        EP.append((curr_state, action.tolist(), reward))
        if next_state == None:
            terminal = True
            break
        curr_state = next_state
        i,j = curr_state[0], curr_state[1]
        if len(EP) >= 100:
            too_long = True
            print('The Episode was stuck in a loop')
            break
    if too_long == True:
        EP = generates_episode(Policy)
        # print(f'(curr_state, action, reward) = {(curr_state, action, reward)}')

    return EP
```

```python
def MC_first_Visit(policy):
    '''
    Args:
    N: number of states

    '''
    V = np.zeros((WORLD_SIZE, WORLD_SIZE))
    Returns = value = np.zeros((WORLD_SIZE* WORLD_SIZE), dtype = 'object')
    for i in range(len(Returns)):
        Returns[i] = list()
    epsilon = 1e-2
    ep = 0
    while True:
        ep += 1
        episode = generates_episode(policy)
        print(f'The episode {ep} is: \n {episode}')
        old_value = V.copy() # For episode ep - 1
        # print(f'The oldvalue assighnment \n {old_value}')
        G = 0
        reverse_ep = episode.copy()
        reverse_ep.reverse()
        visited = reverse_ep.copy()

        for curr_state, action, reward in reverse_ep:

            visited.remove((curr_state, action, reward))
            G = DISCOUNT * G + reward
            # print(f'The state is = {curr_state}')
            # print(f'The visited is = \n ={visited}\n ')
            # print(f'The value of G = {G}')
            if not ((curr_state, action, reward) in visited):
                Returns[curr_state[0]*curr_state[1]].append(G)
                V[curr_state[0],curr_state[1]] =
                    mean(Returns[curr_state[0]*curr_state[1]])
                # print('Value update = ',V[curr_state], 'The updated Value = ',V, sep
                    = '\n')
            else:
                print('Already Visited')
        # print(f"The updated value in episode {ep} is = \n {V}")

        if np.sum(np.abs(V - old_value)) < 1e-2:
            print('__________________________________________________')
            # print(old_value)
            print('__________________________________________________')
            print(f'The value function converges at episode {ep} \n at the value is =
                \n{V}')
            print('__________________________________________________')
            print('__________________________________________________')
            break

    return V


def policy_iteration_MC(start_policy):
    policy = start_policy
    epoch = 0
    while True:
        value = MC_first_Visit(policy)
```

```
        epoch += 1

        stable = True
        new_policy = greedy_policy(value)
        print(f' The new policy at epoch {epoch} \n {new_policy}\n
            --------------------------')

        for i in range(len(new_policy)):
            for j in range(len(new_policy[i])):


                if not (new_policy[i, j].tolist() == policy[i,j].tolist()):
                    stable = False
        if stable == True:
            optimalPolicy = policy
            optimal_value = value
            break

        policy = new_policy


        return policy

policy = [[np.array([0,1,2,3]), np.array([0,1,2,3]),
    np.array([0,1,2,3]),np.array([0,1,2,3]),np.array([0,1,2,3])],
          [np.array([0,1,2,3]),np.array([0,1,2,3]),np.array([0,1,2,3]),np.array([0,1,2,3]),np.array([
          [np.array([0,1,2,3]),np.array([0,1,2,3]),np.array([0,1,2,3]),np.array([0,1,2,3]),np.array([
          [np.array([0,1,2,3]),np.array([0,1,2,3]),np.array([0,1,2,3]),np.array([0,1,2,3]),np.array([
          [np.array([0,1,2,3]),np.array([0,1,2,3]),np.array([0,1,2,3]),np.array([0,1,2,3]),np.array([
policy = np.array(policy)
# episode = generates_episode(policy)
# left, up, right, down
ACTIONS = [np.array([0, -1]),
          np.array([-1, 0]),
          np.array([0, 1]),
          np.array([1, 0])]
# MC_first_Visit(policy)
policy = policy_iteration_MC(policy)

left, up, right, down = 0,1,2,3
arrow_dic = dict([(0 , "left"), (1 ,"up"), (2 , "right"), (3, "down")])

Arrow = np.zeros_like(policy, dtype='object')
# print(Arrow)
for i in range(len(policy)):
    for j in range(len(policy[i])):
        temp = []
        for index in range(len(policy[i,j])):
            x= arrow_dic[policy[i,j][index]]
            temp.append(x)
        Arrow[i,j] = temp
# print(Arrow.tolist())

for i in range(len(policy)):
    for j in range(len(policy[i])):
        # for index in range(len(policy[i,j])):
        print(Arrow[i, j], end = ' , ')
    print()
```

Explanation will be given in the viva.

Google Colab Notebook

Please fnd the link to the google colab notebook:
https://colab.research.google.com/drive/1SUZGc8QwLwSf7NQFFGH8O0Q4ZLBHoh-6?usp=sharing