Introduction to Reinforcement Learning

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

March 26, 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 $s \in \mathcal{S}$:
      $a \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
      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:

```
    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 {
    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 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().tolis
            policy[i, j] = np.array(best_actions)
    return policy

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])
                    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 iterat
                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 epoch {epoch}= \n{new_policy}')

        epoch += 1


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


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)
```

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 $\pi*$.

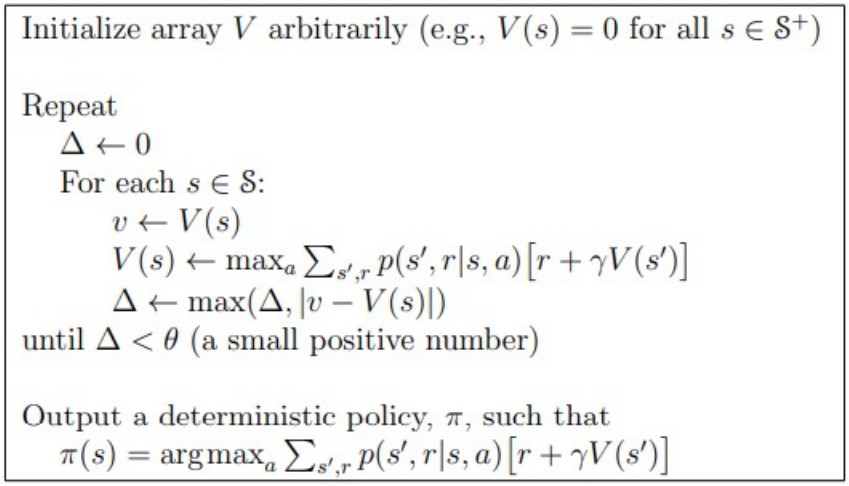

Figure 4.5: Value iteration.

Figure 2: Policy Iteration Psuedocode

```
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 {
    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 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().tolis
            policy[i, j] = np.array(best_actions)
    return policy

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])
                    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 iterat
                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}')


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)
```

Google Colab Notebook

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

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove the template text from your paper may result in your paper not being published.