1-a

December 7, 2023

[52]:
```python
import numpy as np
import math
```

[53]:
```python
flappyworld1 = np.array(
    [
        ["terminal","terminal","terminal","terminal","terminal"],
        ["unshaded","unshaded","terminal","unshaded","unshaded"],
        ["unshaded","unshaded","terminal","unshaded","unshaded"],
        ["unshaded","unshaded","unshaded","unshaded","goal"],
        ["unshaded","terminal","unshaded","unshaded","unshaded"],
        ["unshaded","terminal","unshaded","unshaded","unshaded"],
        ["terminal","terminal","terminal","terminal","terminal"]
    ]
)
```

[54]:
```python
N_ROW, N_COL = flappyworld1.shape
```

[55]:
```python
gamma = 0.9
r_g = 5
r_r = -5

s0 = 2
actions = { "rightup" : np.array([-1,1]), "rightdown" : np.array([1,1]) }
```

[56]:
```python
def state_to_coordinates( state ):
    state -= 1
    column = int( state / N_ROW)
    row = state - column * N_ROW
    return row, column
```

[57]:
```python
def coordinates_to_state ( row, column ):
    return (column * N_ROW + row) + 1
```

[58]:
```python
def dynamics ( state_1, action_1 ):
    row_1, col_1 = state_to_coordinates( state_1 )
    label = flappyworld1[row_1][col_1]
```

```
    if label in [ "terminal", "goal" ]:
        return state_1

    row_2 = row_1 + actions[action_1][0]
    col_2 = col_1 + actions[action_1][1]

    if ( row_2 > N_ROW-1 ) or ( row_2 < 0 ) or ( col_2 > N_COL-1 ) or ( col_2 <␣
↪0 ):
        return coordinates_to_state( row_1+1, col_1 )
    else:
        return coordinates_to_state( row_2, col_2 )
```

[59]:
```
dynamics ( 32, "rightdown" )
```

[59]: 32

[60]:
```
def helper(root, arr, ans):
    arr.append(root)
    left =  dynamics ( root, "rightup" )
    right =  dynamics ( root, "rightdown" )

    if root == left or root == right:
        # This will be only true when the node is leaf node and hence we will␣
↪update our ans array by inserting array arr which have one unique path from␣
↪root to leaf
        ans.append(arr.copy())
        del arr[-1]
        # after that we will return since we don't want to check after leaf node
        return

    # recursively going left and right until we find the leaf and updating the␣
↪arr and ans array simultaneously
    if left != right:
        helper(left, arr, ans)
        helper(right, arr, ans)
    else:
        helper(left, arr, ans)
    del arr[-1]


def Paths(root):
    ans = [] # creating answer in which each element is a array having one␣
↪unique path from root to leaf
    arr = [] # arr is a array which will have one unique path from root to leaf␣
↪at a time.arr will be updated recursively
    helper(root, arr, ans) # after helper function call our ans array updated␣
↪with paths so we will return ans array
```

```python
    return ans

def printArray(paths, reward):
    optimal_reward = 35*-5
    len_optimal = 35
    visited_state = set()
    goal_optimal_reward = 35*-5
    len_goal_optimal = 35
    for path in paths:
        state_list = []
        label_list = []
        reward_list = []

        for state in path:
            visited_state.add(state)
            state_list.append(str(state))
            row,col = state_to_coordinates( state )
            label = flappyworld1[row][col]
            label_list.append(label)
            reward_list.append(str(reward[label]))

        print("path : ", end="")
        print(" ---> ".join(state_list), end="")
        print(" : length of shortest path : " + str(len(state_list)))
        print("label : ", end="")
        print(" ---> ".join(label_list))
        print("reward : ", end="")
        print(" + ".join(reward_list), end="")
        total_reward = sum( list(map(int, reward_list)) )
        print( " = " + str(total_reward) + " = total reward")
        if total_reward > optimal_reward:
            optimal_reward = total_reward
            if len_optimal > len(state_list):
                len_optimal = len(state_list)
        lastState = int( state_list[-1] )
        r,c = state_to_coordinates(lastState)
        if flappyworld1[r][c] == "goal" and total_reward > goal_optimal_reward:
            goal_optimal_reward = total_reward
            if len_goal_optimal > len(state_list):
                len_goal_optimal = len(state_list)

        print()
    print("="*60)
    print(str(len(visited_state)) + " traversed states : ", end=" ")
    visited_state = [ str(state) for state in visited_state]
    print(visited_state)
    print("="*60)
```

```
[61]: def v_function_policy_evaluation( policy, states_space, value, reward ):
    value_new = [0] * len(value)
    for i, state in enumerate(states_space):
        next_state = dynamics ( 32, "rightdown" )
        value_new[i] = reward[state] + gamma * value[next_state]
    return value_new
```

0.0.1 Question 1.(a)

- PART1 : briefly explain what the optimal policy would be in Flappy World 1.

- PART2 : is the optimal policy unique ?
- PART3 : does the optimal policy depend on the value of the discount factor $\gamma \in [0, 1]$?

- PART4 : Explain your answer.

0.0.2 Hints

- What is the optimal policy? (the one that has the highest discounted sum of reward.)
- What is the difference between positive vs negative reward values of r_s ?
- Unique or not, list conditions / why you think it is unique.

0.0.3 Overview : WITHOUT considering the discount factor γ

Let's represent policies as paths of a tree. We first get an overview by traverseing all possible paths from the starting state to a terminal state (either RED or GREEN) . - root : starting state 2 - leaf nodes : terminal nodes

```
r_s  = -4
print("="*30+"r_s = " + str(r_s)+"="*30)
reward = { "terminal" : r_r, "goal" : r_g, "unshaded" : r_s }
printArray(Paths(2),reward)
```

```
==============================r_s = -4==============================
path : 2 ---> 8 : length of shortest path : 2
label : unshaded ---> terminal
reward : -4 + -5 = -9 = total reward

path : 2 ---> 10 ---> 16 : length of shortest path : 3
label : unshaded ---> unshaded ---> terminal
reward : -4 + -4 + -5 = -13 = total reward

path : 2 ---> 10 ---> 18 ---> 24 ---> 30 ---> 31 ---> 32 : length of shortest path : 7
label : unshaded ---> unshaded ---> unshaded ---> unshaded ---> unshaded ---> unshaded ---> goa
reward : -4 + -4 + -4 + -4 + -4 + -4 + 5 = -19 = total reward

path : 2 ---> 10 ---> 18 ---> 24 ---> 32 : length of shortest path : 5
label : unshaded ---> unshaded ---> unshaded ---> unshaded ---> goal
reward : -4 + -4 + -4 + -4 + 5 = -11 = total reward
```

```
path : 2 ---> 10 ---> 18 ---> 26 ---> 32 : length of shortest path : 5
label : unshaded ---> unshaded ---> unshaded ---> unshaded ---> goal
reward : -4 + -4 + -4 + -4 + 5 = -11 = total reward

path : 2 ---> 10 ---> 18 ---> 26 ---> 34 ---> 35 : length of shortest path : 6
label : unshaded ---> unshaded ---> unshaded ---> unshaded ---> unshaded ---> terminal
reward : -4 + -4 + -4 + -4 + -4 + -5 = -25 = total reward

============================================================
12 traversed states :  ['32', '2', '34', '35', '8', '10', '16', '18', '24', '26', '30', '31']
============================================================
```

Observation from the route traversal

- there are 12 explored states:
  - staring state : 2
  - RED terminal states : 8, 16, 35
  - GREEN terminal state : 32
  - UNSHADED states : 10, 18, 24, 26, 30, 31, 34
- there are 6 distinct paths from the starting state 2 (root) to a terminal state (leaf node).
  - path A : $2 \rightarrow 8$
  - path B : $2 \rightarrow 10 \rightarrow 16$
  - path C : $2 \rightarrow 10 \rightarrow 18 \rightarrow 24 \rightarrow 30 \rightarrow 31 \rightarrow 32$
  - path D : $2 \rightarrow 10 \rightarrow 18 \rightarrow 24 \rightarrow 32$
  - path E : $2 \rightarrow 10 \rightarrow 18 \rightarrow 26 \rightarrow 32$
  - path F : $2 \rightarrow 10 \rightarrow 18 \rightarrow 26 \rightarrow 34 \rightarrow 35$

path	length	R_{acc}	$r_s = -4$	$r_s = -1$	$r_s = 0$	$r_s = 1$	ending state
A	1	$1 * r_s + r_r = 1r_s - 5$	-9 (max)	-6	-5	-4	RED
B	2	$2 * r_s + r_r = 2r_s - 5$	-13	-7	-5	-3	RED
C	6	$6 * r_s + r_g = 6r_s + 5$	-19	-1	5 (max)	11 (max)	GREEN
D	4	$4 * r_s + r_g = 4r_s + 5$	-11	1 (max)	5 (max)	9	GREEN
E	4	$4 * r_s + r_g = 4r_s + 5$	-11	1 (max)	5 (max)	9	GREEN
F	5	$5 * r_s + r_r = 5r_s - 5$	-25	-10	-5	0	RED

Let R_{acc} represents the reward accumulated along the path.

A policy is a function that maps S to A.
Let $\searrow$ represents the action "right and down".
Let $\nearrow$ represents the action "right and up".
We represent the function as a dictionary, where the key is the state, and the value is the action.

path	policy	states
A	{2:$\nearrow$}	$2 \rightarrow 8$
C	{2:$\searrow$,10:$\searrow$,18:$\nearrow$,24:$\nearrow$}	$2 \rightarrow 10 \rightarrow 18 \rightarrow 24 \rightarrow 30 \rightarrow 31 \rightarrow 32$

path	policy	states
D	{2:↘,10:↘,18:↗,24:↘}	$2 \to 10 \to 18 \to 24 \to 32$
E	{2:↘,10:↘,18:↘,26:↗}	$2 \to 10 \to 18 \to 26 \to 32$

Without considering the discount factor γ, the optimal policy(ies) corresponds(x) to the path(s) that renders(x) the max R_{acc}. | r_s | optimal path | unique? | | — | — | — | | -4 | A | unique | | -1 | D & E | not unique | | 0 | C & D & E | not unique | | 1 | C | unique |

0.0.4 Types of Paths (Policies)

We can categorize paths into 2 types. - $Type_G$: Paths end at a **GREEN** terminal state.
- $Type_R$: Paths end at a **RED** terminal state.

Define $L_{Gmax} = \max_{P \in Type_G} |P|$, the longest length among all paths that are in $Type_G$
Define $L_{Gmin} = \min_{P \in Type_G} |P|$, the shortest length among all paths that are in $Type_G$
Define $L_{Rmax} = \max_{P \in Type_R} |P|$, the longest length among all paths that are in $Type_G$
Define $L_{Rmin} = \min_{P \in Type_R} |P|$, the shortest length among all paths that are in $Type_G$

Here, the length of path equals to the number of actions in a policy excluding action taken at the terminal state.

path	length	R_{acc}	$r_s = -4$	$r_s = -1$	$r_s = 0$	$r_s = 1$	ending state	type
A	1	$1 * r_s + r_r = 1r_s - 5$	-9 (max)	-6	-5	-4	RED	type R
B	2	$2 * r_s + r_r = 2r_s - 5$	-13	-7	-5	-3	RED	type R
C	6	$6 * r_s + r_g = 6r_s + 5$	-19	-1	5 (max)	11 (max)	GREEN	type G
D	4	$4 * r_s + r_g = 4r_s + 5$	-11	1 (max)	5 (max)	9	GREEN	type G
E	4	$4 * r_s + r_g = 4r_s + 5$	-11	1 (max)	5 (max)	9	GREEN	type G
F	5	$5 * r_s + r_r = 5r_s - 5$	-25	-10	-5	0	RED	type R

$L_{Gmax} = \max_{P \in Type_G} |P| = \max\{|A|, |B|, |F|\} = \max\{1, 2, 5\} = 5$
$L_{Gmin} = \min_{P \in Type_G} |P| = \min\{|A|, |B|, |F|\} = \min\{1, 2, 5\} = 1$
$L_{Rmax} = \max_{P \in Type_R} |P| = \max\{|C|, |D|, |E|\} = \max\{6, 4, 4\} = 6$
$L_{Rmin} = \min_{P \in Type_R} |P| = \min\{|C|, |D|, |E|\} = \min\{6, 4, 4\} = 4$

if $r_s > 0, R_{acc} = \max\{L_{Rmax} * r_s + r_r, L_{Gmax} * r_s + r_g\} = \max\{5r_s - 5, 6r_s + 5\} = \max\{0, r_s + 10\} - 5 + 5r_s$
if $r_s = 0, R_{acc} = \max\{r_r, r_g\} = \max\{-5, 5\}$
if $r_s < 0, R_{acc} = \max\{L_{Rmin} * r_s + r_r, L_{Gmin} * r_s + r_g\} = \max\{1r_s - 5, 4r_s + 5\} = \max\{0, 3r_s + 10\} - 5 + r_s$

if $r_s > 0, r_s + 10$ is larger \$ $\to$\$ longest Type_G path(s) is(are) optimal
if $r_s = 0, r_g$ is larger \$ $\to$\$ all Type_G path(s) is(are) optimal
if $r_s < 0$,
- if $3r_s + 10 > 0 :$ $3r_s + 10$ is larger \$ $\to$\$ shortest Type_G path(s) is(are) optimal
- if $3r_s + 10 = 0 :$ \$ $\to$\$ shortest path(s) is(are) optimal
- if $3r_s + 10 < 0 :$ 0 is larger \$ $\to$\$ shortest Type_R path(s) is(are) optimal

if $r_s > 0$, longest path(s) that ends(x) at the GREEN state is(are) optimal

if $r_s = 0$, all path(s) that ends(x) at the GREEN state is(are) optimal
if $r_s < 0$,
- if $r_s > -10/3$: shortest path(s) is(are) that ends(x) at the GREEN state is optimal
- if $r_s = -10/3$: shortest path(s) is(are) optimal
- if $r_s < -10/3$: shortest path(s) that ends(x) at a RED state is(are) optimal

$r_s = 1 > 0$, path C is the longest path that ends at the GREEN state $ \rightarrow$ path C is optimal.
$r_s = 0$, path C,D,E end at the GREEN state $ \rightarrow$ path C,D,E are optimal.
$r_s = -1 > -10/3$: path D,E are the shortest paths that end at the GREEN state $ \rightarrow$ path D,E, are optimal.
$r_s = -4 < -10/3$: path A is the shortest path that ends at a RED state $ \rightarrow$ path A is optimal.

We may preview question 1-(b)

```
What value of r_s from 1-(a) would cause the optimal policy to return the shortest path to the
```

```
Answer : r_s = -1
```

0.0.5 Reduction of the search space

There are 12 explored states: - staring state : 2 - RED terminal states : 8, 16, 35 - GREEN terminal state : 32 - UNSHADED states : 10, 18, 24, 26, 30, 31, 34

For a terminal state $s_{terminal}$, since taking an action on $s_{terminal}$ will end the episode, there is NO next state s' to transition to.
Consequently, formula $V_k^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s)V_{k-1}^\pi(s')$ can be reduced to $V_k^\pi(s_{terminal}) = R(s_{terminal})$
This implies that $V_k^\pi(s_{terminal})$ will remain invariant after the 1st iteration.
- RED : $V_k^\pi(s_{red}) = R(s_{red}) = r_r = -5, \forall k > 1$
- GREEN : $V_k^\pi(s_{green}) = R(s_{green}) = r_g = 5, \forall k > 1$

So, there is no need to take terminal states into the calculation of convergence as their values/utilities remain invariant $V^\pi(s_{terminal}) = R(s_{terminal})$.
- RED : $V^\pi(s_{red}) = R(s_{red}) = r_r = -5$
- GREEN : $V^\pi(s_{green}) = R(s_{green}) = r_g = 5$

Also, we know that, 1. taking any action on state 31 will result in a transition to state 32
$V_1(31) = R(31) + \gamma * P(32|31) * V_0(32)$
$V_1(31) = R(31) + \gamma * P(32|31) * r_g$
$V_1(31) = R(31) + \gamma * r_g$
$V_1(31) = r_s + \gamma * r_g$
$V(31) = r_s + \gamma * r_g$ is a constant

2. taking any action on state 30 will result in a transition to state 31
   $V_1(30) = R(30) + \gamma * P(32|31) * V_0(31)$
   $V_1(30) = R(31) + \gamma * P(32|31) * (r_s + \gamma * r_g)$
   $V_1(30) = R(31) + \gamma * (r_s + \gamma * r_g)$
   $V_1(30) = r_s + \gamma * (r_s + \gamma * r_g)$
   $V_1(30) = r_s * (1 + \gamma) + \gamma^2 * r_g$
   $V(30) = r_s * (1 + \gamma) + \gamma^2 * r_g$ is a constant

3. taking any action on state 34 will result in a transition to state 35

$V_1(34) = R(34) + \gamma * P(35|34) * V_0(35)$
$V_1(34) = R(31) + \gamma * P(32|31) * r_r$
$V_1(34) = R(31) + \gamma * r_r$
$V_1(34) = r_s + \gamma * r_r$
$V(34) = r_s + \gamma * r_r$ is a constant

Therefore, the only states we need to consider in the Policy Iteration Algorithm are : [2, 10, 18, 24, 26]

- (deterministic) Bellman backup for a particular policy π, this is a part of policy evaluation because this is trying to figure out how good is a particular policy in a decision process.)
  - Iteration of Policy Evaluation : $V_k^\pi(s) = R^\pi(s) + \gamma \sum_{s' \in S} P^\pi(s'|s) V_{k-1}^\pi(s')$
- optimal policy : $\pi^*(s) = \arg\max_\pi V^{\pi(s)}$
  - There exists a unique optimal value function, but the optimal policy MAY NOT be unique.
- Policy Search :
  - Number of deterministic policies = $|A|^{|S|} = 2^{12} = 4096$
- Policy Iteration :
  - set i = 0
  - initialize $\pi_0(s)$ randomly for all states s.
  - while i == 0 or $\|\pi_i - \pi_{i-1}\|_1 > 0$
    - $V^{\pi_i} \leftarrow$ MDP V function policy **evaluatio** of π_i
    - $ _{i+1} ←$ policy **improvement**
    - i ← i + 1
- State-action value of a policy π
  - $Q^\pi(s,a) = R(s,a) + \gamma * \sum_{s \in S} P(s'|s) V^\pi(s')$
- Compute State-action value of a policy π_i
  - For $s S, a A: $
    $Q^{\pi_i}(s,a) = R(s,a) + \gamma * \sum_{s \in S} P(s'|s) V^{\pi_i}(s')$
- Compute new policy $\pi_{i+1}, \forall s \in S$:
  - $\pi_{i+1}(s) = argmax_a Q^{\pi_i}(s,a), \forall s \in S$