Report – Project 1

Name: Sandesh Kumar Srivastava

UBIT: Sandeshk

Part 1 - Build a deterministic environment

I have defined a 4x4 grid environment with the following MDP formulation:

State set(S) = {S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16}

Action set(A) = {Down, Up, Right, Left}

Reward set(R) = {0, -1, 2, 4, 1, 20}

Start position for agent is defined as [0,0] and goal position is defined as [3,3].

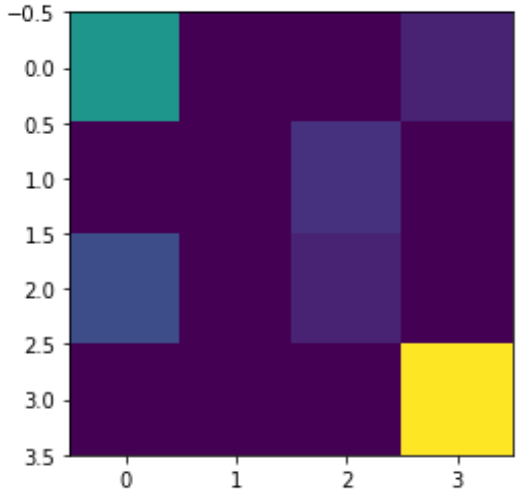
Maximum number of steps for an episode is set as 7.

Rewards are defined for each state as:

0	-1	-1	**1**
-1	-1	**2**	-1
4	-1	**1**	-1
-1	-1	-1	**20**

Objective of the game for the agent is to reach the goal state within allowed maximum number of steps and collect maximum rewards in the process. If the agent cannot reach the goal within maximum number of steps, the game terminates.

So, the initial configuration of the grid looks like:



For the deterministic environment, $P(s',r|s,a) = \{0,1\}$ i.e. the probability of going from state s to another state s' by taking action a and getting reward r can be either 0 or 1. In other words, it is definite whether $P(s',r|s,a)$ will be either 0 or 1.

So, in the deterministic environment, when we take an action we know for sure what will be the next position for agent and how much reward will it get. The logic for this is defined in step() function of the environment.

Part 2 - Build a stochastic environment

For the stochastic environment $\sum P(s',r|s,a) = 1$ i.e. the summation of probabilities of going from state s to another state s' by taking action a and getting reward r for all possible states s' will be 1. In other words, the value of $P(s',r|s,a)$ will be some value between 0 and 1 and the sum over all possible states s' will be 1.

So, in the stochastic environment, when we take an action there is a probability about the next position for the agent and the corresponding reward that it will get. The logic for this is defined in step function of the environment. I create a random variable with probability distribution [0.94, 0.02, 0.02, 0.02]. If the first number is generated (94% times), I do not change the action but for other numbers generated (2% each), I change the action to a different action. This gives the environment a stochastic behavior where for most trials, it performs the requested action but for some random trials it performs a different action.

Transition probability matrix for state S1:

S	**S'**	**r**	**A**	**P(s',r|s,a)**
S1	S4	-1	Down	0.94
S1	S1	0	Down	0.02
S1	S2	-1	Down	0.02
S1	S1	0	Down	0.02
S1	S1	0	Up	0.94
S1	S4	-1	Up	0.02
S1	S2	-1	Up	0.02
S1	S1	0	Up	0.02
S1	S2	-1	Right	0.94
S1	S1	0	Right	0.02
S1	S1	0	Right	0.02
S1	S4	-1	Right	0.02
S1	S1	0	Left	0.94
S1	S4	-1	Left	0.02
S1	S2	-1	Left	0.02
S1	S1	0	Left	0.02

Transition probability matrix for state S10:

S	S'	r	A	P(s',r|s,a)
S10	S14	-1	Down	0.94
S10	S6	-1	Down	0.02
S10	S11	1	Down	0.02
S10	S9	4	Down	0.02
S10	S6	-1	Up	0.94
S10	S11	1	Up	0.02
S10	S9	4	Up	0.02
S10	S14	-1	Up	0.02
S10	S11	1	Right	0.94
S10	S9	4	Right	0.02
S10	S14	-1	Right	0.02
S10	S6	-1	Right	0.02
S10	S9	4	Left	0.94
S10	S14	-1	Left	0.02
S10	S6	-1	Left	0.02
S10	S11	1	Left	0.02

Similarly transition probabilities are defined for each state and corresponding action.

Broadly following classes are defined for this implementation:

1. DeterministicGridEnvironment: It defines a deterministic environment for the specified configuration.
2. StochasticGridEnvironment: It defines a stochastic environment for the specified configuration.
3. RandomAgent: It defines a random agent which takes a random action at any given time. The agent follows a random policy in which all actions are equally likely for any state.
4. DynamicProgrammingAgent: It defines an agent which employs Dynamic Programming to learn better policy. The agent starts with a random policy and it performs policy evaluation followed by policy improvement till it gets a stable policy. It then uses this policy to take better action for each state.

The two environment follows the OpenAI Gym structure and consists of following methods:

1. __init__: I override the __init__ method to define observation_space and action_space according to number of states and actions respectively. I also define the reward structure for the environment here.
2. reset: Rests the environment state before starting a new episode.
3. step: Updates the environment when an agent performs a specified action. It returns the following parameters:
    - observation: the updated state of the environment.
    - reward: Reward that agent collected by performing this action.
    - done: Flag to indicate if agent reached goal position or exhausted all the allowed maximum number of steps
    - info: Used to return the current agent state.

Part 3 - Implement tabular method

I have used Dynamic Programming method to solve the two defined environments. Brief description of the methods implemented:

1. __init__: I override the __init__ method to get environment information, initialize V values and policy for all states.
2. policy_evaluation: Performs the policy evaluation part by calculating new V Values for states and continue doing this till the maximum change in V value for any state is smaller than threshold which is set at 0.5. For calculating V, I am using gamma which is also set as 0.5.
3. policy_improvement: Once the policy evaluation is completed, I perform policy improvement to check if the provided policy is stable or not. If there is no change in policy for any of the states with the updated policy, I treat this as policy as stable.
4. probability_to_transition: Helper function to check the probability of transitioning from state s1 to s2 by performing an action.
5. step: Function to select action based on the policy defined.

Policy Evaluation:

For policy evaluation, I start with the initial value of V as 0 for all the states. Policy for each state is also initialized as equal probability of 0.25 for each of the four actions. At each iteration, I update the V value of state based on following rule for all actions:

```
V_S_next += self.policy[S][action]*self.probability_to_transition(S,S_prime,action)*(self.reward[S_prime]+gamma*V[S_prime])
```

I have kept gamma and threshold to be 0.5. I calculate delta as `delta = max(delta,abs(V_S - V_S_next))`

where V_S is the old value of state and V_S_next is the new V value for this state. I calculate V values for states as long as delta is more than threshold.

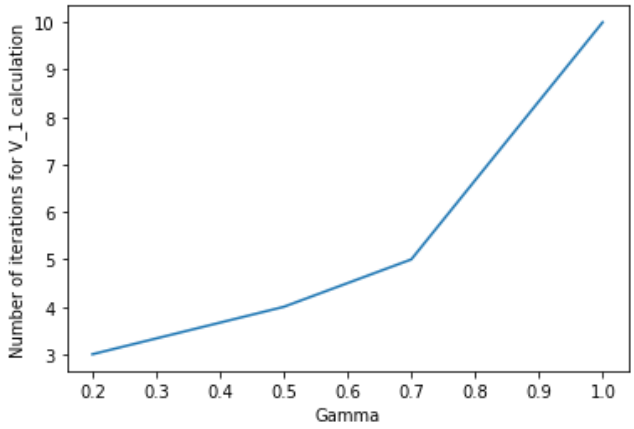
Policy Improvement:

For policy improvement, I calculate the new probabilities for each action based on the V values calculated in policy evaluation. If there is no change in policy for any state then the method terminates as the policy is considered stable otherwise I perform policy evaluation again.

Effect of discounting factor gamma on algorithm:

Keeping other parameters constant, I tried to see the effect of gamma on the algorithm and the policy learnt.

Below is a graph depicting the number of iterations taken for calculating V_1 from V_0 for different gamma values.



Another important observation is that for gamma = 1.0, the reward collected is 0 and the agent is not able to reach the goal state. The final V value calculated is:

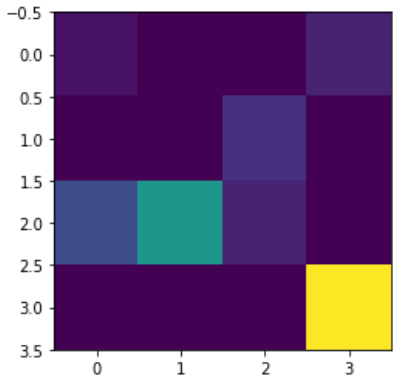
```
[[-100.    -38.55   -9.46    0.4 ]
 [ -36.73  -15.97   19.25   11.36]
 [  -8.64   19.25   18.25  120.  ]
 [   0.57    9.99  120.    100.  ]]
```

The problem happens at states S15 (or S12) with V value as 120. Here, the policy thinks that the best action for this state is to go down (or go right) as that gives the maximum V value. Because of this the agent gets stuck in these states. So, we need to reduce the gamma to reduce importance of future reward close to goal.

Results:

The performance of RandomAgent and DynamicProgrammingAgent is observed on both deterministic and stochastic environment.

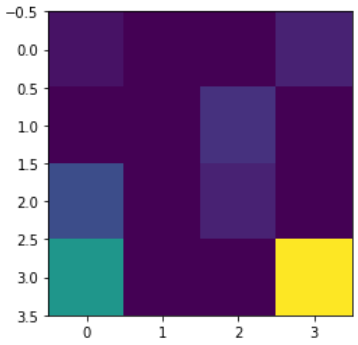
1. **Random agent behavior for Deterministic environment**
   The agent takes random action and performs very poorly for almost all runs. One sample run with end state is depicted below.




```
Total reward:  -6
Total number of steps:  7
```

2. **Random agent behavior for Stochastic environment**

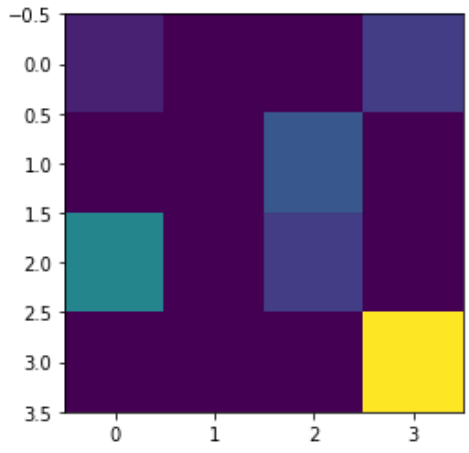
   The agent takes random action and performs very poorly for almost all runs. One sample run is depicted below.




```
Total reward:  8
Total number of steps:  7
```

3. **DP trained agent behavior for deterministic environment**

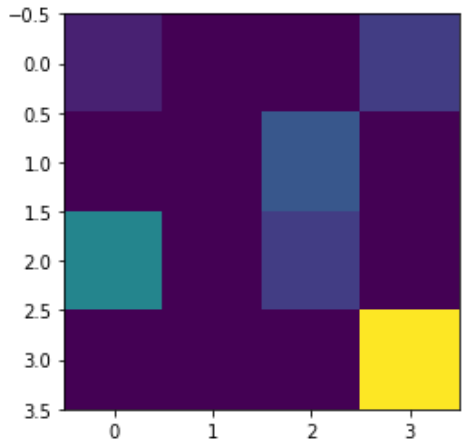
   The agent takes action according to the policy learnt and is mostly always able to reach the goal. One sample run is depicted below.




```
Total reward:  22
Total number of steps:  6
```

4. **DP trained agent behavior for Stochastic environment**

   The agent takes action according to the policy learnt and is mostly always able to reach the goal. The output is mostly similar to the deterministic environment as shown in 3. However, sometimes due to stochastic nature the agent performs a different action. One such sample run is depicted below.




```
Total reward:  22
Total number of steps:  7
```

Conclusion:

As we can see from the results that the DP trained agent is able to learn an efficient policy and perform better than the random agent for both deterministic and stochastic environment.