Introduction to Machine Learning

Project 4 Report

UBID: sppandey@buffalo.edu

PERSON NO: 50288608

Objective :

In the given project we are dealing with an agent who is trying to achieve the4 shortest path to his goal in a given environment. For our given problem we have a5x5 grid world, wherein a green square which is controlled by the agent and has to navigate it to the yellow square which is treated as the goal.

Ground Rules:

1) Agent has 100 steps to reach to his goal.
2) Every time the agent achieves the goal a reset happens which places the green block and yellow block back to their original places.
3) Learning Step: The agent needs to determine the optimal path to the goal
4) Rewards and Penalties: There will be a positive point for an optimal step and for negative step a point will be deducted
5) Possible actions for agent – left, right, up, down.

How can this be achieved without any given data set?

Answer:

Reinforcement Learning :

Apart from supervised and unsupervised learning we have another type of learning that derives an optimal solution on the basis if trial and error. Reinforcement learning buttresses the concept of agent, target and optimal path.Instead of training on a predetermined dataset or solutions the agent learns how to behave by performing actions and observing the results .

The feedback rewards from trial and error act as the learning steps for the agent who in turn maps states to action to maximize the long-term total reward as a delayed supervision signal.

Referring to the data given in the description the following parameters play a crucial role to implement successful reinforcement learning.

Basic reinforcement is modeled as a Markov decision process a 5-tuple (S, A, P_a, R_a, γ), where

- S is a finite set of states
- A is a finite set of actions
- $P_a(s,s') = Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to

  state s' at time t+1

- $R_a(s,s')$ is the immediate reward (or expected immediate reward) received after transitioning from

  state s to state s' due to action a

- $\gamma \in [0, 1)$ is the discount factor, which represents the difference in importance between future rewards and present rewards.

How can the Agent remember the results?

Answer: The agent uses something called as Experience replay. Experience replay derives its roots from the topic of Deep -Q-Learning and thus helps the agent to remember previous results.

How does it work?

Answer:

1) Storing the agents experiences and randomly drawing batches to train the network.
2) Drawing random experiences to train the network prevents the network from learning only the latest steps.
3) What if the steps are in millions the question about storage comes in place, for this the experience replay buffers stores a fixed number of recent memories and old ones are removed.
4) When needed, we can pull put a random batch from buffer and train the network.

How does the agent Determines if the step will reward him or not?
Answer:

In a given environment where agent is unsure where the next action would be, it is better for the agent to try all kinds of things before it starts to see the patterns. This can be done by selecting an action by a certain percentage which is called as 'exploration rate' or 'epsilon'

Hyperparameter Tuning for this project.

In order to obtain an optimal mean reward we will be tuning few parameters such as epsilon min and max, lambda value, Gamma value and learning steps in our testing.

Code Implementations :

1. **Build a 3-layer neural network using Keras library :**

We have introduced two hidden layers both with activation function as 'relu' and having 128 nodes. For the final layer the activation function is set to 'linear'

```
### START CODE HERE ### (≈ 3 lines of code)

model.add(Dense(128, input_dim=state_dim, activation = 'relu'))
model.add(Dense(128, activation = 'relu'))
model.add(Dense(action_dim, activation = 'linear'))

### END CODE HERE ###
```

2. **Implement exponential-decay formula for epsilon : As discussed previously, agent selects an action based on a certain percentage. The formula for that is as follows;**

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-\lambda|S|},$$

where

$\epsilon_{min}, \epsilon_{max} \in [0, 1]$

λ - hyperparameter for epsilon

$|S|$ - total number of steps

We implement it as part of the code:

```
### START CODE HERE ### (≈ 1 line of code)
#Exponential decay formula to determine exploration rate
self.epsilon = self.min_epsilon + (self.max_epsilon - self.min_epsilon) * (math.e ** (-(self.lamb)* (self.steps)))

### END CODE HERE ###
```

3. **Implement Q-function : The formula in theory is given as :**

$$Q_t = \begin{cases} r_t, & \text{if episode terminates at step } t+1 \\ r_t + \gamma max_a Q(s_t, a_t; \Theta), & \text{otherwise} \end{cases}$$

Implemented code:

```
  ### START CODE HERE ### (≈ 4 line of code)
  #Q-Function Implementation

  if st_next is None:
    t[act] = rew
  else:
    t[act] = rew + self.gamma * np.max(q_vals_next)

  ### END CODE HERE ###
```

Scenarios tested with different Hyperparameters:

Since we had plethora of options to work with, we have tried to run the model with 3 different setting and have listed out observations:

Scenario 1:

For Scenario 1 we have decided to use the provided configuration and changed the number of steps to 25000

Other parameter values are as follows:

```
[ ]  #-------------------- AGENT ---------------------------

     class Agent:
       """The agent, which learns to navigate the environment

       """

       def __init__(self, state_dim, action_dim, memory_capacity = 10000,
                    batch_size = 64, gamma = 0.10, lamb = 0.001,
                     max_epsilon = 1., min_epsilon = 0.01):
         self.state_dim = state_dim
         self.action_dim = action_dim
```

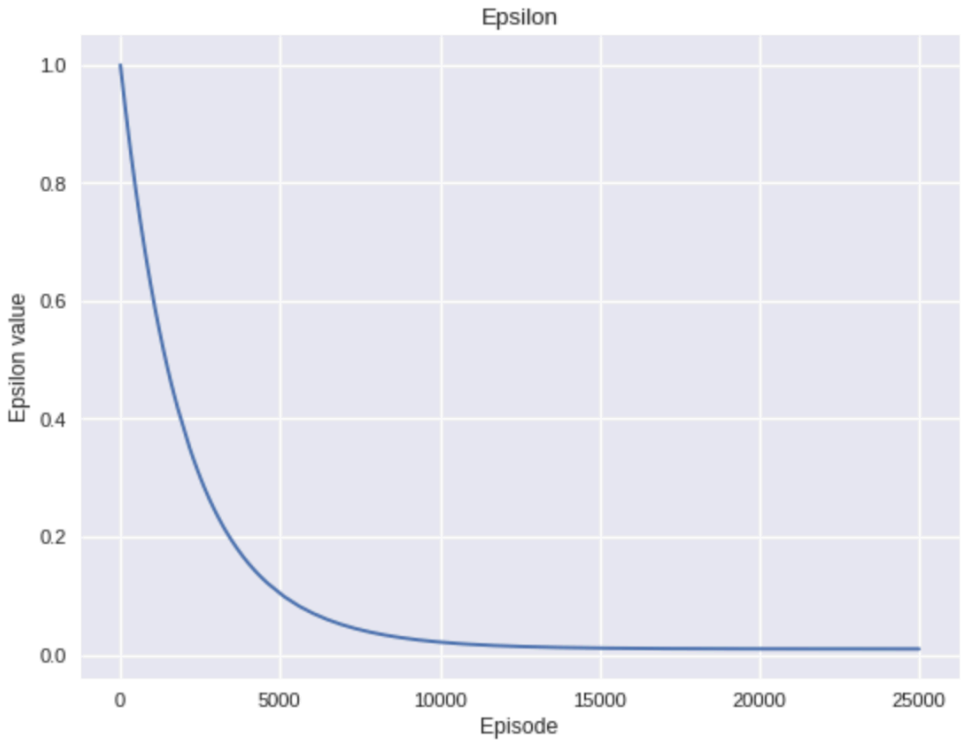
```
#-------------------- MAIN ----------------------------
print('Setting up environment')
env = Environment(5)

state_dim = 4
action_dim = 4 # left, right, up, down
print('Setting up agent')
MAX_EPSILON = 1 # the rate in which an agent randomly decides its action
MIN_EPSILON = 0.01 # min rate in which an agent randomly decides its action
LAMBDA = 0.00005      # speed of decay for epsilon
num_episodes = 25000 # number of games we want the agent to play
```

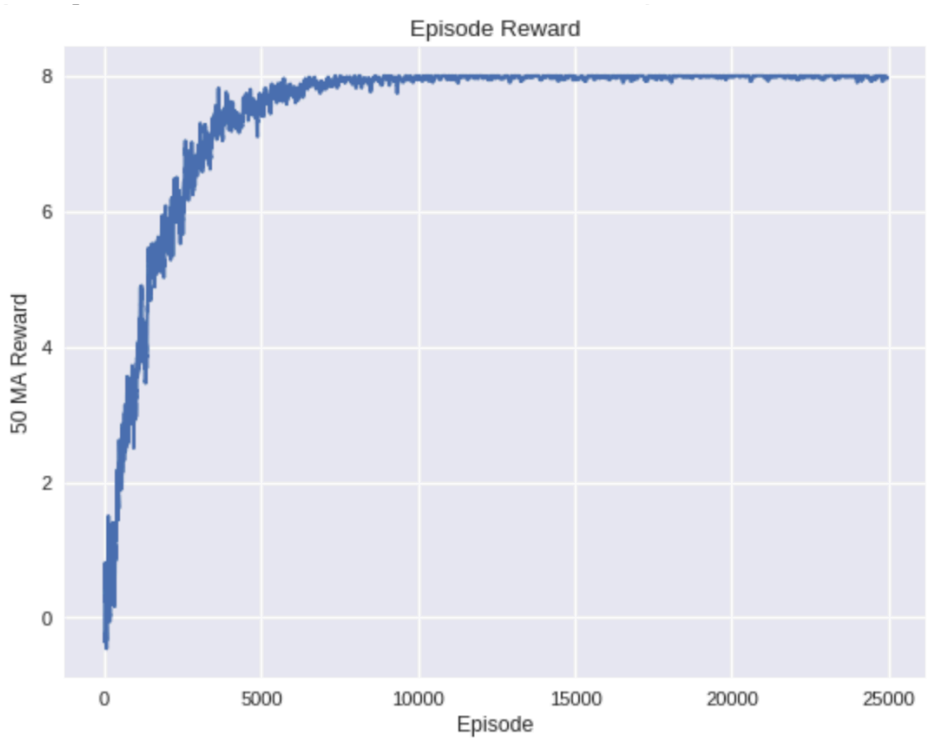
At the end of all episodes we got the best mean time as compared to all models:

```
Episode 24900
Time Elapsed: 970.76s
Epsilon 0.010028099737147397
Last Episode Reward: 8
Episode Reward Rolling Mean: 7.454618765372364
----------
```

Graph for epsilon vs episodes for the scenario:



Graph for reward per episode:



Scenario 2:

For Scenario 2 we decided to change lamba values, epsilon max values and number for steps. This setting showed an unsual dip in the mean rewards turning out not to be and optimal setting for the agent. The number of steps in this setting was set to 40000

Parameter values are as follows:

Gamma and Lambda values changed:

```
def __init__(self, state_dim, action_dim, memory_capacity = 10000,
             batch_size = 64, gamma = 0.99, lamb = 0.01,
              max_epsilon = 1., min_epsilon = 0.001):
  self.state_dim = state_dim
  self.action_dim = action_dim
```

Number of steps=40000 and LAMBDA=0.0005 :

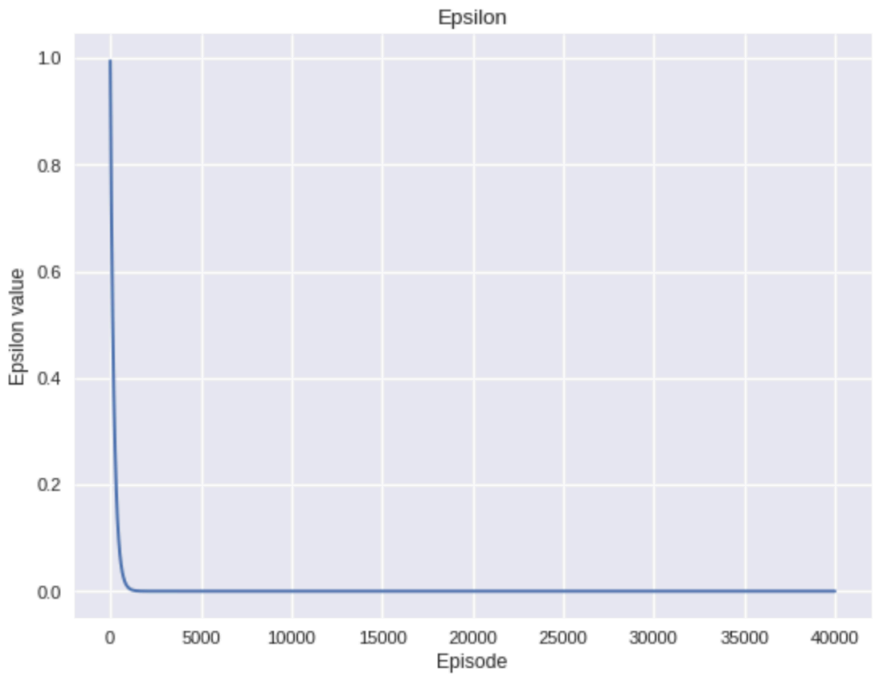
```
state_dim = 4
action_dim = 4 # left, right, up, down
print('Setting up agent')
MAX_EPSILON = 1 # the rate in which an agent randomly decides its action
MIN_EPSILON = 0.00001 # min rate in which an agent randomly decides its action
LAMBDA = 0.0005      # speed of decay for epsilon
num_episodes = 40000 # number of games we want the agent to play
```

As mentioned earlier the model didn't perform well thus reflecting a sudden dip in the mean rewards:

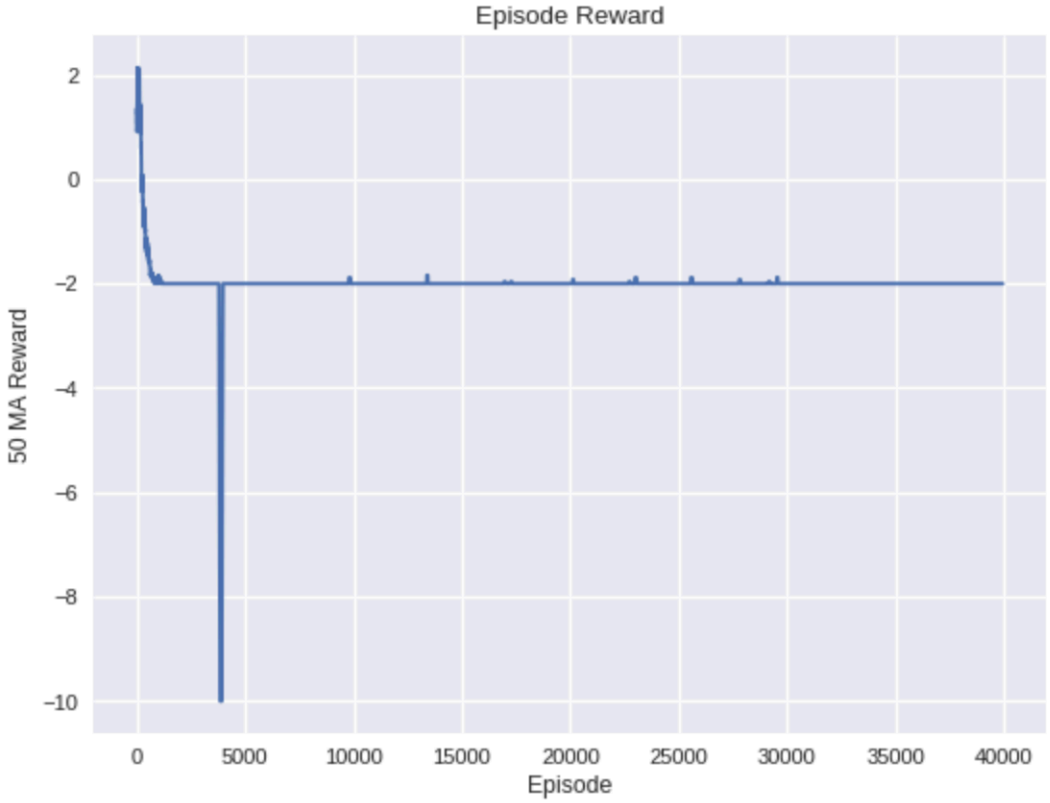
```
----------
Episode 20400
Time Elapsed: 935.39s
Epsilon 1e-05
Last Episode Reward: -2
Episode Reward Rolling Mean: -1.9864538692675238
----------
```

This by far was the worst case we tested for set of changed hyperparameters.

Graph for epsilon vs episodes for the scenario:



Graph for reward per episode:



Scenario 3:

After a disastrous run in Scenario 2 we decided to work only with epsilon values and steps as that would impact running time but at the same time improve the guess fir agent leading to better mean reward and our model didn't disappoint us.

Parameter values are as follows:

Number of steps=45000 and MIN_EPSILON=0.1 :

```
[20] #-------------------- MAIN ----------------------------
     print('Setting up environment')
     env = Environment(5)

     state_dim = 4
     action_dim = 4 # left, right, up, down
     print('Setting up agent')
     MAX_EPSILON = 1 # the rate in which an agent randomly decides its action
     MIN_EPSILON = 0.1 # min rate in which an agent randomly decides its action
     LAMBDA = 0.00005      # speed of decay for epsilon
     num_episodes = 45000 # number of games we want the agent to play
```

At the end of training the model and observing the mean rewards we found that episode 4000 the model gave the best mean reward which later dropped down by a small margin until all the episodes were finished.

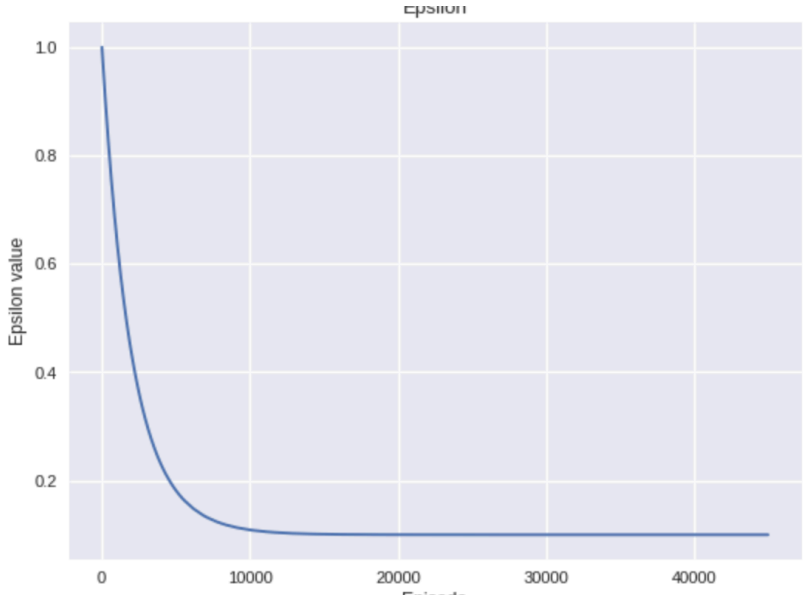
Best result :

```
----------
Episode 40000
Time Elapsed: 1608.38s
Epsilon 0.10000001975201125
Last Episode Reward: 8
Episode Reward Rolling Mean: 7.310167664970803
----------
```

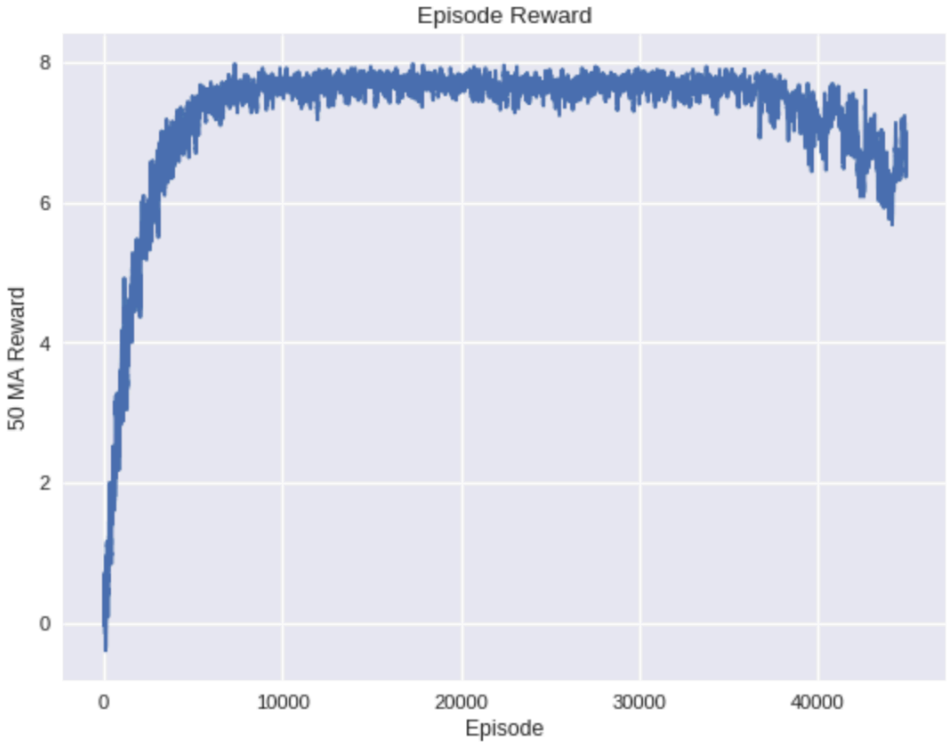
By the end of all episodes we had:

```
----------
Episode 44900
Time Elapsed: 1809.60s
Epsilon 0.10000000202537593
Last Episode Reward: 6
Episode Reward Rolling Mean: 7.258052275618848
----------
```

Graph for epsilon vs episodes for the scenario:



Graph for reward per episode:



Writing Task :

1. **Explain what happens in reinforcement learning if the agent always chooses the action that maximizes the Q-value. Suggest two ways to force the agent to explore.**

   Answer:

   The Q-learning algorithm can be explored using two ways:

   1) Exploration
   2) Exploitation

   Using one of the two ways above totally impact the Q-value , however it depends on the agent the method he chooses to opt for. Given a situation the agent may need to make complete random guess as in case of taking the first action. On the contrary he may materialized experience replay and thus can take an informed action.

   Let's look in detail how the methods selected above can have impact.

   Exploration: Chances of taking this step is about 10% of the, as the agent may need to take a completely random action in order to acquire new experiences and results. The strategy function may restrict us to do so. The epsilon value governs the number of random moves the agent can make thus it is set 0.1 making it a ratio of 10:1 of moves: random move

   Exploitation: An agent after some moves may take this method and thus has 90% chances of getting picked up \as the agent has learned from the previous step. Thus in a way the agent already knows whats the next best action could be as he already explored it in previous step.

   Thus using the above two methods the agent may be forced to explore.

2. **Calculate Q-value for the given states and provide all the calculation steps.**

   Consider a deterministic environment which is a 3x3 grid, where one space of the grid is occupied by the agent (green square) and another is occupied by a goal (yellow square). The agent's action space consists of 4 actions: UP, DOWN, LEFT, and RIGHT. The goal is to have the agent move onto the space that the goal is occupying in as little moves as possible. The episode terminates as soon as the agent reaches the goal.

   Initially, the agent is set to be in the upper-left corner and the goal is in the lower-right corner. The agent receives a reward of:

   1 when it moves closer to the goal

-1 when it moves away from the goal

0 when it does not move at all (e.g., tries to move into an edge)

Consider the following possible optimal episode and their resulting states, that reach the goal in the smallest number of steps. In the example below, s4 is a terminal state.

Answer :

Initial Stage:

State	UP	DOWN	LEFT	RIGHT
S0				
S1				
S2				
S3				
S4	0	0	0	0

As soon as agent has achieved the goal the episode terminates at that moment. Thus if agent is at S4 state it implies that it's the end of episode and thus agent will have no more moves to make. Thus using the formula for Q function as follows.

Starting from the bottom :

(S3,Down) = 1(rt : reward for correct action)

Since episode terminates at (t+1) step for (s3,Down)

(S2,Right) = 1(rt : reward for correct action) + 0.99* 1

(S2,Right) = 1.99

(S1,Down) = 1(rt : reward for correct action) + 0.99* 1.99

(S1, Down) = 2.97

(S0, Right) = 1(rt : reward for correct action) + 0.99* 2.97 = 3.94

(S0, Down) = 1(rt : reward for correct action) + 0.99* 2.97 = 3.94

(S1,Right) = 1(rt : reward for correct action) + 0.99* 1.99 = 2.97

(S2,Down) = 1(rt : reward for correct action) + 0.99* 1 = 1.99

Actions that lead to crossing over of edges leads to reward is 0 and next state is the current state

(S0, UP) = 0 + 0.99 * 3.94 = 3.90

(S0, Left) = 0 + 0.99 * 3.94 = 3.90

(S1, UP) = 0 + 0.99 * 2.97 = 2.94

(S3, Right) = 0 + 0.99 * 1 = 0.99

If the agent takes actions that takes him away from answer penalty of -1 is given;

(S3, UP) = -1 + 0.99 * 1.99 = 0.97

(S3, LEFT) = -1 + 0.99 * 1.99 = 0.97

(S2, UP) = -1 + 0.99 * 2.94 = 1.94

(S2, LEFT) = -1 + 0.99 * 2.94 = 1.94

(S1, LEFT) = -1 + 0.99 * 3.94 = 2.90

Thus populating all values in table we get

State	UP	DOWN	LEFT	RIGHT
S0	3.90	3.94	3.90	3.94
S1	2.94	2.97	2.90	2.97
S2	1.94	1.99	1.94	1.99
S3	0.97	1	0.97	0.99
S4	0	0	0	0