Introduction to Machine Learning

Project 4: Reinforcement Learning On a Grid Based Agent

Date: 12/4/2019

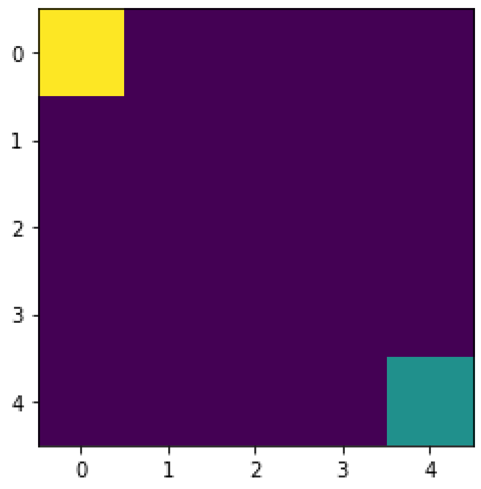
Author: Rugved Hattekar
Person #: 50320920

Introduction:

The aim of the project is to build a build a reinforcement learning agent to navigate the classic 4x4 grid-world environment. Reinforcement Learning is an area of Machine Learning which focuses on creating agents capable of taking actions in an environment in a way that maximizes rewards over time. In this assignment, the agent will learn an optimal policy through Q-Learning which will allow it to take actions to reach a goal. The environment and agent will be built to be compatible with OpenAI Gym environments. OpenAI provides simple physics or gaming simulations that allow you easily test out your RL skills. The objective of the reinforcement learning is to learn to act in a way that will maximize its expected long-term rewards. The algorithm used by the agent to determine its actions is called its policy. Hence we are here to design a policy which would help the agent to navigate to the goal location in shortest path with maximum reward.

Environment:

The environment we provide is a basic deterministic n × n grid-world environment (the initial state for an 4 × 4 grid-world is shown in Figure 3) where the agent (shown as the yellow square) has to reach the goal(shown as the green square) in the least amount of time steps possible.
The environment's state space will be described as an n × n matrix with real values on the interval [0, 1] to designate different features and their positions. The agent will work within an action space consisting of four actions: up, down, left, right. At each time step, the agent will take one action and move in the direction described by the action. The agent will receive a reward of +1 for moving closer to the goal and −1 for moving away or remaining the same distance from the goal.



Environment Agent and Goal. Agent at (0,0), Goal at (4,4) Extracted from OpenAI Gym.

Reinforcement Learning:

Reinforcement Learning is an area of machine learning. Reinforcement learning addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals. In machine learning the we usually formulate the problem based on the Markov Decision process. In a Markov decision process (MDP) the agent can perceive a set S of distinct states of its environment and has a set A of actions that it can perform. At each discrete time step t, the agent senses the current state, chooses a current action a, and performs it. The environment responds by giving the agent a reward and moving to the next step. The task of the agent is to learn a policy, for selecting its next action, based on the current observed state. The best policy for the robot can be to produce the greatest possible cumulative reward for the robot over time. To do so the discount factor is provided to the near future rewards so to give little weight to the future steps and robot could possibly estimate the best possible policy which could reward more. The Markov Bellman equation describes the discount factor with the policy depended on the current state, action, future state and discount factor.

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma . V^*(s')] \quad \text{for all } s$$

- $T(s, a, s')$ is the transition probability from state s to state s', given that the agent chose action a.
- $R(s, a, s')$ is the reward that the agent gets when it goes from state s to state s', given that the agent chose action a.
- γ is the discount rate.

Img.1: The Markov Equation. The V* here takes summation of current state and future states with a discount factor in consideration.

The Q-Learning Policy:

Q-Learning is a simple algorithm for comparing the expected utility for a set of actions given the current state. We only need to consider the most recent state when making a decision because we assume the environmental to be a Markov chain. This simply means that the current state completely characterizes the process.

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Img.2: Q-Learning Algorithms: Source Wikipedia

In simple words, the function Q will tell us which action will lead to which expected cumulative discounted reward, and our policy π will choose the action a which, ideally, will lead to the maximum such value given the current state.

As our agent takes and receives rewards we can update the q_table to perceive the favorability of the action taken. The future value is calculated by taking the difference between the future value of the current state and the future value estimate of the old state and action pair. This difference times the discount factor, epsilon, plus the reward is then appended to our previous state action pair.

The Pseudo Q-Learning Algorithm Training:
The Q-Learning algorithm for deterministic Markov Decision process using q-tables can be explained as follows. The discount factor y may be any constant such that 0 5 < gyamma < 1.
Pseudo Algo:

Step 1: Initially all the states and actions and rewards are initialize as zeros.
Step 2: Observe the current state s
Step 3: Do while:
Step 4: Select and action a and execute it. The current actions can be anything since initially the robot is in the exploration stage.
Step 5: Receive the reward.
Step 6: Observe the new state.
Step 7: Store every value to the Q-table in the following manner.

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

Q-L Update step: Source: Tom Mitchell Intro to ML

Essentially a Q-Learn Table is just a matrix covering all of the possible state and action combinations. We start the Q-Table by initializing an empty numpy matrix. To create the q-table we have considered the state[0] state[1](that is the next state) and action performed. This value then updated as considered as previous values.

Example in Template Code:

Heuristic Agent:

The Heuristic Policy is such that the agent calculates the heuristic distance between the goal position and the current position and tries to minimize the distance between the Goal and current Position. Each step it takes so that the distance between goal and current position is reduced. The Heuristic agent is used in many robotics application.

Random Agent:

The Random Agent is just that it executes the random actions so that robot can move in the Environment.

OpenAI Gym:

OpenAI gym is a toolkit that provides a wide variety of simulated environments (Atari games, board games, 2D and 3D physical simulations, and so on), so you can train agents, compare them, or develop new RL algorithms.

Hyper-Parameters:

Alpha (learning rate): The alpha probability determines how much the agent values newly acquired information over the older data set. If alpha was 0 the agent would learn nothing new, and at a value of 1 would only make decisions based on the most recent data.

Epsilon: Our agent will randomly select its action at first by a certain percentage, called ‘exploration rate’ or ‘epsilon’. This is because at first, it is better for the agent to try all kinds of things before it starts to see the patterns. When it is not deciding the action randomly, the agent will predict the reward value based on the current state and pick the action that will give the highest reward. We want our agent to decrease the number of random action, as it goes, so we introduce an exponential-decay epsilon, that eventually will allow our agent to explore the environment. (Explained in handout)

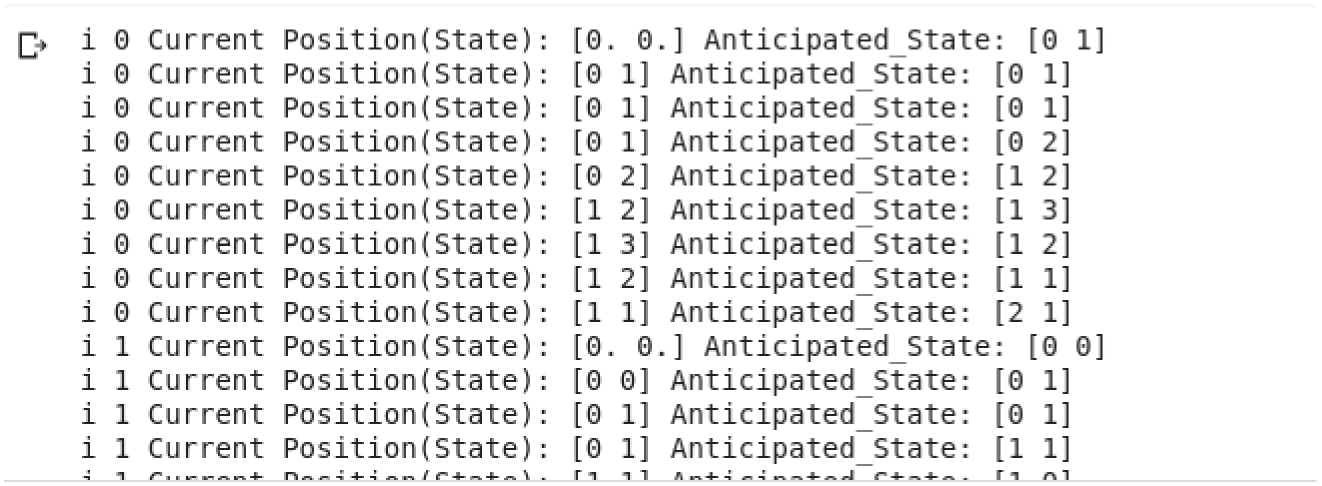
Gamma (discount factor): This rate determines how to preference rewards happening sooner rather than later. A discount factor of 0 would only consider the next reward; while 1 would give all future rewards equal weight. The goal is to keep the thing upright as long as possible, so we will weight all future rewards equally.

Hyper-parameter Decay

As our agent learns by trial and error, our q-table sees more action pairs and becomes more reliable. We will want our exploration and learning rate to gracefully decay as the model improves.
The decay factor is a delicate balance. Too fast: the model will converge on a poor local minimum. Too long: vanishing gradients causing erratic behavior. To calculate epsilon we can simply take a decay factor and raise it to the power of the episode number(0 < df < 1)

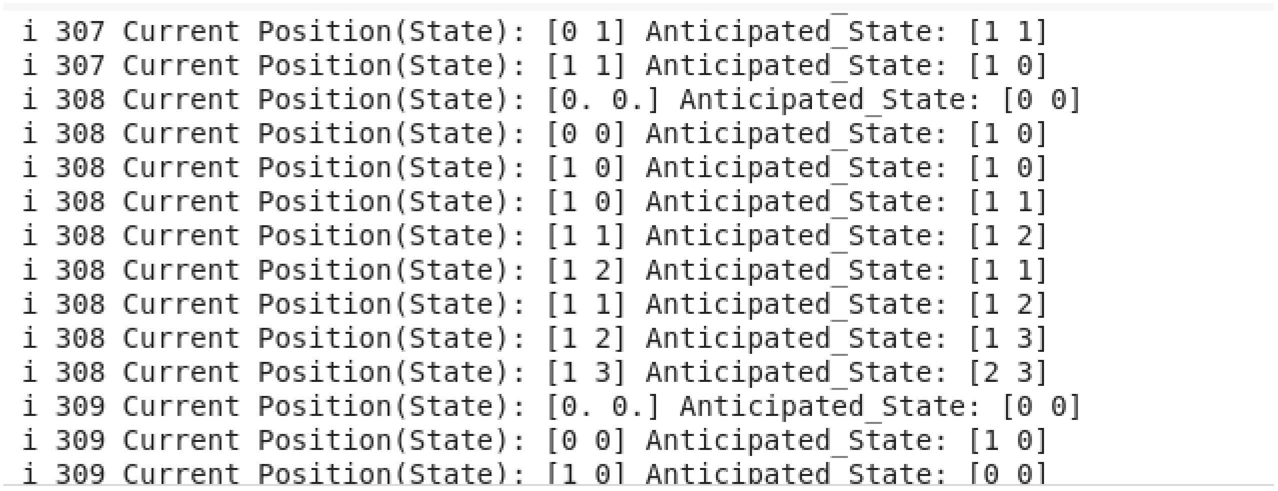
Results:

1. The Agents current state and anticipated state is printed during the training data. It can be seen in that from print values that the robot is initially exploring different path and taking longer routes. But it reaches the goal eventually after training for sufficient episodes.



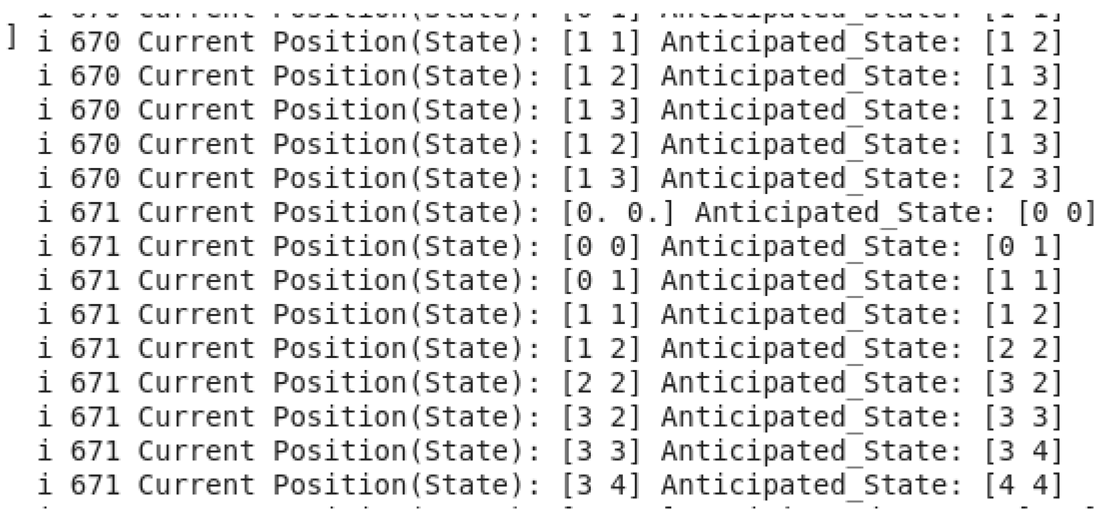
```
i 0 Current Position(State): [0. 0.] Anticipated_State: [0 1]
i 0 Current Position(State): [0 1] Anticipated_State: [0 1]
i 0 Current Position(State): [0 1] Anticipated_State: [0 1]
i 0 Current Position(State): [0 1] Anticipated_State: [0 2]
i 0 Current Position(State): [0 2] Anticipated_State: [1 2]
i 0 Current Position(State): [1 2] Anticipated_State: [1 3]
i 0 Current Position(State): [1 3] Anticipated_State: [1 2]
i 0 Current Position(State): [1 2] Anticipated_State: [1 1]
i 0 Current Position(State): [1 1] Anticipated_State: [2 1]
i 1 Current Position(State): [0. 0.] Anticipated_State: [0 0]
i 1 Current Position(State): [0 0] Anticipated_State: [0 1]
i 1 Current Position(State): [0 1] Anticipated_State: [0 1]
i 1 Current Position(State): [0 1] Anticipated_State: [1 1]
```

Img. 5 Initialy when the training is started the agent is in exploration stage.Does not reaches the goal



```
i 307 Current Position(State): [0 1] Anticipated_State: [1 1]
i 307 Current Position(State): [1 1] Anticipated_State: [1 0]
i 308 Current Position(State): [0. 0.] Anticipated_State: [0 0]
i 308 Current Position(State): [0 0] Anticipated_State: [1 0]
i 308 Current Position(State): [1 0] Anticipated_State: [1 0]
i 308 Current Position(State): [1 0] Anticipated_State: [1 1]
i 308 Current Position(State): [1 1] Anticipated_State: [1 2]
i 308 Current Position(State): [1 2] Anticipated_State: [1 1]
i 308 Current Position(State): [1 1] Anticipated_State: [1 2]
i 308 Current Position(State): [1 2] Anticipated_State: [1 3]
i 308 Current Position(State): [1 3] Anticipated_State: [2 3]
i 309 Current Position(State): [0. 0.] Anticipated_State: [0 0]
i 309 Current Position(State): [0 0] Anticipated_State: [1 0]
i 309 Current Position(State): [1 0] Anticipated_State: [0 0]
```

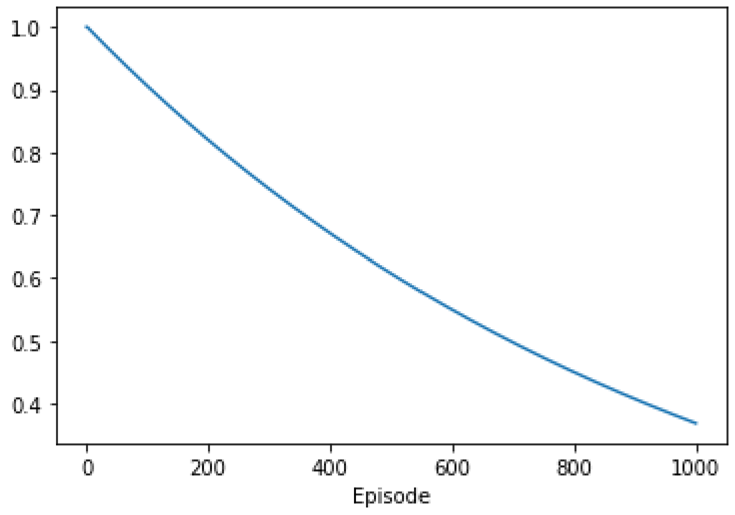
Img.6: After 300 iterations it figures out the different paths to reach the goal



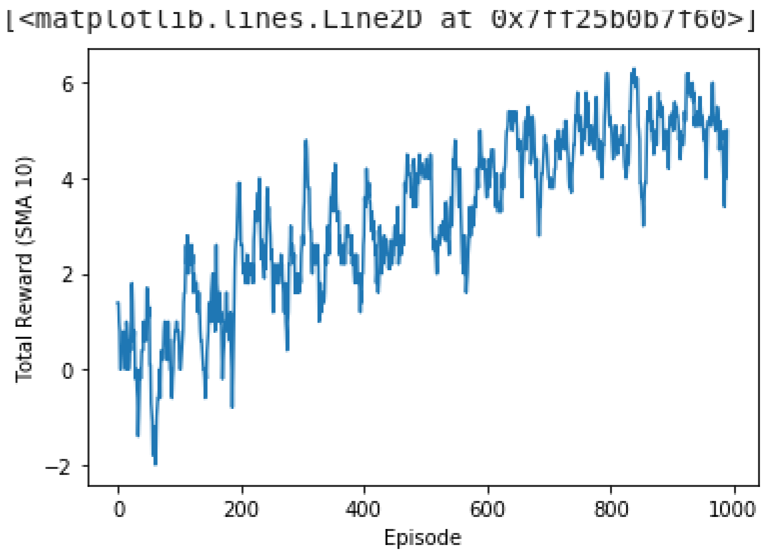
```
] i 670 Current Position(State): [1 1] Anticipated_State: [1 2]
  i 670 Current Position(State): [1 2] Anticipated_State: [1 3]
  i 670 Current Position(State): [1 3] Anticipated_State: [1 2]
  i 670 Current Position(State): [1 2] Anticipated_State: [1 3]
  i 670 Current Position(State): [1 3] Anticipated_State: [2 3]
  i 671 Current Position(State): [0. 0.] Anticipated_State: [0 0]
  i 671 Current Position(State): [0 0] Anticipated_State: [0 1]
  i 671 Current Position(State): [0 1] Anticipated_State: [1 1]
  i 671 Current Position(State): [1 1] Anticipated_State: [1 2]
  i 671 Current Position(State): [1 2] Anticipated_State: [2 2]
  i 671 Current Position(State): [2 2] Anticipated_State: [3 2]
  i 671 Current Position(State): [3 2] Anticipated_State: [3 3]
  i 671 Current Position(State): [3 3] Anticipated_State: [3 4]
  i 671 Current Position(State): [3 4] Anticipated_State: [4 4]
```

Img7: After 600 iterations it starts moving to the goal very frequently.

2. The Exponential Decay of the epsilon as mentioned in the above algorithm can be observed after plotting the graph Episodes(1000) vs Epsilon.



3. The Rewards are given to the agent at every step. The Rewards can be negative for wrong action and positive for the correct action. The Rewards are plotted against the number of episodes(1000).

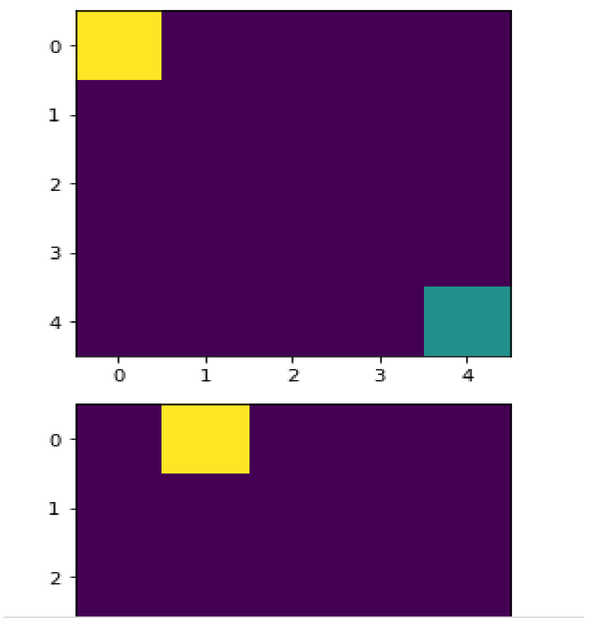


```
[<matplotlib.lines.Line2D at 0x7ff25b0b7f60>]
```

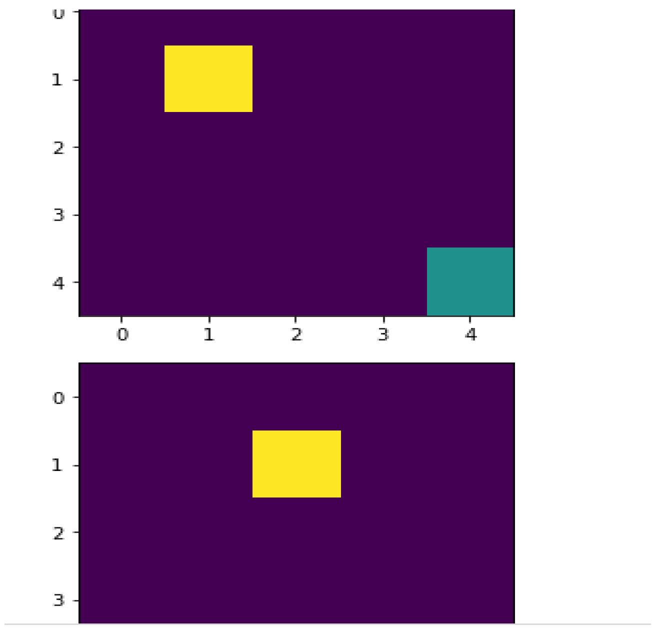
Img 9: Reward vs Episode.

4. Rewards can be seen in the graph above, initially the graph is have higher dips meaning the robot making mistakes and learning from it. The Frequency of good rewards observed increasing over the episodes.

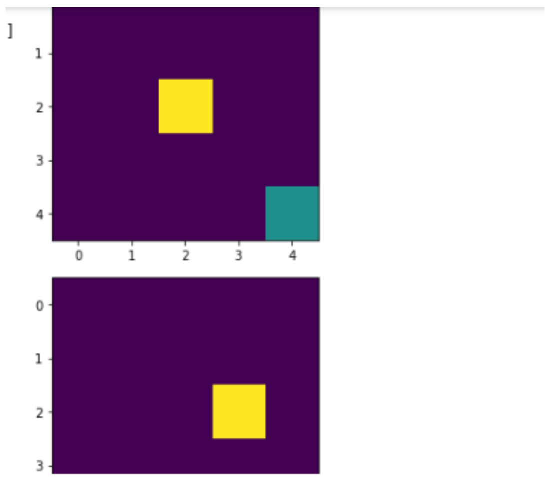
The Final Path:
The final path of the robot can be seen in the images below:



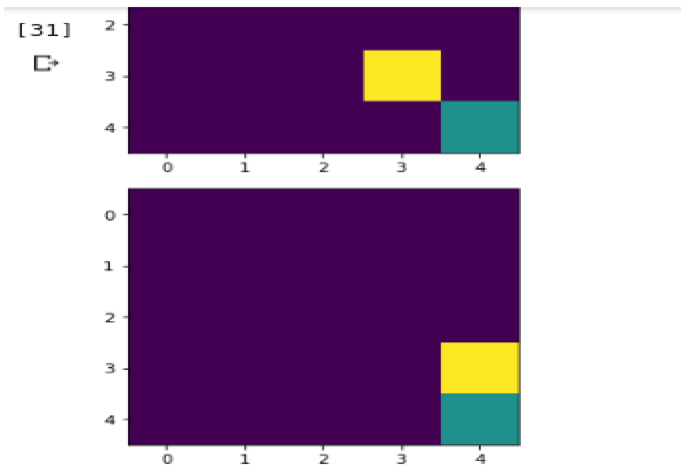
Stage: 1: 2:



Stage: 3:4:

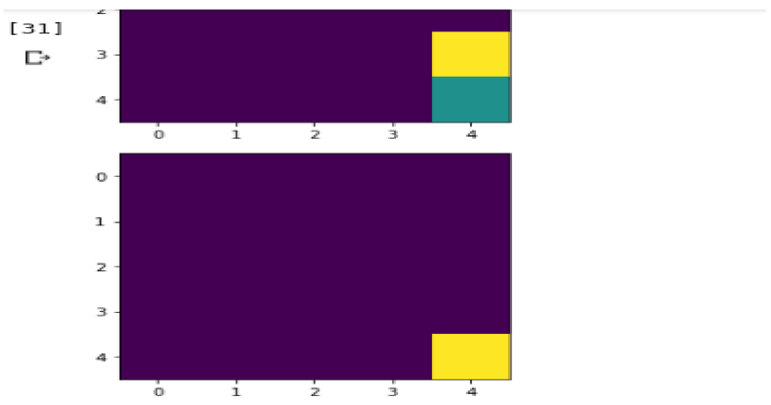


Stage: 4: 5:



Stage: 5: 6:

Reached



Conclusion:

It will be easy to conclude that the Reinforcement Learning is a very powerful too to implement the planning and optimization algorithms and can teach the agent to make decisions based on the Markov Decision Algorithm. The QL Policy teach us to take into account the current and the future stage as well so that the agent can make better decisions considering the all possible ways.