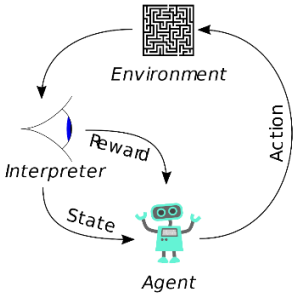
Reinforcement Q-Learning

Akash Kumar Roy
akashroy@buffalo.edu
Person Number: 50316991
University at Buffalo, State University of New York

Abstract: This is a report for Q-Learning algorithm performed on a provided OpenAI Gym Environment. Here The agent will learn an optimal policy through Q-Learning which will allow it to take actions to reach a goal while avoiding obstacles. First it will start random then it will improve its action by looking at the Q-Table and the agent will perform the action that will eventually provide the highest cumulative reward Here reward is the inverse of distance something to represent how close we are, instead of how far we are from the target. Also, going off the grid here means we have lost and so we should be penalized.

1. Introduction: Reinforcement learning is the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment. In reinforcement learning, an artificial intelligence faces a game-like situation. The algorithm employs trial and error to come up with a solution to the problem. To get the machine to do what the programmer wants, the artificial intelligence gets either rewards or penalties for the actions it performs. Its goal is to maximize the total reward. Q-Learning is a part of reinforcement learning. The goal of Q-learning is to learn a policy, which tells an agent what action to take under what circumstances. It does not require a model of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations. To sum it up the Q-Learning Algorithm State Diagram Looks like following



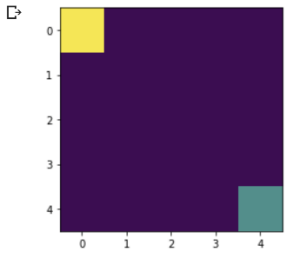
2. Dataset Definition:

Reinforcement learning can be understood as a tuple of agents, environments, states, actions and rewards. Each of the key components are described below:

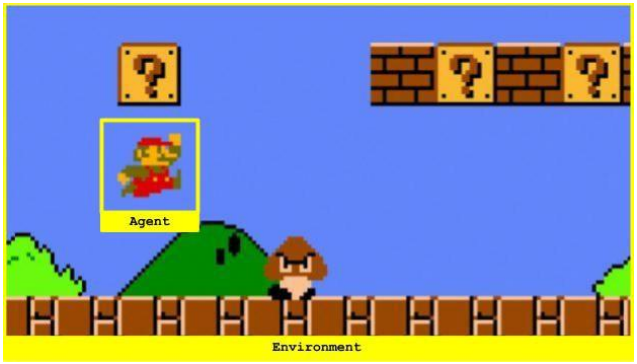
Environment: Environment is the Physical world in which the agent operates. An environment represents the outside world to the agent and an agent that takes actions, receives observations from the environment that consists of a reward for his action and information of his new state.

The environment provided for this project is OpenAI Gym Environment. Gym is a toolkit for developing and comparing reinforcement learning algorithms. It makes no assumptions about the structure of our agent, and is compatible with any numerical computation library, such as TensorFlow or Theano. The gym library

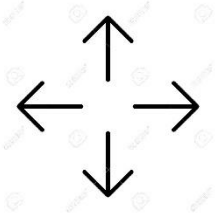
is a collection of test problems and environments that we can use to work out our reinforcement learning algorithms. The 5*5 provided grid environment looks like the following image-



Agents: An agent takes actions; for example, a drone making a delivery, or Super Mario navigating a video game. The algorithm is the agent.



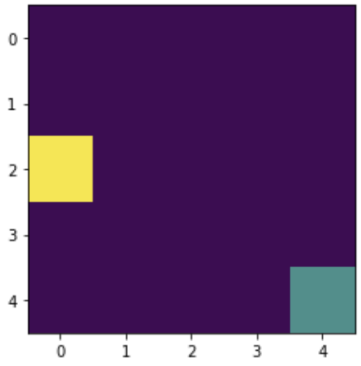
Action: Action is the set of all possible moves the agent can make. An action is almost self-explanatory, but it should be noted that agents usually choose from a list of discrete, possible actions. In the given environment the agent will work within an action space consisting of four actions: up, down, left, right.



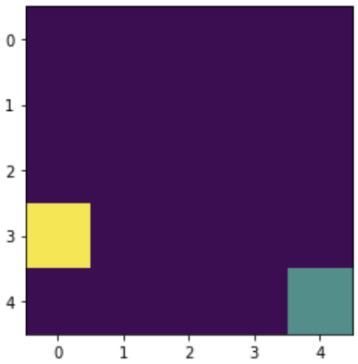
All Possible actions for the agent

States: A state is a concrete and immediate situation in which the agent finds itself; i.e. a specific place and moment, an instantaneous configuration that puts the agent in relation to other significant things such as tools, obstacles, enemies or prizes. It can the current situation returned by the environment, or any future situation. Suppose in a 5x5 grid the agent is at (2,0) the means the agent is at second row and first column, so current state of the agent is (2,0). Any action the agent takes towards maximizing the goal will lead to the agent moving from current state (2,0) to any other state that is possible. (1,0), (2,1), (3,0) Once the agent takes its action the new position will be its current state and (2,0) will be its previous state. The following images explain the state property

Previous State



Next State



Reward: A reward is the feedback by which we measure the success or failure of an agent's actions in a given state. From any given state, an agent sends output in the form of actions to the environment, and the environment returns the agent's new state (which resulted from acting on the previous state) as well as rewards and that effectively evaluate the agent's action. Here 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

3. Preprocessing: Since a simple environment and a framework was provided for training I did not have to do any preprocessing for this project.

4. Architecture:

Q-Learning: The 'Q' in Q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward. Q-learning is an off-policy reinforcement learning algorithm that seeks to find the best action to take given the current state. At any given state, Q-Learning will perform the action that will eventually yield the highest cumulative reward. So, this is a greedy algorithm. More specifically, Q-learning seeks to learn a policy that maximizes the total reward.

Equations

Mapping Function: $Q\theta : S \times A \rightarrow R$,

Policy:

$$\pi(s_t) = \underset{a \in A}{\operatorname{argmax}}\, Q_\theta(s_t, a)$$

The mapping function will tell us which action will lead to which expected cumulative discounted reward, and our policy π will choose an action that will lead to the maximum such value.

Updating Equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big(R_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)\big)$$

we take the reward we got from our selected action (R_{t+1}), and add it to the discounted maximum Q-value for that state, and subtract it from our current estimate of the Q-value of that state. What this is doing is computing the error between the action we just took and what we believe to be the best action available to us from this new state (S_{t+1}). We then scale the error down by our step-size (α) and add it to our current estimate for Q(st,at)which then yields our new estimate for Q(st,at). Q(st,at) is an approximation of the total discounted future rewards from that state when taking that action. This means that when we take that action and get the reward, the reward plus our approximation should be equal to the discounted sum of all future rewards if we're following a greedy policy. In other words, we want to learn a function so that $Q(s_t, a_t) \approx R_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1})$

In the equations there is two hyperparameters they are alpha and gamma. Other than this there is another hyperparameter that is epsilon. **The description of the hyperparameters are as following**:

- **Alpha** is the learning rate.
- **Gamma** is the discount factor. It quantifies how much importance we give for future rewards. It's also handy to approximate the noise in future rewards. Gamma varies from 0 to 1. If Gamma is closer to zero, the agent will tend to consider only immediate rewards. If Gamma is closer to one, the agent will consider future rewards with greater weight, willing to delay the reward.
- **Epsilon** is exploration rate. The agent takes random actions for probability **ε** and greedy action for probability (1-**ε**). In the beginning, epsilon must be at its highest value, because we don't know anything about the values in Q-table. This means we need to do a lot of exploration, by randomly choosing our actions. We will reduce it progressively as the agent becomes more confident at estimating Q-values.

Q-Table: Q-Table is a table for simple lookup table where we calculate the maximum expected future rewards for action at each state. After each stage is performed we create a q-table or matrix that follows the shape of [state, action] and we initialize our values to zero. We then update and store our q-values after an episode. This q-table becomes a reference table for our agent to select the best action based on the q-value. I have provided the Q-Tables in the result section.

Q-Learning Algorithm:

Q-learning(S,A,γ,α)

Inputs

S is a set of states
A is a set of actions
γ the discount
α is the step size

Local

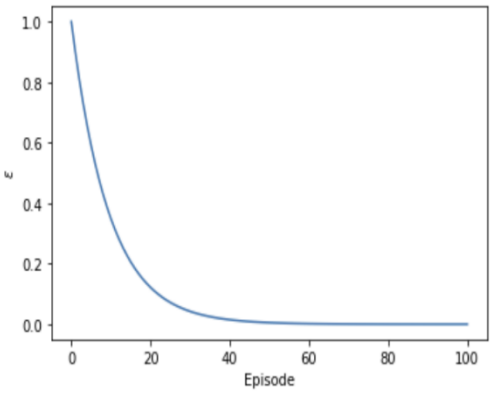
real array Q[S,A]
previous state s
previous action a
initialize Q[S,A] arbitrarily
observe current state s
repeat
select and carry out an action a
observe reward r and state s'
$Q[s,a] \leftarrow Q[s,a] + \alpha(r + \gamma max_{a'} Q[s',a'] - Q[s,a])$
$s \leftarrow s'$
until termination

5.Result:

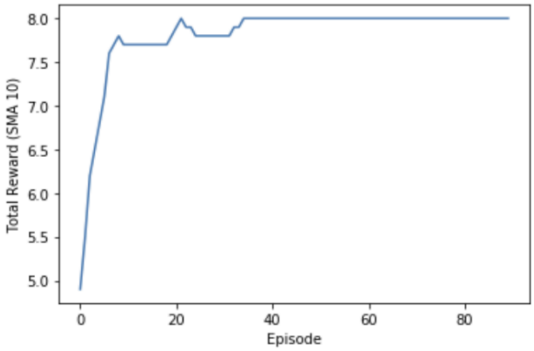
First Observation

Episode	Decay
100	0.9

Graphs:



Epsilon vs Episodes



Total Rewards vs Episodes

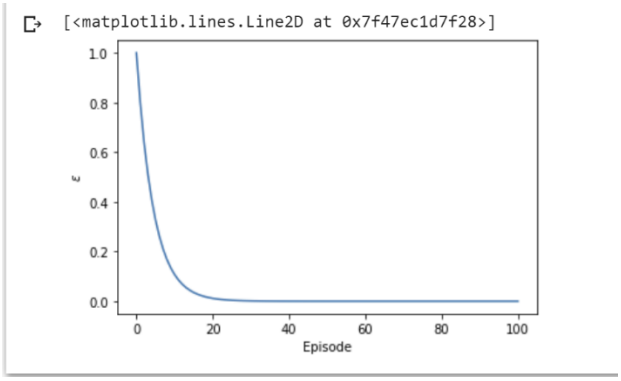
Q-Table:

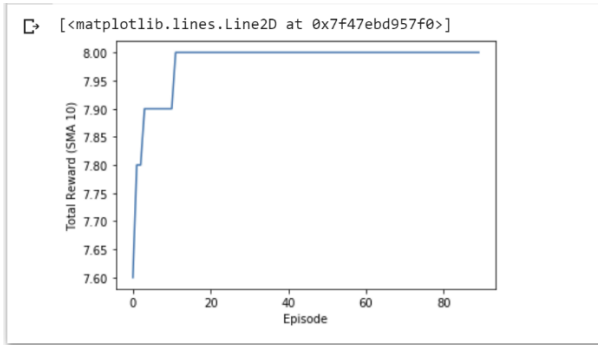
```
[ ]  print(agent.q_table)

[[[ 2.88100000e-01 -1.11595834e-01  5.28157000e+00 -3.46164076e-03]
  [ 3.08909931e-01 -1.65142506e-01  4.95418761e+00 -4.73136307e-04]
  [ 4.53773853e+00 -3.01291480e-01  1.99810000e-01 -1.59213933e-02]
  [ 2.64232808e-01  0.00000000e+00  1.00000000e-01 -1.23341609e-01]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -9.10000000e-02]]

 [[ 1.00000000e-01 -6.68539000e-02  2.08711900e-01 -1.90000000e-01]
  [ 1.00000000e-01  0.00000000e+00  5.07813912e-01 -9.10000000e-02]
  [ 1.09000000e-01  2.65390629e-01  4.02391383e+00 -1.71288100e-01]
  [ 1.00000000e-01 -1.80190000e-01  3.41090079e+00 -4.85901189e-02]
  [ 2.70144471e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]

 [[ 1.90000000e-01  0.00000000e+00  1.00000000e-01  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -8.29000000e-02]
  [ 0.00000000e+00  0.00000000e+00  1.90000000e-01  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  1.31761000e-01 -1.00000000e-01]
  [ 1.89820519e+00 -7.40710000e-02 -2.38510000e-01 -1.00000000e-01]]

 [[ 0.00000000e+00 -9.10000000e-02  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 9.99803373e-01  0.00000000e+00 -1.00000000e-01 -1.00000000e-01]]

 [[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]]
```

Second Observation

Episode	Decay
150	0.85

Graphs:



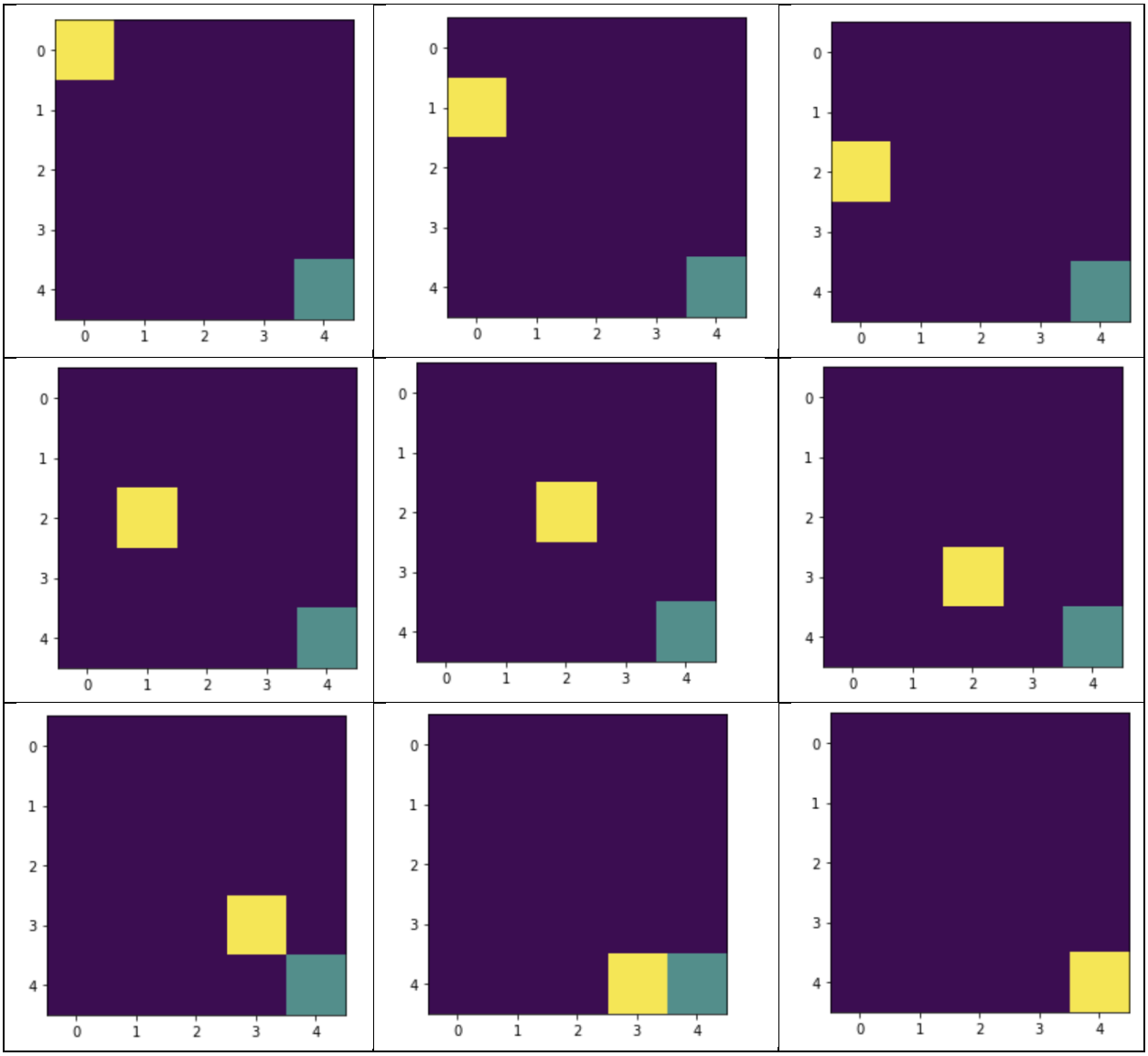


Epsilon vs Episodes **Total Reward vs Episode**

Q-Table:

```
[[[ 0.30011627 -0.33807487  5.19728262 -0.1637794 ]
  [ 0.199      -0.15396632  4.89399748 -0.10249878]
  [ 4.50004599  0.          0.2889829   0.03714792]
  [ 0.         -0.1         0.3952063  -0.08209   ]
  [ 0.43980201  0.          0.          0.        ]]

 [[ 0.         -0.0748      0.33865048  0.        ]
  [ 0.4895389   0.          0.         -0.09019   ]
  [ 0.4616182  -0.01078332  4.00360277 -0.08209   ]
  [ 0.199      -0.21267197  3.40188359 -0.05143409]
  [ 2.69840857  0.         -0.17139117 -0.1729    ]]

 [[ 0.          0.          0.          0.        ]
  [ 0.43552     0.          0.19        0.        ]
  [ 0.         -0.091       0.41650531 -0.07561   ]
  [ 0.         -0.073342    0.63211641 -0.1       ]
  [ 1.8975276  -0.0829     -0.14818522 -0.28742048]]

 [[ 0.          0.          0.          0.        ]
  [ 0.2881     -0.0829      0.          0.        ]
  [ 0.          0.          0.          0.        ]
  [ 0.          0.          0.         -0.1       ]
  [ 0.99973028 -0.07561    -0.1        -0.1       ]]

 [[ 0.          0.          0.          0.        ]
  [-0.091      -0.08209     0.19        0.        ]
  [ 0.          0.          0.1         0.        ]
  [ 0.          0.          0.         -0.091     ]
  [ 0.          0.          0.          0.        ]]]
```

Our agent will work like below after training. It will start from the top left position to the bottom right position which is the target. In the following images the yellow square represents the position of the agent and the Greenish Blue square at the bottom is the goal.



Challenges:

- The main challenge of Q-learning is that the learning process is expensive for the agent, specially, in the beginning steps. Because, every state-action pair should be visited frequently in order to converge to the optimal policy.
- If the number of states in the environment are very high, it becomes difficult to implement them with Q table as the size would become very, very large.

6. Conclusion: After Observing the output of the Q-Learning Algorithm we can see that how the agent improves its performance by looking at the Q-table and then decide its next step depends on the reward value. Reinforcement learning has given solutions to many problems from a wide variety of different domains. Most popular example will be is Google's NasNet which uses *deep reinforcement learning* for finding an optimal neural network architecture.

7. References

- Lecture Slides
- Wikipedia
- https://towardsdatascience.com/q-learning-54b841f3f9e4
- https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/