PROJECT 4 REINFORCEMENT LEARNING

Prakhathi Murugesan
University of Buffalo
prakhath@buffalo.edu

Abstract

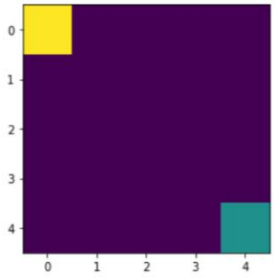
Reinforcement Learning is a type of machine learning when there is no training data, or there is no enough expertise about the problem at hand. It is where the agent learns to perform a specific action in an environment to achieve the maximum reward. The goal of the project is to train the agent to navigate through the classic grid-world environment and reach the goal and achieve the maximum reward. The agent is trained using the Q-Learning algorithm, which allows it to take optimal action to reach the goal. The environment and the agent are built using OpenAI Gym, and the Q-table is updated for each action, and the rewards are recorded.

1. INTRODUCTION:

In Reinforcement Learning, there is no training dataset; the agent learns from its experience, both bad and good, and decides how to act to reach its goal. It learns through the trial-and-error method as it attempts its task, with the goal of maximizing the reward. This trial-and-error method accompanies a reward mechanism, which means for each action, the reward will be provided to the agent, and it learns from it. Reinforcement Learning is used in a lot of interesting applications like game theory and multi-agent interaction, simulation-based optimization, robotics, vehicle navigation, industrial logistics, etc.

2. ENVIRONMENT:

Reinforcement learning environments can take on many different forms. The reinforcement learning community (and, specifically, OpenAI) has developed a standard of how such environments should be designed, and the library which facilitates this is OpenAI's Gym. The environment used in this project is an essential deterministic n × n grid-world environment 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 be rewarded with +1 for moving closer to the goal and −1 for moving away or remaining the same distance from the goal.



The initial state of our basic grid-world environment.

3. MARKOV DECISION PROCESS:

Reinforcement Learning is usually modeled as a Markov decision process (MDP), which uses dynamic programming techniques that is the agent receives a delayed reward in the next step to evaluate its previous action. Reinforcement learning solves the problem of correlating immediate actions with the delayed returns they produce. They operate in a delayed return environment, where it can be difficult to understand which action leads to which outcome over many time steps. Reinforcement learning algorithms can be expected to perform better and better in more ambiguous, real-life environments when choosing from an arbitrary number of possible actions, rather than from the limited options. It involves the following terminologies:

- Environment(e) – A scenario where the agent has to act on.
- Agent – A hypothetical entity which performs actions in an environment to gain some reward.
- Action(a) – All the possible moves that the agent can take.
- State(s) – The current situation returned by the environment.
- Reward(R) – An immediate return sent back from the environment to evaluate the last action by the agent
- Policy(π) – The strategy that the agent employs to determine the next action based on the current state.
- Q-value – It takes a parameter, the current action **a**. **Qπ(s, a)** refers to the long-term return of the current state **s**, taking action **an** under policy **π**.
- Discount factor(γ) – Ranges from 0 to 1. The factor quantifies the difference in importance between immediate rewards and future rewards.
- Episode – The number of iteration that the agent plays/trains.

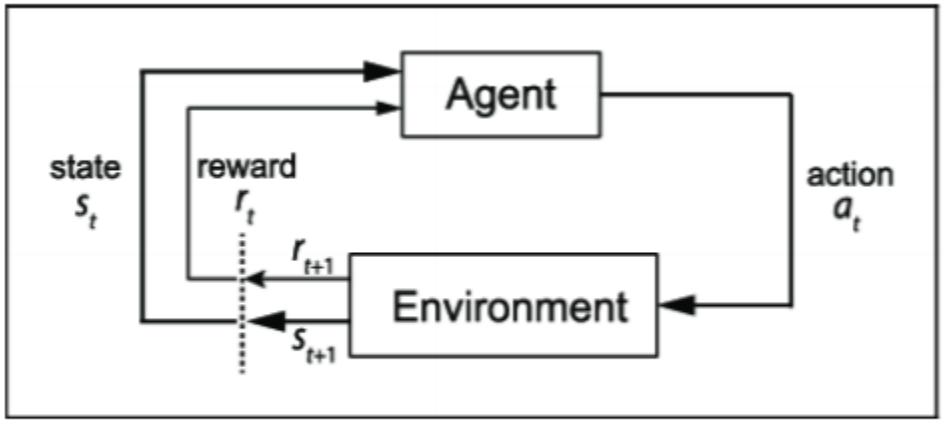


Figure 1: The canonical MDP diagram[1]

An MDP is a 4-tuple (S, A, P, R), where

- S is the set of all possible states for the environment.
- A is the set of all possible actions the agent can take.
- P = P r(st+1 = s 0 |st = s, at = a) is the state transition probability function.
- R : S × A × S → R is the reward function.

Our task is find a policy π : S → A which our agent will use to take actions in the environment that maximizes cumulative reward,

$$\sum_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1})$$

where $\gamma \in [0, 1]$ is a discounting factor (used to give more weight to more immediate rewards), 'st' is the state at time step t, 'at' is the action the agent took at time step t, and 'st+1' is the state which the environment transitioned to after the agent took the action.

4. Q-LEARNING ALGORITHM:

Q-Learning is a reinforcement learning algorithm that aims to find the best action to take, given the current state. Q-learning seeks to learn a policy that maximizes the total reward. Q denotes the quality, which represents how useful a given action is in gaining some future reward. Key factors of Q-learning are:

- Policy
- Q-table

EPSILON-GREEDY STRATEGY:

Reinforcement Learning involves two significant factors, namely, exploration and exploitation. *Exploitation* means the agent keeps doing what it was doing, and *exploration means* the agent tries something new. The rate of these should be monitored because the problem in exploration is that the agent doesn't know the outcome; it could be either better than the current situation or could be worse. If the agent keeps exploiting the known knowledge, it would confine to the same environment and satisfied with the small reward without even trying for the huge reward or ultimate goal. One simple strategy for exploration would be for the agent to take the best-known action most of the time but occasionally explore a new, randomly selected direction even though it might be walking away from the known reward. This strategy is called the Epsilon-greedy strategy, where ***epsilon*** is the percent of the time that the agent takes a randomly selected action rather than taking the action that is most likely to maximize reward given what it knows so far. We usually start with a lot of exploration (i.e., a higher value for epsilon). Over time, as the agent learns more about the environment and which actions yield the most long-term reward, it would make sense to steadily reduce epsilon to 10% or even lower as it settles into exploiting what it knows.

POLICY:

When the agent interacts with the environment, the action that the agent has to take is given by the policy. For selecting the action, the agent can either explore the uncharted territory hoping to get the best reward or exploit the known information and select the action based on the max future reward. We have to balance the exploration/exploitation rate using epsilon and to set the value of how often the agent should explore/exploit.

- The agent will randomly select its action at first by a certain percentage, called 'exploration rate' or 'epsilon.' This is because at first, the agent should try all kinds of things before it starts to see the patterns. If the randomly selected uniform number is less than epsilon, then the agent selects the random choice of action.
- When it is doesn't decide the action randomly, the agent will predict the reward value based on the current state and pick the action that gives the highest reward.

  $\pi(st)$=argmax$a \in AQ\theta(st,a)$

Q-TABLE:

While performing Q-learning, a matrix is created called Q-table, which is of the shape [state, action], and we initialize the values to zero. After each step of every episode, the Q-table is updated. This table is used as the reference table of the agent to select the best action based on the q-value. The Q-Table, which would have entries Qi,j where I corresponds to the ith state (the row), and j corresponds to the jth action (the column) so that if S is located in the ith row, and A is the jth column, Q(S, A) = Qi,j . We use a value iteration update algorithm to update our Q-values as we explore the environment's states:

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

Update rule for Q-Learning

- **Learning rate** (**alpha)**: This is how much the agent accepts the new value vs. the old value. In the above rule, the difference between new and old is taken and then multiplied that value by the learning rate. This value then gets added to the previous q-value which essentially moves it in the direction of the latest update
- **Gamma (*γ*)***: It* is a discount factor. It's used to balance immediate and future rewards. From the above update rule, it is clear that the discount is applied to the future reward. Typically this value can range anywhere from 0.8 to 0.99.
- **Reward:** It is the value received after completing a certain action at a given state. A reward can happen at any given time step or only at the terminal time step.
- **Max**: np.max() uses the numpy library and is taking the maximum of the future reward and applying it to the reward for the current state. It impacts the current action by the possible future reward. This is the beauty of q-learning. We're allocating future rewards to current actions to help the agent select the highest return action at any given state.

Basic steps followed in Q-learning:

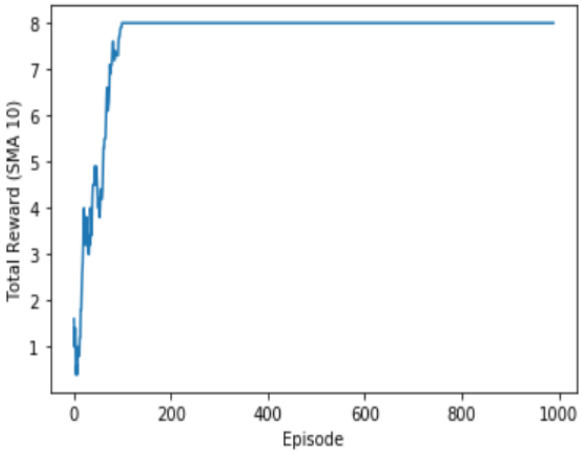
- Agent starts with a state, takes action and receives a reward.
- Agent selects an action by referencing Q-table with the highest value (max) or by random choice(epsilon).
- Update Q-table.
- Repeat the process until the episode terminates, i.e., when the agent reaches the endpoint/goal.
- Repeat the same process for the total number of episodes with the state reset back to the initial state.

We can observe that the agent gets trained and started achieving more rewards even after the few episodes. After each step, we decrease the exploration rate using exploration decay. After each episode, the reward obtained is used to plot the graph, and also the epsilon/exploration rate is plotted.

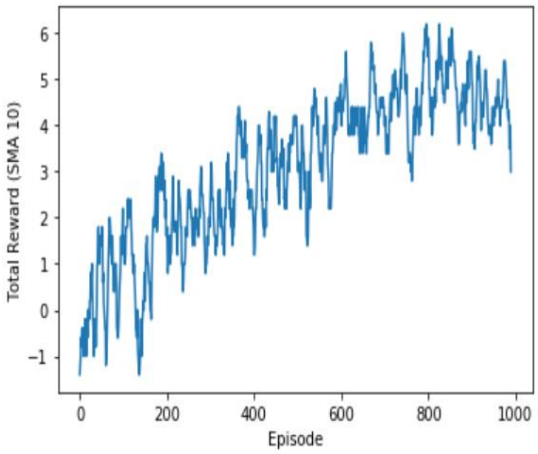
5. CHALLENGES:

The experiment can be done with varying the various parameters such as exploration rate(ε), learning rate(γ), and exploration decay rate(δ). This makes the agent choose different actions, which in turn gives a different reward. These are used for potting the graph, and observation is made for the same. The challenges faced were to find the

optimal rate at which the exploration rate decreases. The optimal method, which decreases the exploration, gives the optimal solution and maximum reward.



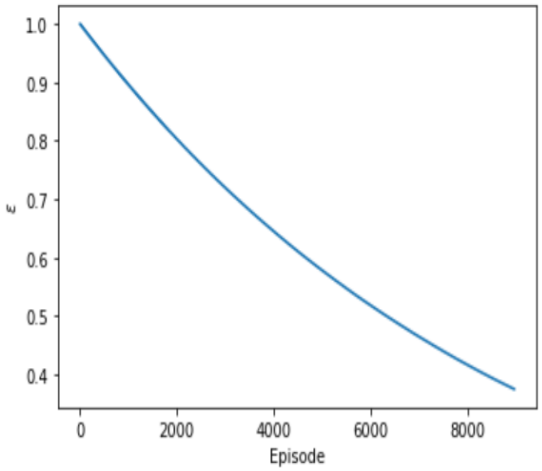
when εis decreased by subtracting it δ

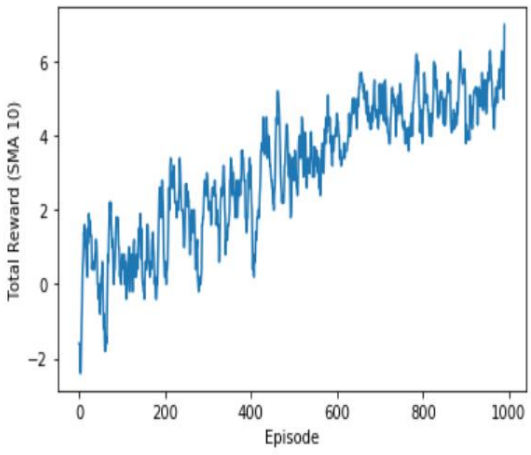


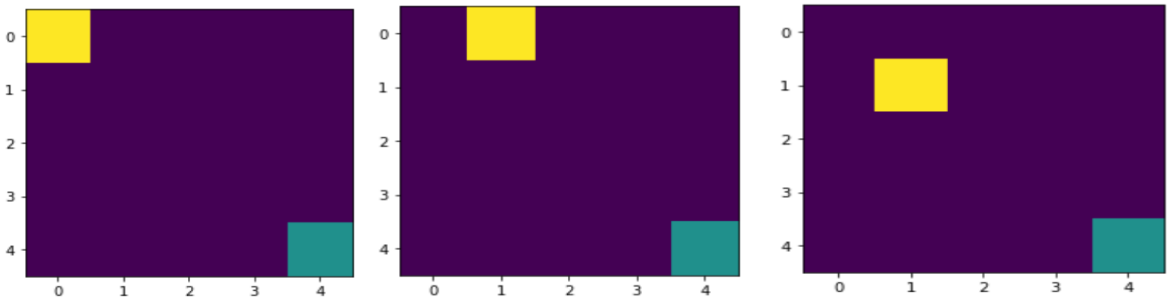
when learning rate(γ) is 0.1

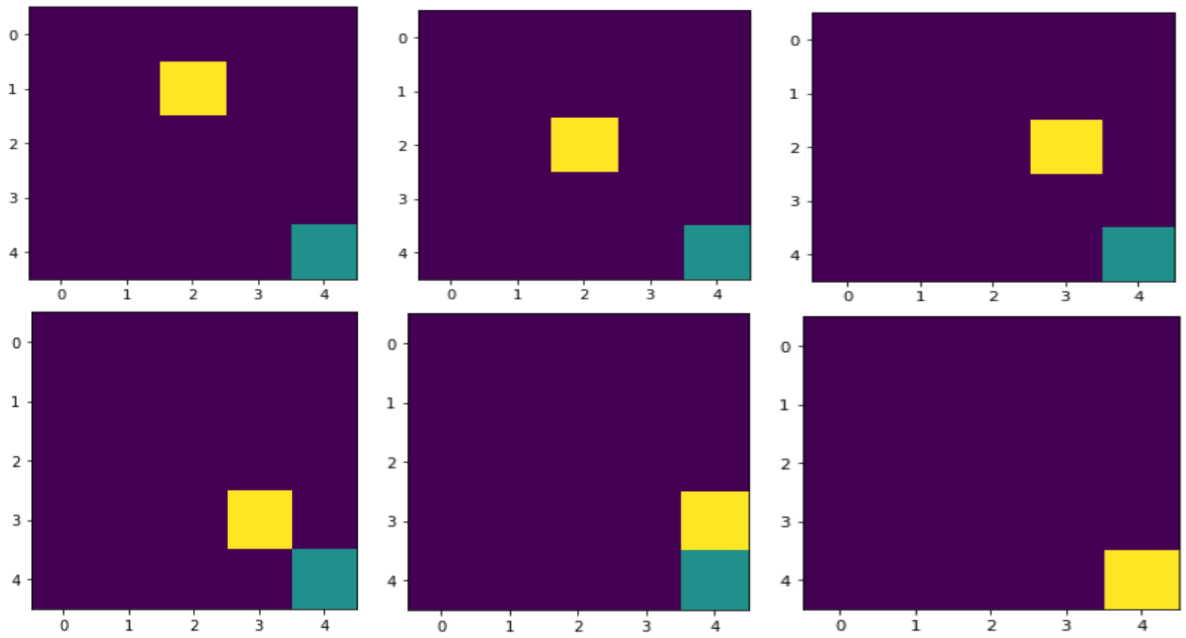
6. RESULTS:

The graph is plotted for rewards obtained per episode, and it is clear that the graph is non-increasing, which denotes that the agent learns from its past actions and improves its course of action to achieve the maximum reward. The graph is plotted for Epsilon Vs. The episode, and it is clear that the exploration rate is maximum at the beginning of the play as the agent gets rewards and familiarizes with the environment, the exploration rate decreases. Thus, the agent reaches the goal and achieves the maximum reward.









7. CONCLUSION:

This project helps get a clear understanding of Reinforcement Learning. It has also helped in understanding the Q-Learning algorithm. We trained the agent to take the optimal action at each step by updating the Q-table. The agent gets trained well by using policy and the q-learning algorithm. Thus, we trained an agent to reach the goal with maximum reward using reinforcement learning.

REFERENCES

[1] Introduction to Reinforcement Learning- https://medium.com/machine-learning-for-humans/reinforcement-learning-6eacf258b265

[2] Fundamentals of Reinforcement Learning – https://www.datacamp.com/community/tutorials/introduction-reinforcement-learning

[3] https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56