REINFORCEMENT LEARNING – Q LEARNING

Akshara Santharam
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
aksharas@buffalo.edu

Abstract

Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment to maximize some notion of cumulative rewards. The aim of this project is to build a reinforcement learning agent to navigate the classic 4x4 grid-world environment. In this project, the agent will learn an optimal policy through Q-Learning which will allow it to take actions to reach a goal while avoiding obstacles. The main task is to implement and explain key components of the Q-Learning algorithm. Specifically, tabular Q-Learning approach is used which utilizes a table of Q-Values as the agent's policy.

1 Introduction

Reinforcement Learning is an area of machine learning, which trains the machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain complex environment. Reinforcement learning, known as a semi-supervised learning model, is a technique to allow an agent to take actions and interact with an environment to maximize the total rewards. It is modeled as a Markov Decision Process (MDP).

Reinforcement learning refers to goal-oriented algorithms, which learn how to attain a complex objective or maximize along a dimension over many steps. It is employed by various software and machines to find the best possible behavior or path to be taken in a specific situation. Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it, so the model is trained with the correct answer but in reinforcement learning, there is no answer and decides what to do to perform the given task. It is bound to learn from its experience in the absence of training dataset.

The main components involved in Reinforcement learning is Input, Output, Training. Input: The input should be an initial state from which the model will start. Output: There are many possible outputs as there are variety of solution to a particular problem. Training: The training is based upon the input. The model will return a state and the user will decide to reward or punish the model based on its output. The model continues to learn, and the best solution is decided based on the maximum reward.

The different types of Reinforcement Learning are Positive and Negative Reinforcement Learning. Positive Reinforcement is defined as when an event occurs due to a particular behavior, it increases the strength and the frequency of the behavior. The effect on the behavior is positive. The advantage of reinforcement learning is that it maximizes performance and it sustain changes for a long period of time. Disadvantages of reinforcement learning is that too much reinforcement can lead to overload of states which can diminish the results. Negative Reinforcement is defined as strengthening of a behavior since negative condition is stopped. The advantage is that it increases behavior and provide defiance to minimum standard of performance. Disadvantage is that only provides enough to meet up the minimum behavior.

Applications of Reinforcement Learning:

- Reinforcement Learning can be used in robotics for industrial automation.
- Reinforcement Learning can be used to create training systems that provide custom instruction and materials according to the requirement of students.
- Reinforcement Learning can be used in machine learning and data processing.

2 Reinforcement Learning

Reinforcement Learning is a machine learning paradigm which focuses on how automated agents can learn to take actions in response to the current state of an environment so as to maximize some reward. This is typically modeled as a Markov Decision Process (MDP).

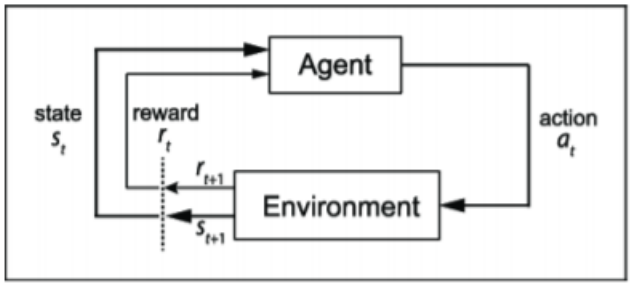

Fig 1: Canonical MDP diagram [1]

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 possible actions the agent can take
- $P = P_r(s_{t+1}=s' \mid s_t = s, a_t = a)$ is the state transition probability function
- R: S x A x S ->R is the reward function

The task is to find a policy π: S → A which our agent will use to take actions in the environment which maximize cumulative reward i.e.,

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

where $\gamma \in [0, 1]$ is a discounting factor, s_t is the state at a time step t, a_t is the action the agent took at time step t, and s_{t+1} is the state which environment transitioned to after the agent took the action.

3 Q-Learning

Q-Learning is a process in which we train some function $Q_\theta : S \times A \rightarrow R$, parameterized by θ, to learn a mapping from state-action pairs to their Q-value, which is the expected discounted reward for following the following policy π_θ:

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

In words, the function Q_θ 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 s_t. Originally, Q-Learning was done in a tabular fashion. Here, we would create an $|S| \times |A|$ array, our *Q-Table*, which would have entries $q_{i,j}$ where i corresponds to the ith state (the row) and j corresponds to the jth action (the column), so that if st is located in the ith row and at is the jth column, $Q(s_t, a_t) = q_{i,j}$. We use a value iteration update algorithm to update our Q-values as we explore the environment's states:

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

Fig 2: Update rule for Q-Learning [2]

Q-Function is used to recursively match to calculate the discounted cumulative total reward. The table is initialized the table with all 0 values and we explore the environment, collect our trajectories, [s_0, a_0, r_0, s_1, a_2, r_2, s_T, a_T, r_T], and use these values to update our corresponding entries in the Q-table.

During training, the agent will need to explore the environment to learn which actions will lead to maximal future discounted rewards. Agent's often begin exploring the environment by taking random actions at each state. Doing so, however, poses a problem: the agent may not reach states which lead to optimal rewards since they may not take the optimal sequence of actions to get there. To account for this, we will slowly encourage the agent to follow its policy to take actions it believes will lead to maximal rewards. This is called exploitation and striking a balance between exploration and exploitation is key to properly training an agent to learn to navigate an environment by itself.

To facilitate this, we have our agent follow what is called an $\in$ -greedy strategy. Here, we introduce a parameter $\in$ and set it initially to 1. At every training step, we decrease its value gradually until we've reached some predetermined minimal value. In this case, we are annealing the value of $\in$ linearly so that it follows a schedule. During each time step, we have our agent either choose an action according to its policy or take a random action by taking a sampling a single value on the uniform distribution over [0, 1] and selecting a random action if that sampled value is than otherwise taking an action following the agent's policy.

4 Environment

Reinforcement learning environments can take on many different forms, including physical simulations, video games, stock market simulations, etc. The reinforcement learning community has developed a standard of how such environments should be designed, and the library which facilitates this is OpenAI's Gym.

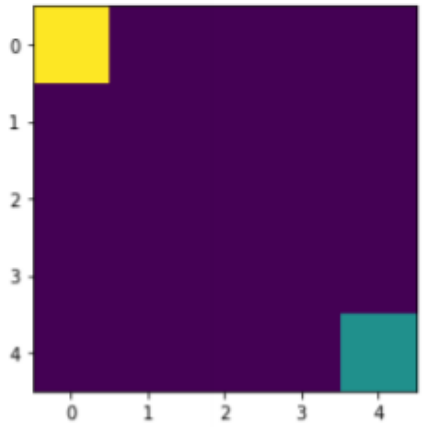

Fig 3: The initial state of our basic grid-world environment [3]

The environment we provide is a basic deterministic n × n grid-world environment where the agent has to reach the goal 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.

5 Q-Table

We initialize all the Q-values to zero for each state-action pair, when we construct the Q-table. The number of rows in the table is equivalent to the size of the state space in the environment and the number of columns is equivalent to the size of the action space. The information can be gathered using env.observation-space.n and env.action-space.n

action-space-size = env.action-space.n

state-space-size = env.observation-space.n

q-table = np.zeros((state-space-size, action-space-size))

Q-Table:

```
[[[ 5.60874615  4.08518892  5.68667622  4.0689436 ]
 [ 5.14617508  3.6336775   5.20868825  4.09082648]
 [ 4.67818987  3.17568403  4.61798109  3.64624995]
 [ 4.06969013  2.20035938  3.25234015  2.84529506]
 [ 3.27763284  0.27371464  0.65848501  1.01529385]]

 [[ 3.69534109  3.78013926  5.17631372  3.24633968]
 [ 3.57796358  3.44627276  4.67468708  3.4114846 ]
 [ 3.39658048  3.17718944  4.08871539  3.15018799]
 [ 3.29364359  2.56545977  3.43336945  2.64923814]
 [ 2.70532929  1.61459842  1.37817947  1.98412384]]

 [[ 1.73389821  2.39820843  3.35612558  0.91854027]
 [ 2.15001504  1.78189701  3.11415559  1.13852954]
 [ 2.60128116  2.14529568  2.94591878  1.31080914]
 [ 0.91753684  1.21723217  2.66874543  0.44197378]
 [ 1.89730096  1.08257035  0.54650935  0.95061092]]

 [[ 1.3585947   0.31033623  0.79448658 -0.30261365]
 [ 1.62469187  0.39296261  0.940978   -0.12557141]
 [ 0.79017017  0.44298953  1.91112437  0.11950275]
 [ 1.39940583 -0.1298954   0.54046701  0.17920214]
 [ 0.99854442  0.40472391 -0.22279543  0.00707968]]

 [[-0.22229862 -0.10627582  1.01695851 -0.08169863]
 [-0.37256795 -0.0308327   1.02746002 -0.32692224]
 [-0.36869571 -0.074071    0.8691903  -0.09492569]
 [-0.27527515 -0.08317626  0.74581342 -0.2319944 ]
 [ 0.          0.          0.          0.        ]]]
```

6 Q-Learning parameters

The parameters needed to implement the Q-Learning algorithm are num-episodes, max-steps-per-episode, learning-rate, discount-rate, exploration-rate, max-exploration-rate, min-exploration-rate, exploration-decay-rate.

- **num-episodes**: With num-episodes, we define the total number of episodes we want the agent to play during training.
- **max-steps-per-episode**: With max-steps-per-episode, we define a maximum number of steps that our agent can take within a single episode.
- **learning-rate**: The amount that the weights are updated during training is referred to as the step size or the "learning rate".
- **discount-rate**: Discount rate is a value between 0 and 1.

exploration-rate, max-exploration-rate, min-exploration-rate and exploration-decay-rate are related to the exploration-exploitation trade-off.

- **exploration-rate**: exploration-rate is set to 1.
- **max-exploration-rate**: max-exploration-rate is set to 1
- **min-exploration-rate**: min-exploration-rate is set to 0.01
- **exploration-decay-rate**: exploration-decay-rate is set to 0.01 to determine the rate at which the exploration-rate will decay.

7 Challenges

7.1 Optimal Path

While rewards enable the agents to learn to reach the goal, it is also essential to make sure that the path with which it reaches the goal is optimal. In the absence of this check, an agent will get stuck in an infinite loop. Every step is given a reward to ensure if the agent reaches the goal the fastest it can.

7.2 Future rewards

In Q Learning, we compute the Q value, by using the future return from a given state and given action. A dilemma is introduced where if we don't solve this, the approach becomes a greedy approach and future rewards cannot be given the same significance as the immediate rewards. So, in order to solve this a discounting factor is used to deal with this.

8 Result

The performance can be evaluated by visualizing $\in$ and visualizing rewards.

- **Visualize $\in$**

  The values of $\in$ is plotted over each episode.

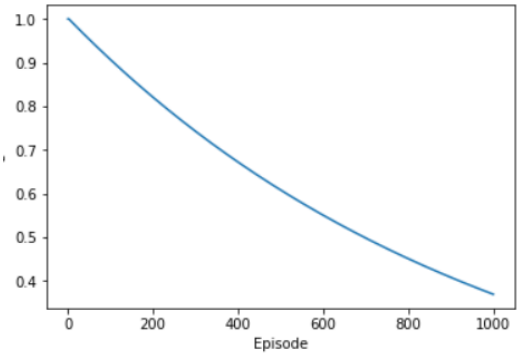

Fig 4: $\in$ vs episode

- **Visualize Rewards**

  The value of total-rewards is plotted over each episode. A rolling mean of window 10 is applied to visualize easier.

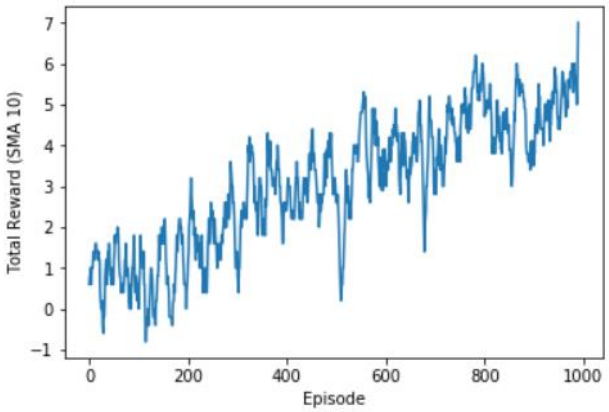

Fig 4: total-rewards vs episode

9 Conclusion

A reinforcement learning agent is built to navigate the classic 4x4 grid-world environment. In this project, the agent has learnt an optimal policy through Q-Learning which will allow it to take actions to reach a goal while avoiding obstacles. Key components of the Q-Learning algorithm are implemented and explained. Specifically, tabular Q-Learning approach is used which utilizes a table of Q-Values as the agent's policy.

References

[1]https://www.researchgate.net/publication/322424392_Data_Science_in_the_Research_Domain_Criteria_Era_Relevance_of_Machine_Learning_to_the_Study_of_Stress_Pathology_Recovery_and_Resilience

[2] https://en.wikipedia.org/wiki/Q-learning

[3]https://www.researchgate.net/figure/3D-view-of-a-4D-Gaussian-Mixture-Model-used-in-our-experiments_fig1_45872040