Reinforcement Learning agent to navigate the Classic 4x4 Grid-World environment.

Project 4 - CSE 574 Intro to Machine learning

Krishna Naga Karthik BODAPATI
Department of Computer Science
(SUNY) University at Buffalo
Buffalo NY, 14221
kbodapat@buffalo.edu

Abstract

The Goal of this project is to build a reinforcement learning agent to navigate the classic 5x5 grid-world environment sing Q Learning which will allow it to take actions to reach a goal while avoiding obstacles.

1 Introduction

1.1 Reinforcement Learning:

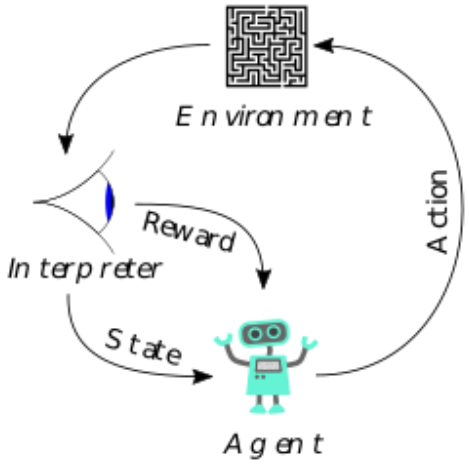
Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize some notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. It differs from supervised learning in not needing labelled input/output pairs be presented, and in not needing sub-optimal actions explicitly corrected. Instead the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). The environment is typically stated in the form of a Markov decision process (MDP), because many reinforcement learning algorithms for this context utilize dynamic programming techniques.The main difference between the classical dynamic programming methods and reinforcement learning algorithms is that the latter do not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible.

Markov Decision Process

A Markov Decision Process is a 4 – tuple (S, A, P_a, R_a) where

- S is the finite set of states,
- A is the finite set of actions,
- $\boldsymbol{P_a(s, s') = Pr(s_{t+1} = s' \mid s_t = s, a_t = a)}$ is the probability that the action ***a*** in state ***s*** at time ***t*** will lead to state ***s'*** at time t + 1.
- $\boldsymbol{R_a(s, s')}$ is the immediate reward received after transitioning from the state ***s*** to state ***s'***, due to action ***a***.

Markov Decision Processes are an extension of Markov chains.



1.2 Q Learning:

Q-Learning is a process in which we train some function $Q\theta : S \times A \to R$, 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 πθ:

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

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

Originally, Q-Learning was done in a tabular fashion. Here, we would create an S A array, our 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 the first is located in the ith row and at is the jth column, Q(st, at) = qi,j. We use a value iteration update algorithm to update our Q-values as we explore the environment's states:

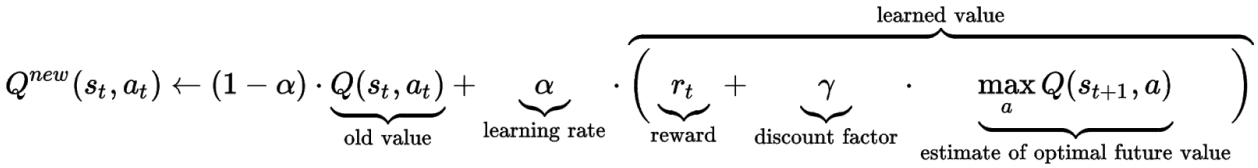


Figure 2: Our update rule for Q-Learning2

We are using our Q-function recursively to match (following our policy π) in order to calculate the discounted cumulative total reward. We initialize the table with all 0 values, and as we explore the environment (e.g., take random actions), collect our trajectories, [s0, a0, r0, s1, a2, r2, . . . , sT , aT , rT], and use these values to update our corresponding entries in the Q-table.
During training, the agent will need to explore the environment in order 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
2https://en.wikipedia.org/wiki/Q-learning
optimal rewards since they may not take the optimal sequence of actions to get there. In order to account for this, we will slowly encourage the agent to follow it's policy in order 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 s-greedy strategy. Here, we introduce a parameter s and set it initially to 1. At every training step, we decrease it's value gradually (e.g., linearly) until we've reached some predetermined minimal value (e.g., 0.1). In this case, we are annealing the value of s linearly so that it follows a schedule.
During each time step, we have our agent either choose an action according to it's 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 s otherwise taking an action following the agent's policy.
In short the goal of Q learning is to learn which says what agent need to do for a given state. We need to find a policy which gives maximum rewards. Since Q learning is a model free algorithm its does not need any transitional probability. The following is the algorithm for Q Learning

1.2.1 Exploration vs Exploitation:

This is a trade off between whether model should choose random next state (exploration) or chose next state with highest value in Q table.This exploration part prevents model to be stuck on choosing the same state for every time. To do this we use a variable **Epsilon.**

1.2.2 Learning Rate:

In Q Learning, Learning rate (alpha) denotes how important the new state. Setting alpha to a less than model gives less importance to the new state when calculating Q value and more importance will be given to current state and vice versa

1.2.3 Discount Factor:

The discount factor gamma determines the importance of future rewards.If gamma is 0 then the agent only considers current rewards and If gamma is 1 then agent only considers future rewards

3 Environment

For this project, the 5 × 5 gridworld shown in Figure 1 was chosen as the default underlying environment. The automated agent, located initially at the top-left corner of the gridworld, has to find a way to reach the goal situated at the bottom-right corner.

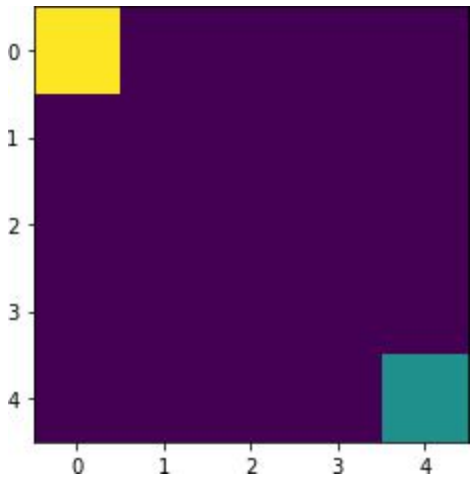


Figure 1: The initial state of our basic grid-world environment.

The environment we provide is a basic deterministic nxn grid-world environment (the initial state for a 5 grid-world is shown in Figure 1) where the agent (shown as the green square) has to reach the goal (shown as the yellow square) in the least amount of time steps possible.
The environment's state space will be described as an nxn 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.

- (a) Action 0: Move Down – transition from state [x, y] to [x + 1, y]
- (b) Action 1: Move Up – transition from state [x, y] to [x – 1, y]
- (c) Action 2: Move Left – transition from state [x, y] to [x, y – 1]
- (d) Action 3: Move Right – transition from state [x, y] to [x, y + 1]

The agent earns an immediate reward of +1 for moving towards the goal and an immediate reward of –1 for either moving away or remaining at the same distance from the goal. In summary, the agent starts at

location [0, 0] and has to find an optimal policy to reach the goal at location [4, 4] while maximizing the total earned reward.

4. Goal

The main challenge of this project is to find a policy $\pi : S \rightarrow A$ which the 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 (used to give more weight to more immediate rewards), s_t is the state at time step t, a_t is the action the agent took at time step t, and s_{t+1} is the state which the environment transitioned to after the agent took the action.

5. Implementation

The important steps in the Q-learning are

Step 1 - choose random next state or choose next state with highest Q-value (epsilon)

Step 2 - Take action and move to the next chosen state (alpha,gamma)

Step 3 - Calculate the Q value for this new state and store that in Q-state

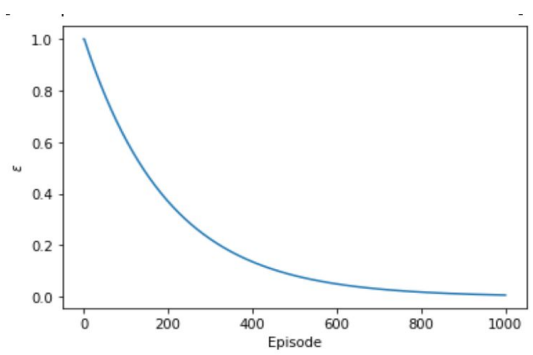
Updating Q table

Initially the values in Q table are all set to zero and the dimensions of Q Table in our problem is (5x5x4) [5x5 grid and 4 actions]. Here we use this formula to update the value in Q table

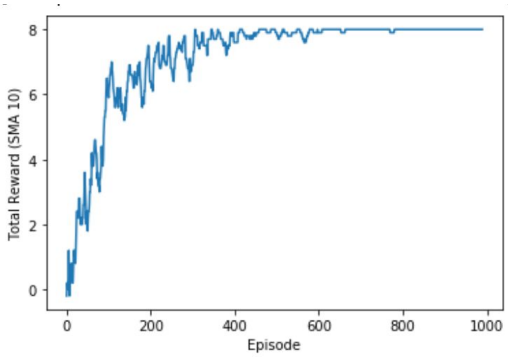
$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of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

6. Results

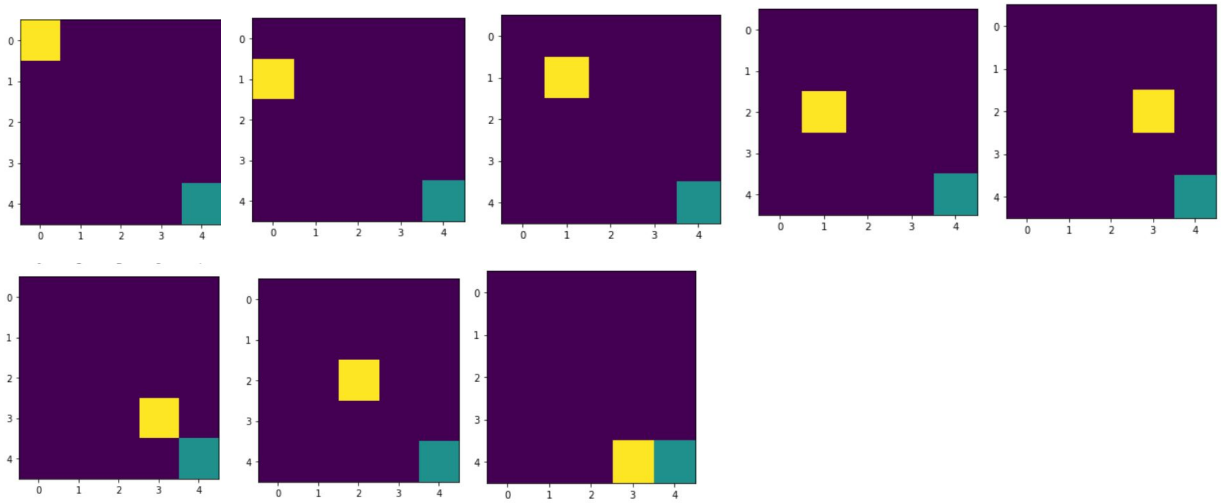
ε-greedy strategy is used for action selection by the agent. An epsilon decays exponentially,It is 1.0 in the start of an episode and decays slowly (0.008 time) per iteration shown in below image was chosen. This ensures that exploration is high in the beginning and very less in the end Similarly the exploration increases for every iteration, the agent and the reward function increases for every episode but it is not smooth because of the random state in the training phase



Rewards vs Episodes



Epsilon vs Episode



These are the following states after learning optimal policy.

Q Table

```
array([[[ 0.99726107,  0.7975654 ,  2.8       ,  0.79503228],
        [ 0.98686144,  0.76608707,  2.8       ,  0.78935776],
        [ 0.96909685,  0.62863649,  2.8       ,  0.77527748],
        [ 0.90152291,  0.73618685,  1.        ,  0.75813219],
        [ 1.        , -0.86491483, -0.89058101,  0.70273868]],

       [[ 1.36036278,  0.57123208,  0.77123208, -0.86491483],
        [ 1.02161106,  0.61698566,  0.3439    , -0.56953279],
        [ 0.94185026,  0.268559  ,  0.271     , -0.56953279],
        [ 0.94766524,  0.152     ,  0.271     , -0.40951   ],
        [ 1.        , -0.5217031 , -0.79410887, -0.77123208]],

       [[ 2.30781931, -0.79410887,  1.111414  ,  0.49006361],
        [ 2.64822919, -0.5217031 ,  0.79410887,  0.268559  ],
        [ 2.65869614, -0.468559  ,  0.527608  , -0.26371   ],
        [ 1.07300405, -0.1       ,  0.3439    , -0.1       ],
```

```
 [ 1.        , -0.61257951, -0.65132156, -0.6861894 ]],

[[ 0.37      , -0.091     ,  2.37973502,  0.08      ],
 [ 0.613     , -0.3439    ,  2.72991165,  0.327608  ],
 [ 0.639559  , -0.468559  ,  1.06674244,  0.327608  ],
 [ 2.7778397 , -0.40951   ,  0.77123208,  0.2168    ],
 [ 1.        , -0.74581342, -0.61257951, -0.40951   ]],

[[-0.1       , -0.1       ,  0.703     ,  0.        ],
 [ 0.        ,  0.        ,  0.5149    ,  0.08      ],
 [-0.1       ,  0.327608  ,  0.        , -0.1       ],
 [-0.1       , -0.1       ,  0.93538918, -0.1       ],
 [ 0.        ,  0.        ,  0.        ,  0.        ]]])
```

7. References:

https://en.wikipedia.org/wiki/Reinforcement_learning
https://en.wikipedia.org/wiki/Q-learning