Reinforcement Learning on 4x4 Grid Game

Neelesh K Bhajantri
Department of Computer Science
University at Buffalo
Buffalo, New York 14260
neeleshb@buffalo.edu

Abstract

In this project we have implemented the policy function, updating the Q table and training process. We have tuned the hyper parameters such that the epsilon value decreases for every iteration as a result the number of steps in the grid world reduces. We have achieved the grid game in 9 steps.

1 Introduction

Reinforcement learning, due to its generality, is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics and genetic algorithms. In the operations research and control literature, reinforcement learning is called approximate dynamic programming, or neuro-dynamic programming. The problems of interest in reinforcement learning have also been studied in the theory of optimal control, which is concerned mostly with the existence and characterization of optimal solutions, and algorithms for their exact computation, and less with learning or approximation, particularly in the absence of a mathematical model of the environment. In economics and game theory, reinforcement learning may be used to explain how equilibrium may arise under bounded rationality.
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), as illustrated in Figure 1.

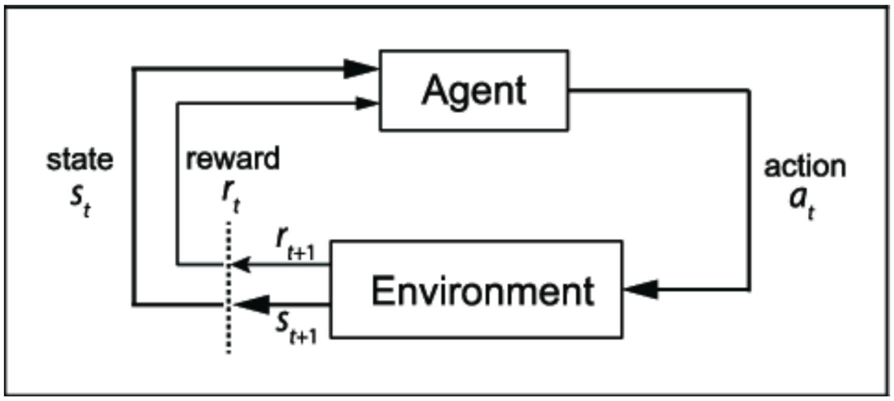
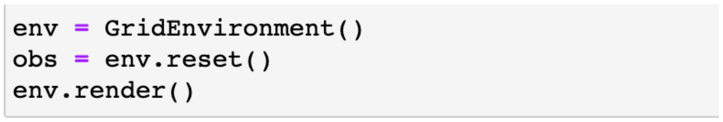


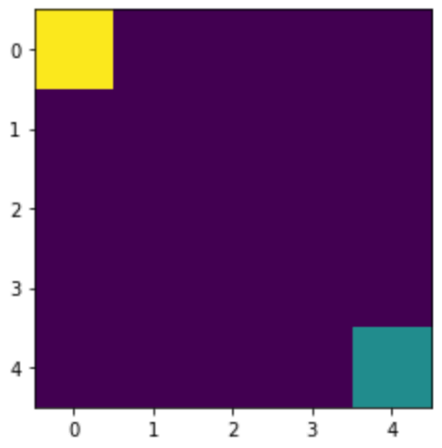
Figure 1: The canonical MDP diagram[1]

2 Environment

The environment that is provided is a basic deterministic n × n grid-world environment (the initial state for a 4 × 4 grid-world is shown in Figure 3) 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 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.
We are given a grid world environment which has a transition function, reset function, step function and a render function. This is what we are going to use to train the game and run our process in.



```
env = GridEnvironment()
obs = env.reset()
env.render()
```



We are also given a random agent that randomly takes steps, which doesn't learn or remember anything. Then we are given a heuristic agent which succeeds to reach the goal but takes a lot of steps.

3 Architecture

3.1 Tabular Q-Learning

Our task is 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 γ ∈ [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.

Q-Learning is a process in which we train some function Q_{theta}:S X A->R, parameterized by theta, to learn a mapping from state-action pairs to their Q-value, which is the expected discounted reward for the policy:

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

Reinforcement learning involves an agent, a set of states S, and a set A of actions per state. By performing an action in A, the agent transitions from state to state. Executing an action in a specific state provides the agent with a reward (a numerical score).

The goal of the agent is to maximize its total (future) reward. It does this by adding the maximum reward attainable from future states to the reward for achieving its current state, effectively influencing the current action by the potential future reward. This potential reward is a weighted sum of the expected values of the rewards of all future steps starting from the current state.

As an example, consider the process of boarding a train, in which the reward is measured by the negative of the total time spent boarding (alternatively, the cost of boarding the train is equal to the boarding time). One strategy is to enter the train door as soon as they open, minimizing the initial wait time for yourself. If the train is crowded, however, then you will have a slow entry after the initial action of entering the door as people are fighting you to depart the train as you attempt to board. The total boarding time, or cost, is then:

- 0 seconds wait time + 15 seconds fight time

On the next day, by random chance (exploration), you decide to wait and let other people depart first. This initially results in a longer wait time. However, time fighting other passengers is less. Overall, this path has a higher reward than that of the previous day, since the total boarding time is now:

- 5 second wait time + 0 second fight time.

Through exploration, despite the initial (patient) action resulting in a larger cost (or negative reward) than in the forceful strategy, the overall cost is lower, thus revealing a more rewarding strategy.

3.2 Implement Policy function
We have defined the policy function such that for every random value of uniform is its less than the epsilon we set the action variable to explore the action space. This means that the variable is moving towards the goal.
If that's not the case then we have to go back to the learned parameters because this is going to the path away from the goal we have to reach. Basically, policy function is going to tell the action variable whether to move near or select other position of learned parameters to reach the end goal.

3.3 Updating the Q-Table:
Here, we would create an |S| × |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 s_t 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:

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

3.4 Training Algorithm
We stored the training progress of each episode making the agent remembering its previous steps, we have solved the environment over 1000 episodes. We are going to reset the

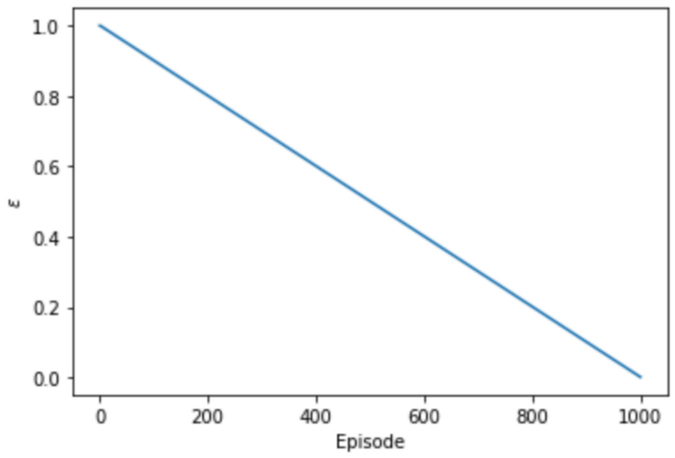
environment, rewards and steps for every new episode. We are going to select an action state, store a copy of the current state move to the next state, add the reward and update the agent with these environments of performed action reward and current state.

4 Results

For every episode we are updating the epsilon value also and this change or decrease is going to result in the better performance of the action variable because it is going to reach more near gaining rewards.
Here is the change of epsilon values for every episode:

```
plt.xlabel('Episode')
plt.ylabel('$\epsilon$')
plt.plot(epsilons)
```

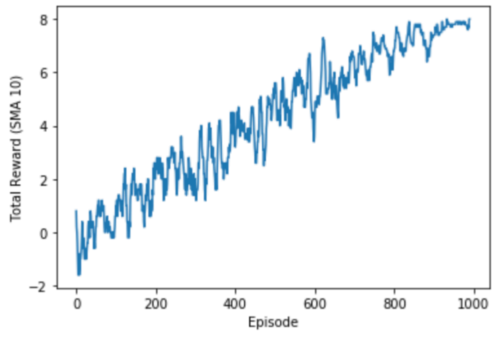
```
[<matplotlib.lines.Line2D at 0x1021cbdd0>]
```



Let’s also see how the rewards change per episode:

```
window = 10
plt.xlabel('Episode')
plt.ylabel('Total Reward (SMA 10)')
plt.plot([np.mean(total_rewards[tr:tr+window]) for tr in range(window, len(total_rewards))])
```

```
[<matplotlib.lines.Line2D at 0x1124adb10>]
```



5 Conclusion

We built a reinforcement learning agent to play the 4x4 grid world game and have successfully reached the end goal in lesser number of steps than that of the heuristic method took.