Reinforcement Learning

Snigdha Gupta
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
snigdhag@buffalo.edu

Abstract

This project aims at implementing a machine learning paradigm called Reinforcement Learning. We build an agent that navigates the classic 5x5 grid-world environment and learns an optimal policy through Q-Learning allowing the agent to reach the goal in minimum steps. We create an environment that is compatible with OpenAI Gym library and train the agent by finding a policy which is used by the agent to make decisions thereby maximizing the rewards.

1 Introduction

1.1 Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning that focuses on how something, might act in an environment in order to maximize some given reward. Reinforcement learning algorithms study the behavior of subjects in such environments and learn to optimize that behavior.

A common application of RL is in the gaming world where an agent learns to complete the goal of the game.

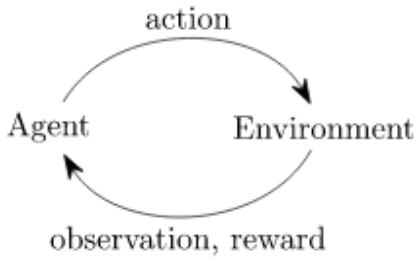

Figure 1 Concept of reinforcement learning

1.2 Markov Decision Process

Markov Decision Process or MDP is typically how an automated agent learns to take an action in response to the current state of an environment in order to maximize some reward.

Markov decision processes give us a way to formalize sequential decision making. This formalization is the basis for structuring problems that are solved with reinforcement learning.

Components of an MDP:

- **Agent** – An **agent** takes actions; such as a video game character navigating in the game.
- **Environment** – The world through which the agent moves, and which responds to the

agent. The **environment** takes the agent's current state and action as input, and returns as output the agent's reward and its next state. For a video game character, the environment is scenario in which the character is present.

- **State –** 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. For example, position of the video game character in the game.
- **Action** – It is the set of all possible moves the agent can make. An agent usually chooses from a list of discrete, possible **actions**. In video games, the list might include running right or left, jumping high or low.
- **Reward –** A **reward** is the feedback by which we measure the success or failure of an agent's actions in a given state. For example, in a video game, the character receives points as rewards.

In an MDP, the agent interacts with the environment it is placed in. These interactions occur sequentially over time. At each time step, the agent will get some representation of the environment's state. Given this representation, the agent selects an action to take. The environment is then transitioned into a new state, and the agent is given a reward as a consequence of the previous action.

This process of selecting an action from a given state, transitioning to a new state, and receiving a reward happens sequentially over and over again, which creates something called a trajectory that shows the sequence of states, actions, and rewards.

Throughout this process, it is the agent's goal to maximize the total amount of rewards that it receives from taking actions in given states. This means that the agent wants to maximize not just the immediate reward, but the cumulative rewards it receives over time.

This is mathematically expressed later in **Architecture**.

1.3 Q-Learning

Q-Learning is a RL technique used for learning the optimal policy in Markov Decision Process. The objective of Q-learning is to find a policy that is optimal that is the expected value of the total reward over all successive steps is the maximum achievable. So, in other words, the goal of Q-learning is to find the optimal policy by learning the optimal Q-values for each state-action pair.

We train some function $Q_\theta : S\ X\ A \rightarrow \mathcal{R}$, parameterized by θ, to learn a mapping from the stat-action pairs to their Q-value, which is the expected discounted reward for following the policy π_θ:

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

Figure 2 Policy

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|x|A| array, our Q-Table, which would have entries $q_{i,j}$ where i corresponds to the i^{th} state (the row) and j corresponds to the j^{th} action (the column), so that if s_t is located in the i^{th}row and a_t is the j^{th} 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:

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

Figure 3 Update rule for Q-Learning

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 (taking random actions), collect our trajectories, $[s_0, a_0, r_0, s_1, a_1, r_2, \ldots\ldots, s_T, a_T, r_T]$ and use these values to update our corresponding entries in the Q-table.

Exploration is the act of exploring the environment to find out information about the environment.

Exploitation is the act of exploiting the information that is already known about the environment in order to maximize the return.

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. This however, poses a problem: the agent may not reach the states which reach to 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 its 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 $\varepsilon\,(epsilon) -$ greedy strategy. Here, we introduce a parameter ε and set it initially to 1. At every training step, we decrease its value gradually (e.g., exponentially) until we've reached some predetermined minimal value (e.g. 0,1). In this case, we are annealing the value of ε 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 ε otherwise taking an action following the agent's policy.

2 Environment

Reinforcement learning environments can take on many different forms, including physical simulations, video games, stock market simulations, etc. The reinforcement learning community (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 we provide is a basic deterministic n x n grid-world environment (the initial state for a 5 x 5 grid-world where the agent (shown as yellow square) has to reach the goal (shown as the green square) in the least amount of time steps possible. Hence our environment is a **3D structure of 5x5x4 dimension**.

The environment's state space will be described as an n x 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: **right, left, up and down**. 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.

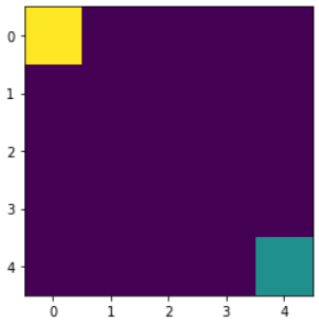

Figure 4 Initial state of basic grid-world environment

Policy

A policy is a function that maps a given state to probabilities of selecting each possible action from that state. We will use the symbol π to denote a policy.

When speaking about policies, formally we say that an agent "follows a policy." Here the agent needs to follow the policy as mentioned below:

Policy: 0 – down (+1), 1 – up (-1), 2 – right (+1), 3 – left (-1)

3 Preprocessing

The Q-table is used to store the maximum expected future reward for action at each state. This table guides us to the best action at each state.

Initially we set every value in the Q-table to 0 (one-hot encoding).

4 Architecture

The canonical graph of Markov Decision Process is shown below:

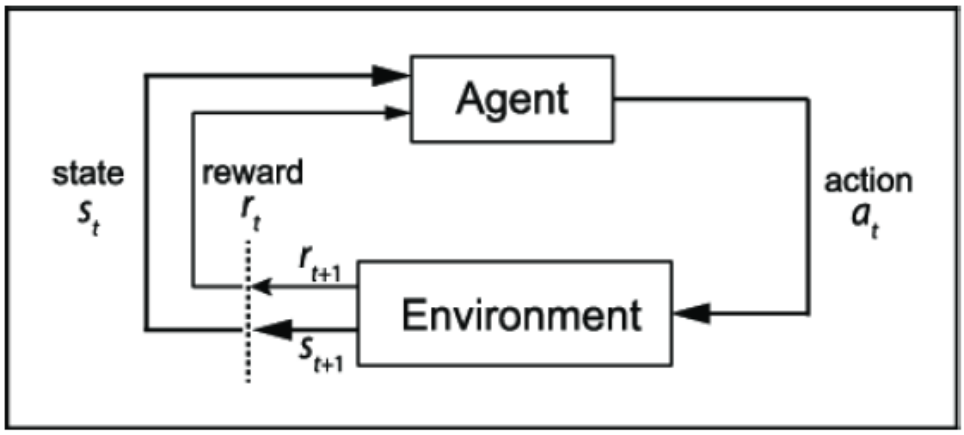

Figure 5 Canonical graph of Markov Decision Process

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

Our task is to find a policy $\pi : S \rightarrow$ A which our agent will use to take actions in the environment which maximize cumulative reward, i.e.:

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

Figure 6 Cumulative Reward

Where $\gamma \epsilon [0, 1]$ 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 Results

The following results were obtained:

We first train the agent for **1000 episodes** while **linearly decaying** the epsilon with a **decay rate ($\delta\varepsilon$)** of **0.001**.

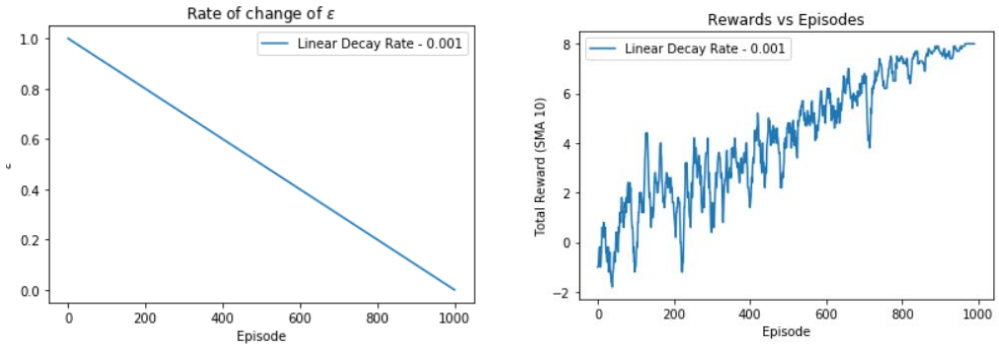

Figure 7 (a) Rate of change of Epsilon (b) Reward vs Episode with linear decay

In the next attempt, we train the agent for **1000 episodes**. However, this time the epsilon decays **exponentially** with a **decay rate ($\delta\varepsilon$)** of **0.99**.

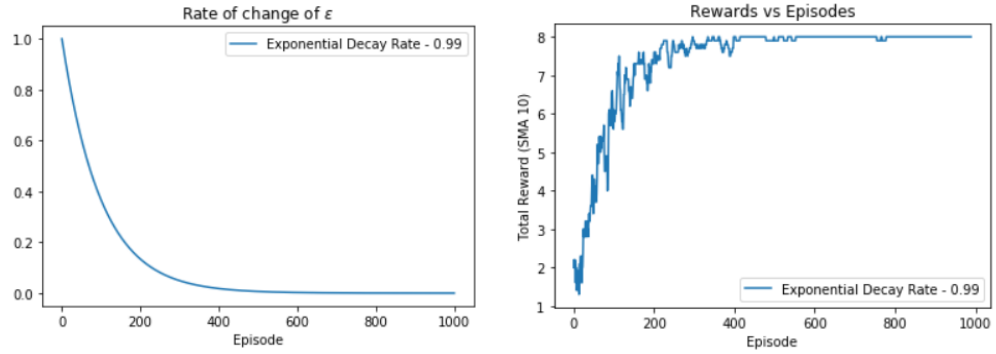

Figure 8 (a) Rate of change of Epsilon (b) Reward vs Episode with exponential decay

Next, we train the agent for **100 episodes** while **exponentially** decaying the epsilon with a **decay rate ($\delta\varepsilon$)** of **0.95**.

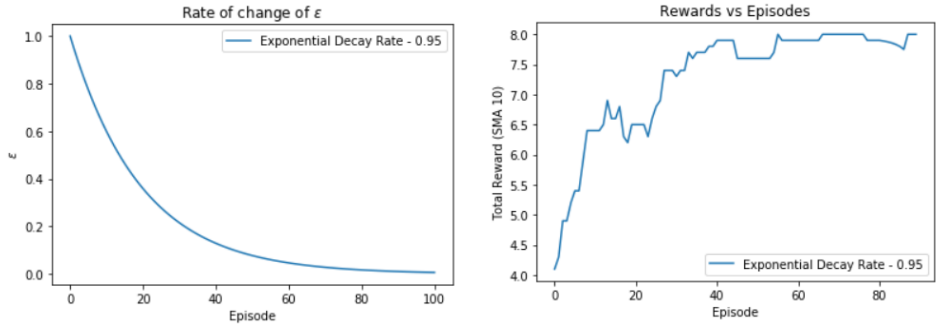

Figure 9 (a) Rate of change of Epsilon (b) Reward vs Episode with exponential decay

On successive trial and error attempts we finally train the agent for **1000 episodes** while **exponentially decaying** the epsilon with a **decay rate ($\delta\varepsilon$)** of **0.95**.

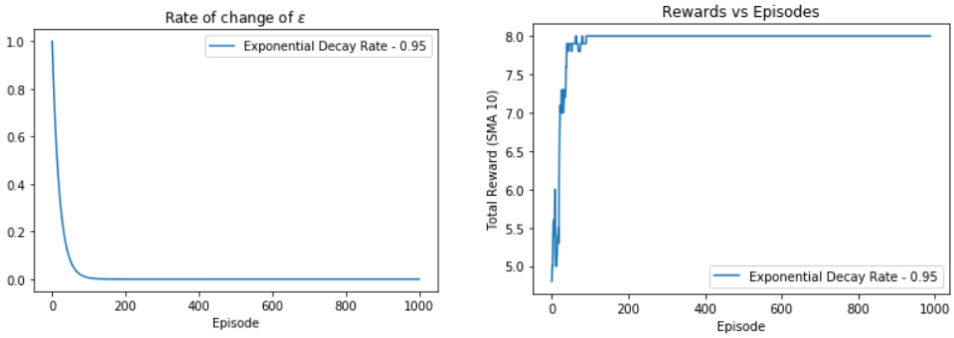

Figure 10 (a) Rate of change of Epsilon (b) Reward vs Episode with exponential decay

The agent learns with a learning rate (α) of 0.1 and discounted rate (γ) of 0.9.

```
[[[ 5.6953279   0.19556279  0.69952451 -0.16210182]
  [ 0.109      -0.236971    0.449371   -0.09697266]
  [ 0.          0.          0.2881     -0.147952  ]
  [ 0.199       0.          0.1         0.        ]
  [ 0.1         0.         -0.1         0.        ]]

 [[ 5.217031   -0.17922172  0.3529      0.11172737]
  [ 0.75481106  0.          0.12167098 -0.0401936 ]
  [ 0.55004316  0.          0.         -0.074071  ]
  [ 0.         -0.091       0.40904599  0.        ]
  [ 0.67914253  0.          0.         -0.1       ]]

 [[ 0.3639241  -0.24686353  4.68559     0.24352997]
  [ 0.2152     -0.27404601  4.0951      0.01671498]
  [ 0.1        -0.29063575  3.439       0.15467734]
  [ 0.199      -0.14707793  2.71        0.05287295]
  [ 1.9        -0.0812881  -0.03158165  0.03581039]]

 [[ 0.1        -0.06612499  0.23059     0.        ]
  [ 0.4339      0.          0.         0.        ]
  [ 0.          0.          0.        -0.0748    ]
  [ 0.1         0.          0.28323233  0.        ]
  [ 1.          0.         -0.09863811 -0.089461  ]]

 [[ 0.         -0.091       0.         0.        ]
  [-0.1        -0.14941     0.19       0.        ]
  [ 0.          0.          0.         0.        ]
  [ 0.          0.          0.         0.        ]
  [ 0.          0.          0.         0.        ]]]
```

Figure 11 Q-Table of learned agent

6 Conclusion

The above results suggest that decaying the epsilon exponentially gives us expected results. We observe that when decay rate is reduced linearly, the agent exploits the environment for a greater number of episodes. Also, when epsilon is reduced exponentially, and decay rate is 0.95 then the agent starts to exploit the environment much earlier. Hence the most optimal result for training the agent to reach the goal is obtained at **1000 episodes; with decay rate, $\delta\varepsilon = 0.95$; learning rate $\alpha = 0.1$; and discounted rate $\gamma = 0.9$.**

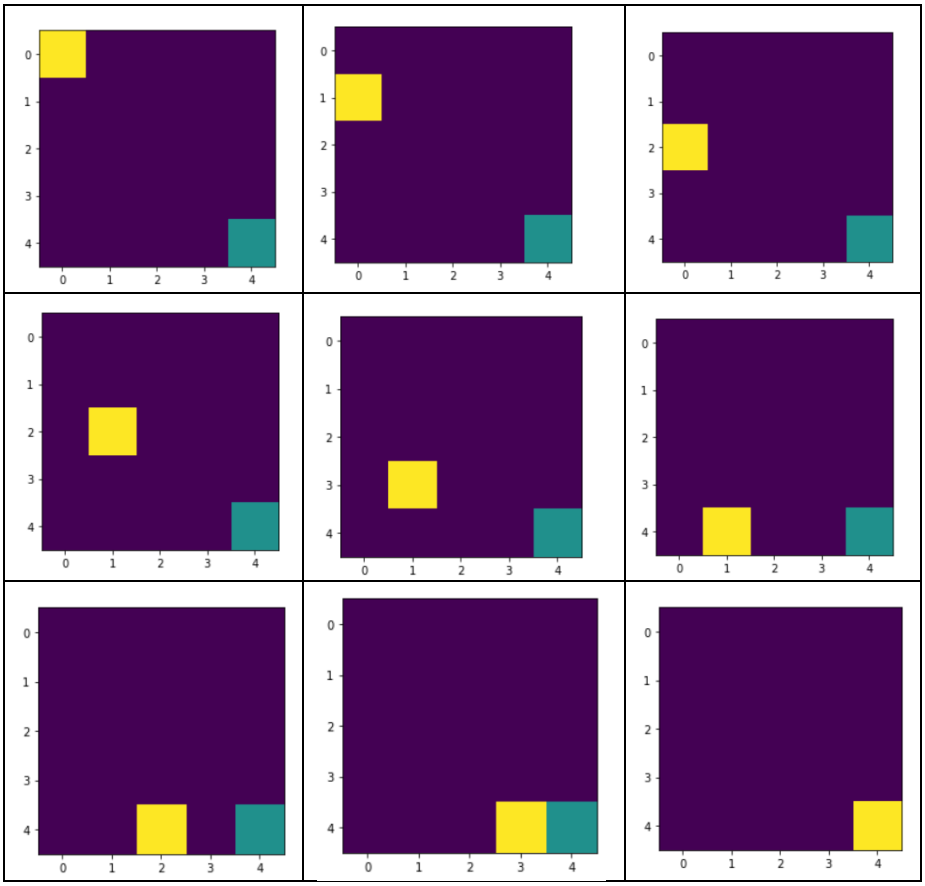

Figure 12 Agent taking decision to reach goal in minimum number of steps

The agent reached its goal in 8 steps which is the optimal number of steps.

References

[1] https://deeplizard.com/learn/video/nyjbcRQ-uQ8

[2] https://skymind.ai/wiki/deep-reinforcement-learning

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

[4] https://deeplizard.com/learn/video/eMxOGwbdqKY

[5] https://medium.com/machine-learning-for-humans/reinforcement-learning-6eacf258b265

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

[7] https://gym.openai.com/docs/