CSE 474/574: Introduction to Machine Learning (Fall 2019)

Sargur N. Srihari
University at Buffalo, The State University of New York
Contact: 716-645-6162 (O), srihari@buffalo.edu

November 11, 2019

1 Overview

Your task is to build a reinforcement learning agent to navigate the classic 4x4 grid-world environment. The agent will learn an optimal policy through Q-Learning which will allow it to take actions to reach a goal while avoiding obstacles. To simplify the task, we will provide the working environment as well as a framework for a learning agent. The environment and agent will be built to be compatible with OpenAI Gym environments and will run effectively on computationally-limited machines. Furthermore, we have provided two agents (random and heuristic agent) as examples.

2 Background

2.1 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), as illustrated in Figure 1.

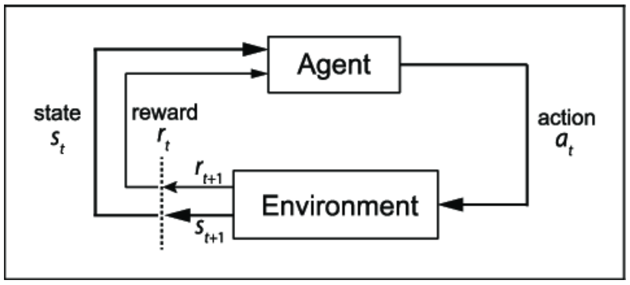

Figure 1: The canonical MDP diagram[1]

[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

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
- $P = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the state transition probability function
- $R : S \times A \times S \to \mathbb{R}$ is the reward function

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

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

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.

2.2 Q-Learning

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

$$\pi(s_t) = \operatorname*{argmax}_{a \in A} Q_\theta(s_t, a) \tag{2}$$

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 s_t is located in the ith row and a_t 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:

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

Figure 2: Our update rule for Q-Learning[2]

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, $[s_0, a_0, r_0, s_1, a_2, r_2, \ldots, s_T, a_T, r_T]$, and use these values to update our corresponding entries in the Q-table.

During training, the agent will need to explore the environment 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

[2] `https://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 ϵ-greedy strategy. Here, we introduce a parameter ϵ 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 ϵ 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 ϵ otherwise taking an action following the agent's policy.

3 Task

For this project, you will be tasked with both **implementing** and **explaining** key components of the Q-learning algorithm. Specifically, we will provide a simple environment and a framework to facilitate training, and you will be responsible for supplying the missing methods of the provided agent classes. For this project, you will be implementing tabular Q-Learning, an approach which utilizes a table of Q-values as the agent's policy.

3.1 Environment

Reinforcement learning environments can take on many different forms, including physical simulations, video games, stock market simulations, etc. 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 (`https://gym.openai.com/`).

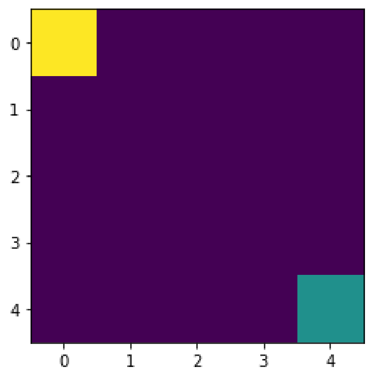

Figure 3: The initial state of our basic grid-world environment.

The environment we provide is a basic deterministic $n \times n$ grid-world environment (the initial state for an 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 $n \times n$ 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.

[2]`https://becominghuman.ai/lets-build-an-atari-ai-part-1-dqn-df57e8ff3b26`

3.2 Code

This project will not be (relatively) computationally intensive, so most machines will be capable of training the agent quickly. We will provide code templates in the form of Jupyter Notebooks hosted on Google Colab: *https://colab.research.google.com/drive/1rQjtiSfuZ*$_{Jd}$*rW6DXtb92A8tvUqAyP1*.

Google Colab is a free service which hosts Jupyter Notebook instances, giving free access to cloud-based Python environments as well as free GPU/TPU allocation time. This project will be designed to run completely in Google Colab with the intent that you can complete the entire project without having to install anything (although you are welcome to work in whatever environment you want).

We will provide code in the form of a template implementation of the learning agents. You will be tasked with filling in the missing sections to get the implementation working. The expectation will be that the agent is compatible with OpenAI Gym environments. We will also provide a notebook to test your implementation and asses it's abilities.

Successful implementations will have agents capable of learning to navigate the environment and reach it's maximum reward on average within a fair amount of epochs.

3.3 Report

Along with the code, you will be expected to describe your efforts through a report. The report should contain a section describing your challenge, the Q-learning algorithm, and the environment with which your agent is trained. Each of the key components listed above should be described in it's own section, including the role it plays in the algorithm, any hyperparameters associated with it and how the hyperparameters affect the agent's learning, and challenges and details related to your implementation of the component. The report should conclude with results and reflections.

The expectation, here, is that you will be able to describe each of these components at a high-level in your own words. Since the code will be given as a template (and there are a lot of resources online with regards to Q-Learning), we expect the coding portion to be straight-forward. As such, grading will be primarily based on your ability to explain what you have implemented versus the end results.

4 Deliverables

You need to submit only one zip-file *proj4.zip*, that will contain the following:

- Jupyter Notebook (*main.ipynb*), that will contain your working implementation.
- The report (*report.pdf*), which is to be written in LaTeX

Submit the project on the CSE student server with the following script:
`submit_cse474 proj4.zip` for undergraduates
`submit_cse574 proj4.zip` for graduates

5 Scoring Rubric

- Report (30 points)
- Code (70 points):
  - Update Q Table (30 points)
  - Define Policy (20 points)
  - Define the training process (20 points)

6 Due Date and Time

The due date is **11:59PM, Dec 4**.