ECE433/COS435 Introduction to RL
Assignment 4: Q-learning and SARSA
Spring 2024

Fill me in

Your name here.

Due March 4, 2024

Collaborators

Fill me in

Please fill in the names and NetIDs of your collaborators in this section.

Instructions

Writeups should be typesetted in Latex and submitted as PDFs. You can work with whatever tool you like for the code, but **please submit the asked-for snippet and answer in the solutions box as part of your writeup. We will only be grading your write-up.**

Question 1. Q-Learning (40 points)

Tabular setting

If the state and action spaces are sufficiently small, we can maintain a table containing the value of $Q(s, a)$ – an estimate of $Q^*(s, a)$[1] – for every (s, a) pair. In this *tabular setting*, given an experience sample (s, a, r, s'), the update rule is

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a) \right) \tag{1}$$

where $\alpha > 0$ is the learning rate and $\gamma \in [0, 1]$ is the discount factor.

[1] Here, $Q^*(s, a)$ refers to optimal Q value function.

Question 1.a: Regular Q-Learning (10 points)

Why is it difficult to extend this learning rule to the game of Tetris or similar Atari games?

Solution

Your solution here...

Approximation setting

Here, we instead represent our Q-values as a function $\hat{q}(s, a; \mathbf{w})$, where $\mathbf{w}$ are parameters of the function (typically a neural network's weights and bias parameters). In this *approximation setting*, the update rule becomes

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(r + \gamma \max_{a' \in \mathcal{A}} \hat{q}(s', a'; \mathbf{w}) - \hat{q}(s, a; \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{q}(s, a; \mathbf{w}). \tag{2}$$

In other words, given current parameters at iteration i, $\mathbf{w}_i$, we aim to minimize the loss at $\mathbf{w}_{i+1}$ which is:

$$L(\mathbf{w}_{i+1}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a' \in \mathcal{A}} \hat{q}(s', a'; \mathbf{w}_i) - \hat{q}(s, a; \mathbf{w}_{i+1}) \right)^2 \right] \tag{3}$$

Question 1.b: Continuous actions (10 points)

Consider an environment such as Mountain Car Continuous where the action space is $[-1, 1] \in \mathbb{R}$. How might our representation of the Q-function described in (1.a) change to support our use case?

Solution

Your solution here...

Question 1.c: Action spaces (10 points)

We can represent a Q-function $Q(s, a)$ as either a function $Q(s; w) : S \to \mathbb{R}^{|A|}$ outputting the vector of Q-values $[Q(s, a_1), \ldots, Q(s, a_{|A|})]$ all at once, or a function $Q(s, a; w) : S \times A \to R$ outputting a single Q-value $Q(s, a)$. What is a benefit of implementing the former over the latter? What is one drawback?

Solution

Your solution here...

Question 1.d: Policy iteration (10 points)

Policy iteration is a reinforcement learning algorithm that provably improves a policy. In policy iteration, there are two steps: one called "policy evaluation" and another is "policy improvement". Step "policy evaluation" estimates the "value" of a state under the current policy π being learned:

$$V^{\pi}(s) = \sum_{s' \in S} P_{\pi(s)}(s' \mid s) \; [r(s, a, s') + \gamma \; V^{\pi}(s')]$$

Step "policy improvement" improves the current policy π based on the evaluation. Other than the fact Q-learning learns a Q-function and policy iteration learns a V-function, how do these two methods differ from each other and explain in details? (Consider the policy usage and the environment transition function)

Solution

Your solution here...

Question 2. Learning Value Functions (55 points)

We will implement three RL algorithms:

- 1. Q-Learning (Off-policy TD control)
- 2. SARSA (On-policy TD control)
- 3. Monte Carlo (Episodic control)

We will use a simple neural network to approximate $Q(s, a)$ in a discrete-action environment (e.g., "CartPole-v1").

Question 2.a (10 points)

Paste the code block implementing `QNetwork` below.

Solution

```
    ############################
    # YOUR IMPLEMENTATION HERE #

    ############################

```

Question 2.b (5 points)

Paste the code block implementing `epsilon-greedy policy` below.

Solution

```
    ############################
    # YOUR IMPLEMENTATION HERE #

    ############################

```

Question 2.c.1 (15 points)

Paste the code block implementing `q_learning_update` below.

Solution

```
    ############################
    # YOUR IMPLEMENTATION HERE #

    ############################

```

Question 2.c.2 (15 points)

Paste the code block implementing `sarsa_update` below.

Solution

```
    ############################
    # YOUR IMPLEMENTATION HERE #

    ############################

```

Question 2.c.3 (5 points)

Paste the learning curves from the three methods (Q-learning, SARSA, Monte Carlo regression) below, and analyze the performances of these methods.

Solution

Your answer here...

Question 2.c.d (10 points)

Examine the training loops and update functions for Q-learning and SARSA. Assuming that you're given a dataset of transitions (s_t, a_t, r_t, s_{t+1}) collected from a policy $\beta(a|s)$, will these

two methods converge to the same policy? Why or why not?

Solution

Your answer here...

Question 3 (5 points)

Course Feedback

What would you change so far about the course?

Solution

Your answer here...

Midterm Questions

For this year's midterm, the course staff is making questions, but also allowing students to create (a public bank) of potential questions we may put on the midterm. This will also form a nice study guide for the midterm itself.

Propose 3 simple questions that you would want to see on the midterm. There will be a Google Form to submit your questions, which we will publicly release as a study guide. Use the Google Form linked **here.**

The spreadsheet with all other student's questions can be found **here**.