CS 5033 - RL Project Report

Airi Shimamura Khoi Trinh

1. Introduction

For our RL project this semester, we want to build an RL agent that can easily beat the CartPole game.

CartPole v1 is a game environment provided in the Gym package from OpenAI. In this environment, there is a pole, attached to a cart (hence the name CartPole). The cart moves along a frictionless track. Force is applied in the left and right direction of the cart. The goal of the game is to keep the pole upright for as long as possible. For each step taken, a +1 reward is given, including the termination step. The maximum points achievable in the game is 475. There are a few conditions that, if met, will end the game.

First, if the pole angle is greater than 12 degrees, the game ends.

Second, if the cart position is greater than 2.4 (or the center of the cart reaches either end of the display), the game ends.

Finally, if episode length is greater than 500, the game ends.

For the criteria related to the CartPole environment, the agent will play the game for a 10000 episodes, and in each episode, the last 100 steps will have their rewards recorded (if the number of steps is less than 100, then the average will be over however many steps was taken for that episode) in order to generate an average. If any of the failing conditions is met, a penalty of -10 is given, otherwise, a reward of +1 is given.

2. Literature Review

2.1. Reinforcement Learning

The goal of reinforcement learning (RL) is not simply to play a game. Rather, the driving force is to create systems that can be adaptive in the real world (Arulkumaran et al., 2017).

The essence of RL is learning through interaction and reward-driven behavior. The RL agent interacts with the environment and the results of its actions; it can then adjust its behavior in response to the reward(s) received (**?**)

This trial-and-error behavior can be considered the root of RL.

2.2. SARSA

SARSA is an on-policy temporal difference learning algorithm (Sutton and Barto, 1998).

It is a popular reinforcement learning algorithm. At each time step, the agent selects an action according to its policy, observes the next state and reward, and updates the estimated value of the current state-action pair using a learning rate and discount factor. The algorithm maintains a Q-value function that estimates the expected future rewards for each state-action pair. The agent selects an action using an exploration-exploitation strategy and takes the selected action in the environment.

The general equation for SARSA learning is:

$$Q(s,a) < -Q(s,a) + \alpha * (R + \gamma * Q(s', a') - Q(s,a))$$

where α is the learning rate, and γ is the discount factor for future reward. For any given state-action pair, a new state-action value is obtained with a small correction to the old state-action value (**?**)

2.3. Q-Learning

Q-learning is a reinforcement learning algorithm that enables an agent to learn an optimal policy by observing and updating the estimated value of state-action pairs. The agent selects an action and observes the next state and reward. Q-Learning is a very similar algorithm to SARSA learning. However, Q-learning uses an off-policy learning approach, updating the Q-value function using the maximum expected future reward

3. Hypothesis

Hypothesis 1: We epxect the Q Learning algorithm to perform better overall than the SARSA algorithm over 10000 episodes.

Hypothesis 2: For both algorithms, we expect exponential decay of ϵ to perform the best, as in, have the highest average reward over 10000 episodes.

Both Airi and Khoi's experiments will be performed in accordance to this hypothesis.

4. Experiments Done

4.1. Airi's Experiments - Q-Learning

4.2. Khoi's Experiments - SARSA Learning

For SARSA Learning, Khoi is also implementing an ϵ greedy method. For his setup, the hyperparameters are: $\epsilon = 0.5$; $\gamma = 0.9$; and $\alpha = 0.5$

Addtionally, Khoi is also implementing three differement decaying methods for the ϵ hyperparameter. These methods will be explained further in the **General Analysis** section, with **Hypothesis 2**.

For one run of 10000 episodes, with ϵ kept static at 0.5, here are his current results. Note that due to the random nature of taking actions, each run will produce a different graph.

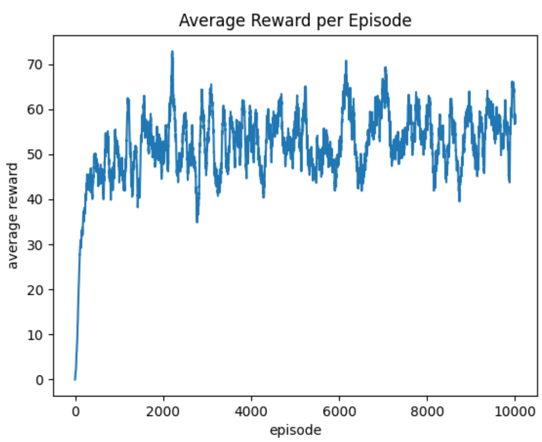

Figure 1. SARSA Results Over 10000 Episodes, Episode-related decay

Looking at the graph, we can see that the average rewards reached a plateau of approximately 60 after about 1000 to 1500 episodes, this shows that the agent is indeed being trained to take more optimal actions as the number of episodes increased.

5. General Analysis

5.1. Hypothesis 1

For hypothesis 1, we compared the performance of Q Learning and SARSA learning, over a static ϵ value of 0.5

Looking at the figures, we can see that [...]

5.2. Hypothesis 2

We explored three different methods of ϵ decay. First, ϵ is kept static at a value of 0.5. Second, ϵ followed the decaying function of $\epsilon = 0.5 * \frac{1}{no.of episodes}$. Finally, ϵ follows the exponential decaying fucntion of $\epsilon = 0.5 * e^{-0.001 * no.of episodes}$

For SARSA, comparing the three methods of ϵ decay, yields the following results:

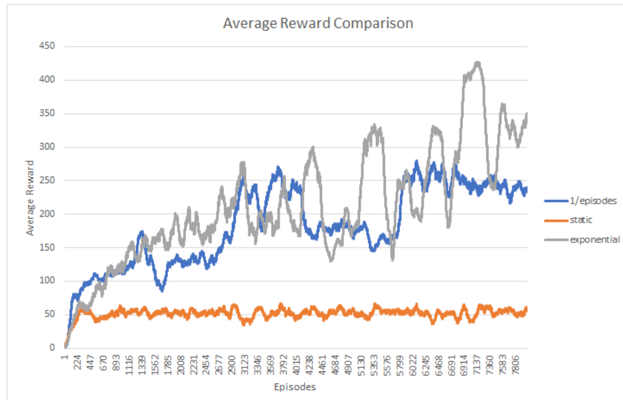

Figure 2. Average Reward of SARSA, Comparing DIfferent ϵ Decay

Additionally, the mean and standard deviation for each method are as follows:

Method	Mean	Standard Deviation
Static	52.07	7.545
1/Episodes	171.9	15.454
Exponential	207.45	80.459

Table 1. Mean of Average Reward and Standard Deviation

As these results show, for the SARSA algorithm, exponential decay of ϵ performed the best.

Next, for Q Learning, the comparison graph is as follows:

6. Comparison to Other Methods

7. Future Work