Reinforcement Learning Project 1: Q learning and SARSA Algorithms for a Modified Wampus World

Gouthamrajan Nadarajan

1 Part 1: Defining RL Environments

1.1 Deterministic Environment Formulation

A modified version of the Wampus World was used for my work. My environment definition for wampus world had four different obstacles which included three pits and one wampus. The agent was punished if it fell into a pit with a reward of -100 and was punished if it went to the wampus square with a reward of -500. The agent wins and the game ends if it reaches the final goal state where it receives a reward of +1000. In this example, the agent has four possible actions that it can take: move up, move down, move right, and move left. The random agent picks an action with uniform probability and the environment is deterministic where $p(s', r \mid s, a) = 1$ for all cases except on the boundaries where an action that results in the agent moving of the grid results in the agent just staying in the same place. The agent could take a maximum of 10 steps to reach the goal through its random actions before the environment would terminate.

The wampus world grid is shown below in **Figure 1**. Here the agent is located in the (0,0) position. The pits are located in (2,0), (2,2) and (3,3) positions. The Wampus is located in the (0,2) position and the goal state is located in the (1,2) position.

Please refer to **assignment_1_deterministic_part_1.ipynb** to see the deterministic environment implementation.

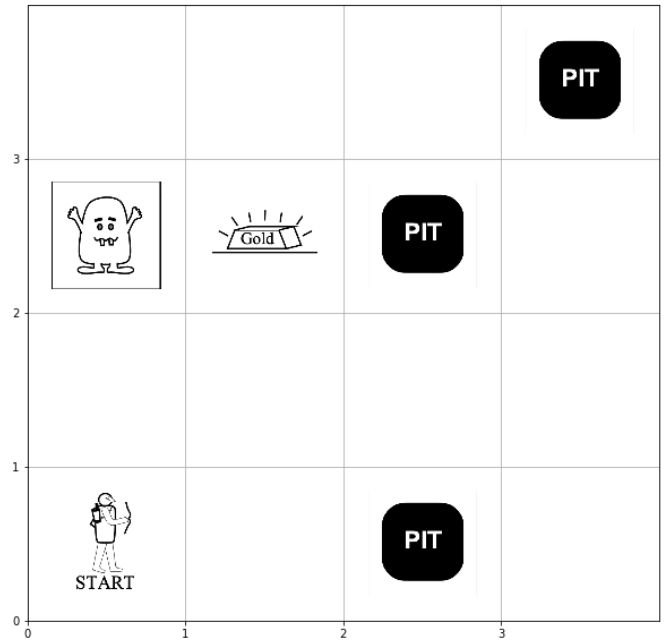

Figure 1: Illustrates the Wampus world environment. Please refer to assignment_1_deterministic_part_1.ipynb to visualize the per step renderings of the environments. This rendering was disabled (commented out) in other scripts due to the need for computational efficiency.

1.2 Stochastic Environment Formulation

In this case of the stochastic environment definition, we keep the same environment as shown

in **Figure 1**, but we change the actions that the agent can take. In this new scenario the agent can take five different actions which include: move up, move down, move right, move left, and stay put (do nothing). To make this a stochastic environment, we have a scenario where for each of the actions, $p(s', r \mid s, a) ! = 1$. Instead there is a 0.2 probability that the agent stays put at its own location when taking any action. In my code solution, this is incorporated where a random number is sampled from a range of 1 to 10 with uniform distribution where if the number is >8, then the agent does not take the desired action and instead stays put in its current location. Please refer to **assignment_1_stochastic_part_1.ipynb** to see the implementation of the stochastic environment. Note that the rendering for this approach was done using google widgets and not the Matplotlib library as done for the other assignments.

A common concern in defining environments is the safety of the agent. Care must be taken as to not let the agent learn an undesired pattern by maximizing certain rewards over others. This can be prevented by increasing the discount factor parameter (which would allow the model to focus on long term rewards more than short term rewards). Another way to combat this is to add a small negative reward for each step that is taken which will force the agent to learn the most optimal path. These are illustrated in part 2 of the assignment where I apply tabular methods to the wampus environment.

2 Part 2: Applying Tabular methods: Q learning and SARSA

In this part of the project, I applied both Q learning and SARSA tabular methods. These are applied for both the stochastic and deterministic environments. Please refer to **assignment_1_part_2_deterministic.ipynb** to see the deterministic model implementation and refer to **assignment_1_part_2_stochastic.ipynb** to see the stochastic implementation.

Q learning involves using an bootstrapping method that estimates the state action value for each possible combination of actions and states. This update function for this method is shown in Equation 1 below

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a) - Q(s, a)]$$

Equation 1: Q learning update function

The SARSA algorithm is a modification of the Q learning algorithm where we follow our policy to obtain the next state instead of simply taking the greedy action. The SARSA update function is shown below in Equation 2.

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r_{t+1} + \gamma\, Q(s', a') - Q(s_t, a_t)]$$

Equation 2: SARSA update function

The difference in SARSA from Q learning is that $\max_a Q(s', a)$ is replaced by $Q(s', a')$. Therefore, instead of just looking up the Q value that corresponds to the max action in our Q table, SARSA learns action values according to its policy. This makes SARSA an on-policy whereas Q learning is an off policy method. SARSA is considered a more conservative algorithm compared to Q learning because it does not use the greedy estimate and instead follows its policy.

For the deterministic environment, the cumulative reward graph for the deterministic environment is shown in **Figure 2**. The environment for this method is the same as the one described in part 1 of the report. We used an linear epsilon decay function with a decay rate of $\varepsilon = 0.05$. The agent is structured to take the greedy action with a probability of $p(1 - \varepsilon)$. Therefore, during earlier episodes, the agent explores more and takes random actions but in later episodes the agent increasingly takes the greedy action that maximizes the reward. The epsilon decay function which was used for the deterministic and stochastic environments for both the SARSA and Q learning

algorithms is shown in **Figure 3**. I also used a learning rate of 0.01 and a gamma value of $\gamma = 0.75$, max steps per episode=10 and total episodes=100 for all experiments. These parameters were determined mostly through trial and error.

The reward structure for part 2 of the assignment was modified due to unstable agent training. Here each pit carried a -5 reward, the wampus carried a -10 reward, and reaching the goal state resulted in a +100 reward. I also added a -1 reward for each step that the agent took. This was in order to motivate the agent to find the fastest path to the goal state.

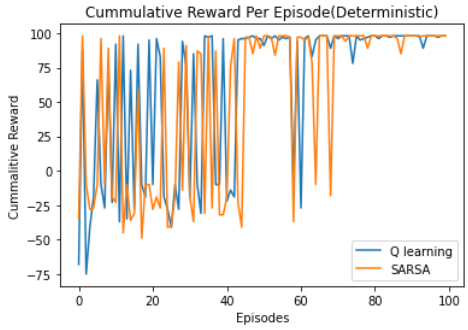

Figure 2: Cumulative reward per episode for deterministic Wampus World environment. Compares both SARSA and Q learning methods. Here we see that Q learning shows a somewhat faster convergence than SARSA.

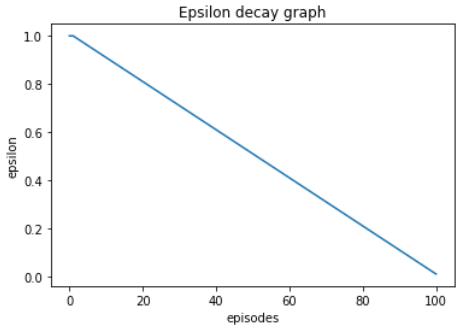

Figure 3: Epsilon decay graph

For the stochastic environment, the same environment architecture was kept as in part 1 but with the modified new rewards. For the stochastic environment, the cumulative reward graph is shown in **Figure 4**. We can see that the SARSA algorithm converges significantly faster than the Q learning algorithm. One possible reason for this could be due to the SARSA algorithm being more conservative and thus learns more effectively in the earlier episodes which thus allows it to converge faster by avoiding the costly negative rewards in later episodes.

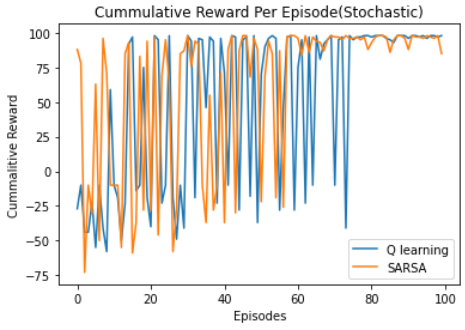

Figure 4: Cumulative reward per episode for stochastic Wampus World environment. Compares both SARSA and Q learning methods. Here we see that SARSA shows significant faster convergence than Q learning.

3 Conclusion

For assignment 1, I explored developing RL environment and solving those environments using Q learning and SARSA. I saw that for the deterministic environment, the Q learning and SARSA performed similarly. For the Stochastic environment, the SARSA method converged faster than the Q learning algorithm which I attribute to SARSA being more conservative in nature and avoiding risky actions.

4 Acknowledgement

The object-oriented setup of the environment follows the openAI Gym structure. It was also motivated from the TA practical session presented by Nitin Kulkarni.

The rendering(using google widgets and Matplotlib) and the Wampus World grid was also motivated from the TA practical session presented by Nitin Kulkarni