Queen Mary University of London

Department of Electrical Engineering & Computer Science

ECS7002P – Artificial Intelligence in Games

Assignment 2 Report

Luke Abela - 200588919

Hasan Emre Erdemoglu - 200377106

Suraj Gehlot - 200469881

11 December 2020

Notes on the Assignment:

The implementation of this code and the report can also be found in GitHub, using the following link:

https://github.com/heerdemoglu/ECS7002-FrozenLake

The same structure follows in this repository.

Question 1: Code Organization

The code is organized into three folders and two scripts that is responsible in running the small lake (*RunEnvironmentSmall.py*) and big lake (*RunEnvironmentBig.py)* environments. It uses Listing 5, given in the assignment document altered for small and big lake, respectively.

These main methods will also have code which will output numerical answers described in the report.

More details on folder structure can be found below:

- **deprecated:** Contains the initial listings and implementations of the assignment. This is later changed to the structure at the time of submission. This code is not used in the implementation directly. It has been refactored and used within folders **environment** and **learning_methods**.
- **environment:** Contains separate scripts for *Environment*, *EnvironmentModel* and *FrozenLake* environment. *Auxillary_Functions.py* is used to construct grid maps in the *FrozenLake* environment. Note that, *FrozenLake.py* implements any grid environment with slips, along with transition probabilities and the rewards associated with the environment.
- **learning_methods:** Contains three separate scripts, *TabularModelBasedMethods.py*, *TabularModelFreeMethods.py* and *LinearWrapper.py* to implement sections 2, 3 and 4 of the assignment document respectively.

Question 2: Value and Policy Iteration for Big Frozen Lake

To find an optimal policy for the big frozen lake, policy iteration required 6 iterations. To find an optimal policy for the big frozen lake, value iteration required 19 iterations. Hence, the policy iteration algorithm was faster.

Question 3: SARSA and Q-Learning for Small Frozen Lake

It is assumed that algorithms are converged to optimal policy if the difference between the value of the algorithm and the optimal policy is less than 0.1 for all states. Meanwhile, the number of episodes change in each run; Q-learning takes around 500 episodes to converge whereas SARSA takes all 2000 episodes; which is the computational limit for this question. It took around 14000 episodes for SARSA to converge to the optimal policy when the computational budget is increased.

Bonus Question 1: Linear Function Approximation (LFA) Implementation

$\phi(s, a)$ is a feature vector which has a dimensionality of "number of actions" by "number of features", where number of features is calculated by number of actions times number of states.

In this question, the number of features is sparse enough to represent state-action space in one-hot encoding. Therefore, for each set of actions available in a specific state; a unique set of features can be extracted. Encoding can be done on the fly, without needing to store all encoded values within the environment.

Parameter vector θ has the same dimensionality with the number of features. It can be interpreted as the overall value of a state-action. One-hot encoded feature vector $\phi(s, a)$ can be used as a selector (using dot product) to filter values of the unrelated state-actions. Through training, this parameter vector can be updated, again utilizing the feature vector to correctly update the encoded state-values while not altering unrelated state-action values.

When the environment is finite; this non-tabular approach does not hold every-state action value and policies in memory; it encodes the state in question on the fly. Parameter vector holds general information about what the values of the states and the policies that must be taken, which can be decoded to get the actual state-action values for a state using one-hot encoded feature vector.

The tabular approach is a special case of this algorithm as the table is initially constructed before starting the process. Here, individual state-values are not kept in memory and is calculated by using the feature vector encodings on the fly. Tabular form is a special case, where encoded values are individually kept in memory.

Bonus Question 2: Optimal Policy for Big Frozen Lake Using SARSA and Q-learning

For the big frozen lake environment SARSA and Q-learning cannot find the optimal policy because the state space is too large. Note that both SARSA and Q-learning algorithms update the values of the path that they take when the reach to the terminal state, discounting the value at every step from terminal state to initial state. Therefore, in the path, the states that are closer to the goal have larger values, which decay as the states get closer to the starting point.

If the path becomes too large, the algorithm will not be able to accumulate values in states which are closer to initial states. Therefore, the results for these steps will not be updated efficiently; therefore, these algorithms will not be able to learn optimal policies. It is seen that for the large lake environment; all values are returned as zeros, as initialized in the first episode; implying that the algorithm was not able to learn anything from the environment.

These algorithms work for the small frozen lake because the state space is not too large, a smaller set of actions are available from initial to terminal state. Here, discounts affect the value of the states which are closer to the initial states less, so that these algorithms will be able to successfully construct a path from initial to final state using the state-action values. The values accumulated in each episode constructs the final policy of these algorithms.