Queen Mary University of London

Artificial Intelligence in Games

Group Assignment 2 Report

Sanjay Anandakrishnan Venkatesh
Robert Andrew Ellis

Question 1: Code Organization

The code for the environment was organized in five new functions. These were calc_probs, check_probs, calc_rewards, and act. The function calc_probs steps through all of the possible states and calculates all of the probabilities for the states. This is also where the slip chance is factored in. The function check_probs then loads the target probabilities from ‘p.npy’ and compares them with the calculated probabilities in order to check their correctness. The function calc_rewards then steps through all of the states and calculates the rewards for them, setting the reward to 1 if the agent is in the goal state. Finally the function act is used to carry out actions during the probability calculation. First four if statements check if the chosen action is valid, making sure that the agent is not trying to move off the edge of the lake. If the action is invalid then the current state is returned. If the action is valid then the resultant state is determined and returned.Overall the code structure followed that given in the brief quite closely. Most of the implementation was done in the areas commented as # TODO. The main addition to this was an additional class created called epsilon_greedy_selection. This class carried out action selection with use of epsilon greedy selection. A random number between 0 and 1 was generated, if the number was lower than the current epsilon value then a random action was selected. If, however, the number was greater than or equal to the current epsilon then the optimal action was selected. This allows the algorithms to carry out a certain level of exploration when formulating policies. The final addition was that of an ‘if name == “main”’ statement so that when the file is run the main function is called.

Question 2: Value and Policy Iteration for Big Frozen Lake

The policy iteration required 6 iterations to find the optimal policy for the big frozen lake . To find an optimal policy for the big frozen lake, value iteration required 21 iterations. This shows policy iteration was faster compared to value iteration .

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

If the optimal policy is less than 0.1 for all states then it is taken that algorithms are converged to optimal policy. Q-Learning takes 1237 episodes to converge whereas SARSA converged on 11042 to the computational limit . When the computational budget is increased it takes increase in episodes for SARSA to converge to the optimal policy.

Question 4 : Linear Function Approximation

The feature vector with a dimension of "number of actions" divided by "number of features," where number of features equals number of actions multiplied by number of states.The number of features in this query is small enough to represent state-action space in one-hot encoding. As a result, a unique collection of characteristics can be derived for each set of actions possible in a given state. Encoding can be done on the fly without requiring the environment to store all encoded values.The dimension of the parameter vector theta is the same as the number of features. It can be thought of as the total worth of a state-action. A selector (using dot product) can be used to filter values of unrelated state-actions using a one-hot encoded feature vector. This parameter vector can be changed through training, with the feature vector being used to update the encoded state-values while leaving irrelevant state-action values alone.When the environment is finite, this non-tabular technique does not save every-state action value and policy in memory; instead, it encodes the state on the fly. The state-action values for a state may be decoded using one-hot encoded feature vector from the parameter vector, which holds general information about the values of the states and the policies that must be followed.The tabular method is a subset of this technique in which the table is built first and then the operation begins. Individual state values are generated on the fly using feature vector encodings, rather than being stored in memory. A specific situation is tabular form, in which encoded values are retained in memory separately.

Question 5: Optimal Policy for Big Frozen Lake Using SARSA and Q-learning

As the state space is too large the SARSA and Q-Learning could not find the optimal policy. Both the algorithms update the values of the path that they will take when it reach the terminal state , slashing the value at every step from terminal state to initial state . As a result, the states closest to the goal have greater values on the path, which decrease as the states come closer to the starting point.On the off chance that the way turns out to be too enormous, the calculation can not gather values in states which are nearer to beginning states. Subsequently, the outcomes for these means won't be

refreshed productively; therefore
these algorithms will not be able to learn optimal policies. All variables are returned as zeros for the large lake environment, as initialised in the first episode, meaning that the algorithm was unable to learn anything from the environment.These algorithms work for a small frozen lake since the state space isn't too big, and a smaller set of actions is available from the beginning to the end. Discounts have a smaller impact on the value of states that are closer to the starting states, allowing these algorithms to successfully design a path from the beginning to final state using state-action values. The final policy of these algorithms is built from the values accumulated in each episode.