COMP 4106 Assignment Three

Connor Raymond Stewart (101041125)
Gabrielle Latreille (101073284)
Abdaljabbar Hersi (100919600)

Monday, April 12, 2021

Matthew Holden
CARLETON UNIVERSITY

1 – Abstract

The report herein describes a *simulated* computational experiment for Carleton Universities COMP 4106 - Artificial Intelligence course. This assignment introduces the topic of implementing model-free reinforcement learning to determine the outcomes of specific states. Such AI tools allow for the creation of autonomous decision-making in computers without using a predefined model for a defined task environment. The task environment in question is a grid-based map consisting of five grids arranged in a plus-sign shape, with an event space operating in a discrete event loop-based time-space. The environment represents a room space where an AI-operated vacuum cleaner attempts to keep all the tiles in the grid space clean for as long as possible. The grid space continuously reassigns the tiles within the grid to be dirty based upon a random probability value, and the AI must *learn* the optimal method to keep the tiles clean for a maximal amount of time. For the assignment implementation, the simulation environment exists as a finite Markov decision state at which point a Q-learning model finds an optimal policy - that is to say, an optimal sequence of movements for the vacuum cleaner - to determine the maximization of the expected clean-ness of the tiles in the grid. The computational experiment results were a success, and the results of the experiment and all relevant implementations will be discussed further in the report herein.

2 – Introduction

The report provides feedback on a self-directed project on temporal difference Q-learning to allow a program to learn the optimal policy for a vacuum cleaner. The vacuum cleaner exists in a hypothetical grid-based environment shaped like a plus-sign, with five squares of equal size being assigned a number for designation. The vacuum cleaner in the grid starts on the square designated as square one and can perform various actions such as cleaning the square, moving horizontally along the squares, or moving vertically along the squares. Each square in the grid can have a binary state of either dirty or clean, and all squares in the grid space are fully observable at all moments in time. The vacuum knows its current position at any unit of time and if any square is dirty or clean.

The environment operates in a discrete-time environment, meaning the vacuum cleaner operates in a discrete event loop. As per the functioning of a discrete event loop, all the vacuum cleaner's decisions are digital rather than analog, and time cannot get divided into smaller bits than a single iteration of the main loop represents. Therefore, the AI operates in a simplified time-space wherein no actions can occur between iterations of the program's loop. Each time the main loop passes an iteration

(or a tick in the simulation time), there is a probabilistic chance that a squares state will change from clean to dirty.

The project assigned resulted in the successful completion of the main task presented in the course. The AI was able to achieve a testing accuracy of 100% for the testing script provided. The report herein will be more rigorous details on the design, implementation, and results of the computational experiment created for the assigned task.

3 – Datasets & Implementation

Multiple parameters for the assignment are predefined as per the specifications of the assignment. These specifications include the string representation for the states (see the form PS1S2S3S4S5) where P represents the square number the vacuum cleaner is in and Si = 0 if the square i is clean; 1 if the square i is dirty. The program has an action-space represented by an internal matrix represented by Python lists, with the following string representation for actions. The actions exist in a column space and are represented as C for cleaning the current square, L for moving left, R for moving right, U for moving up, and D for moving down. Hyperparameters for the Temporal difference Q-learning formula are predefined as per the assignment specifications, with a Gamma of 0.5 representing the discount factor and an Alpha of 0.1 to represent the learning rate. The initial estimate of the Q-function was defined as Q(s, a) equaling zero, as per the assignment's specifications. The reward for each state was calculated as -1 * number of dirty squares, and the Temporal difference Q-learning formula used is shown below:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\left(r(s_t) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right)$$

The assignment3.py files contain a td_qlearning class with 9 membership functions: “__init__”, “qvalue”, “policy”, “open_file”, “reward”,” get_q_value”, “get_action_index”, “get_max_state”, “q_function_learning”. The “__init__” function is a constructor that takes one input argument, which is the full path to a CSV file containing a single trajectory through the state space. The constructor also calls several other memberships functions, such as the open_file function, which takes the CSV file and turns it into an array where each index contains the state-action pairs and the q_function_learning function, which is called for each state-action pair in the trajectory and iteratively updates the estimated Q-value based on the state-action pairs.

The reward function calculates the reward given using the formula specified above. Looking at the membership functions get_action_index and get_max_state, we

see these are helper functions that return the current action taken by the vacuum and the max state value given different possible actions based on the current state. The program dynamically appends actions to the action matrix space based upon the activities taking place and is assigned a position in the matrix based upon the predefined actions column space.

Using all the membership functions above that implement the above formula, we accurately created a class that learns using the specified trajectory in the CSV files. The application is functionally able to read input from the professor's assigned testing script and reproduce results as per the testing script's specifications.

4 – Results & Questions answered

The Implemented algorithm achieves 100% on the testing script and accurately estimates q-values based on the provided trajectory. Solutions to the provided questions in the assignment are given below:

1. *What type of agent have you implemented?*

   The type of agent we have implemented is a utility-based agent; this was implemented by optimizing the q-value as our agent continued to learn.

2. *Suppose there is a state-action pair that is never encountered during a trajectory through state space. What should the Q-value be for this state-action pair?*

   If a state-action pair is never encountered during a trajectory through state space, the Q-value for this state-action pair will be 0. The reason for this is that we initialized the first estimate of a state-action pair as 0. Also, we note that the above mentioned in the assignment as an instruction.

3. *For some cases (even with a long trajectory through state space), the optimal Q-value for a particular state-action pair will not be found. Explain why this might be the case.*

   The above might be the case because the state-space and action pair are enormous, and the amount of iteration needed varies for some cases. To get a sense of the scope of the state-space - since the state gets represented as PS1S2S3S4S5 - we note each state can be clean(0) or dirty(1), so this can be represented numerically as (2^5). Also, there are five different positions that our agent can reside so (2^5 * 5); now, you also need to include the five different actions that the program can take, and the number of iterations needed to optimize the Q-value of a specific

state-action pair. As can be seen, the sheer number starts to get absurd, which is presumably why the optimal Q-value for a particular state-action pair is indeterminable.

4. *In the test cases provided, the trajectories through state space were generated using a random policy. Describe a different strategy to generate trajectories and compare it to using a random policy.*

A different strategy would be to use State-Action-Reward-State-Action or SARSA. The idea is that we use a policy that is already defined. For example:

p(s) = 'C' is current square is dirty
p(s) = 'L' if current square is clean and 4
p(s) = 'R' if current square is clean and 2
p(s) = 'U' if current square is clean and 5
p(s) = 'D' if current square is clean and 1
p(s) = random if current square is clean and 3

Using the policy, we can calculate the q-values with the function below.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Where the at+1 in Q(st+1, at+1) is calculated from the policy.

5 – Discussion and Conclusions

In conclusion, the assignment resulted in an overall success for the group. The assigned tasks were completed as the assignment specified, with results satisfying the provided testing script. Some problems were encountered during the testing phase as we had difficulty connecting the program's trajectory to the trajectory assigned within the input CSV, but this was finally worked out before the deadline.

6 – Statements and Contributions

Overall, the project was completed in group meetings through discord. All group members made significant contributions to the overall project, with approximately equal contribution among each member. Since the group work was evenly divided amongst members and completed in a shared environment, all members of the group completed all aspects of the assignment together.