Data 640 Summer 2022

Professor Steve Knode

Tiania Chan

tiania.chan@gmail.com

Assignment 5b: Q Learning using numpy

**Introduction**

The goal of this project is to use Q Learning implemented using Python and Google Colab to create a learning model that can find the most optimal path to transverse a “board” with obstacles given the starting location and end location. The analysis uses code from the article on reinforcement learning by Sayak Paul ([https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/](about:blank) ) with additional modifications to test use cases.

Code Overview:

Dependencies are libraries are declared at the very top of the code. Since the model will be using the library numpy, this is first declared. Next the learning rate and the discount needed for the Q Learning equation is declared. These will be changed later in the code to see if there is any affect to the model. The next few cells in the variations in the learning rate and discount to see the difference in the results of the model when these values are changed.

The sixth cell is where more dependencies are declared. To start off, the dictionary, location\_to\_state is declared so that the Q Learning equation can be used to calculate the numerical values of each location in the “board” that the model is traversing to find the most optimal path. Since there are 9 locations starting rom L1 to L9 and Python begins its indexes at 0, each value is mapped from 0 to 8. Next, the actions, or what location the model will take after its current location is declared in the list variable, actions. Next, the rewards are declared. This is a numpy array which references the locations L1 through L9 and what actions the model can take. Where the array contains a 0, the model will not take this action as it is discouraged. Where there is a 1, the model is more likely to take that action. Next, the variable state\_to\_location is an array to map the index values to the array for the model to iterate through. The next cell is to allow the model to go from L2 to L5. Since the rewards array did not contain a 1 for the model to go from L2 to L5, it would keep bouncing back between L1 and L2 until the reward array was updated.

The next cell contains the main function that will be called for the model to find the most optimal path through the board given user input for the desired start location and end location. The first part of the method is to copy the rewards array. This lets as many variations of the model to be run without affecting the original rewards array unless manually modified. Next, retrieve the ending state according to the dictionary location\_to\_state to store for use further in the method. This will be necessary to see if the end location has been reached or not. In the next line, by setting the reward to 999 for the end location value, the model will always be striving to reach the end location.

The next part of the function begins the q learning process. Start by declaring a 9 by 9 array of the board. Next, since Q learning uses exploration of the environment, set a high number of iterations that the agent can go through to learn its environment. In this case, it was set to 1000 but this will always depend on the model and the environment, larger and more complex environments will require more possible iterations. Here, the agent will pick a random location using np.random.randint(0,9) to stay within the board. From there, actions taken by the agent is recorded in the variable playable\_actions. In the next for loop, so long as the agent is within range of the board, if the current state is greater than 0, record the action as this is a possible desired action. Next, choose another action for the agent to take.

Next is the Q Learning equation. Mathematically, the equation is as follows:

Text

Description automatically generated with low confidence

Where the agent will look at the reward and add the product of the value of the action and the discount rate. Each time the agent moves further from the desired location, the discount lessens the reward so that the agent is always drawn toward the higher rewards. Next, using the Bellman equation the Q Value is updated. Once all the Q values have been stored for each chain of actions, the agent is finally ready to begin finding the optimal path.

Still in the same function, set the route as a list with the first index value as the start location. Next location is yet unknown so for now, the value is the start location of the agent. Then while the next location is not equal to the end location, the model will find the starting state, find the maximum Q value of the next possible action, take the maximum Q value to determine the next location and append the next location to the list of the route that the agent will take and the start location is updated to the next location. Once the end location has been reached, the model breaks out of the while loop and returns the optimal route and the number of steps taken to reach the end location. The next print statements display the discount value, the learning rate, as well as the results of the optimal route based on starting and end locations. The next few cells are there to show the thought process of identifying why the original agent was able to find the optimal route from L9 to L1 but was stuck if trying to find an optimal route from L1 to L9.

The next few cells contains the alterations in the code necessary for a new model to traverse a “board” containing 10 locations instead of 9 locations. Among the changes were as follows:

* An additional state to location\_to\_state
* An additional action to actions
* Additional row and column to the rewards array
* Initializing a 10 by 10 “board” instead of 9 by 9
* Generating a random number between 0 and 9 to allow the agent to look at 10 locations
* Increasing the range of the for loop from range (9) to range(10)

From there, the results for finding the optimal route between L10 to L1 as well as L10 to L4 are in the final cells.

**Answers to Prompts**

Prompt 3:

In prompt 3, the question is why the optimal route for the original reward table to go form L9 to L1 is L9 -> L8 -> L5 -> L2 -> L1. This is because the optimal route function takes the start and end location, sets the value of the end location to 999 and then finds the Q values for each route possible for the agent or model to take in the “board”. Each iteration then takes the maximum Q value until the agent finds the most optimal path. In the case of L9 to L1, each next location had a value of 1 before reaching the final value of 999 which was the desired goal of the model.

Prompt 5:

In prompt 5, the question is to runt he model with different parameters for gamma and alpha or for different values of the discount and learning rate.

Prompt 6:

Prompt 7:

Prompt 8:

Prompt 9:

**Conclusion and Take Aways**

**References**

**Code Appendix**

References

Rudes, R. ( 2020, Sept 8). An Introductory Reinforcment Learning Project: Learning Tic-Tac-Toe via Self-Play Tabular Q Learning. Retrieved from https://towardsdatascience.com/an-introductory-reinforcement-learning-project-learning-tic-tac-toe-via-self-play-tabular-b8b845e18fe