Q-learning with the Tower of Hanoi

Mike McGrath
CS 445/545
Winter Term, 2019

Introduction

The Towers of Hanoi is a well known puzzle that was invented by French mathematician Edouard Lucas in 1883[1]. It consists of three pegs and six to nine disk of increasing diameter. The disks start on one peg in a pyramid shape with the disks stacked largest to smallest. The goal of the puzzle is to move all the disks from one peg to another without having a larger disk cover a smaller one. The puzzle has been solved mathematically for any given number of disks and there exists several different algorithms to solve the problem.

The puzzle can be solved in a minimum number of moves. For n number of disks the puzzle can be solved in $2^n - 1$ moves[2]. For three disks it takes seven moves to solve the puzzle, for four it takes 15, and for 64 disks it takes $2^{64} - 1$ or 18,446,744,073,709,551,615 moves (Miodrag). Taking one second to make a move it would take 585 billion years to complete the puzzle!

Further complicating the puzzle, the state space for the puzzle grows even faster. The total state space is P^n where n is the number of disks and P is the number of pegs. The state space for three pegs and two disks is nine and for three pegs and six disks it is 729. While the state space grows exponentially, most of the states are not legal moves from one to another. For example, with three disks there are 27 possible states and therefore 27^2 or 729 moves, but the

[1] Hofstadter, Douglas R. Metamagical Themas: Questing for the Essence of Mind and Pattern. Basic Books, 1985.
[2] Petković Miodrag. Famous Puzzles of Great Mathematicians. American Mathematical Society, 2013.

actually number of legal moves is under 100. For six disks there are over a half a million entries on the Q-table, but approximately 2000 transitions that are legal moves.

While numerous algorithms have been used to solve the Towers of Hanoi such as iterative[3], recursive (Hofstadter), and breadth first search[4] this paper will explore using Q-learning to find the optimal policy to solve the puzzle in the fewest possible moves. Q-learning is a model free reinforcement learning algorithm that trains an agent to take certain actions based on an expected reward. In the case of the Towers of Hanoi, the reward is moving a disk to the final state that solves the problem. Through backpropagation the reward is spread through the Q-table.

My hypothesis is that, since the goal is to find the minimum number of moves, the most important hyperparameter will be the epsilon value. The algorithm should be more efficient if it can explore the state space more often than exploiting the best rewards. Alpha and and Gamma should not be as important to the algorithm as long as they are not set to the extremes of 1 or 0 since the algorithm will still learn.

Methods

The Towers of Hanoi was implemented in the Python programming language as a class. The class contained the methods to create a new board with the desired number of pegs and disks, the ability to move the disks, a checker that checks if a move is legal, and a solver to check if

---

[3] Mayer, Herbert, Perkins, Don. Towers of Hanoi Revisited A Nonrecursive Surprise. San Diego State University Dept. Of Mathematical Science, 1984.
[4] Masum, Hassan, Houston, Ben. *Explorations in 4-peg Tower of Hanoi.* http://service.scs.carleton.ca/sites/default/files/tr/TR-04-10.pdf, 2004.

the puzzle is solved. In addition, the class has a method to create all possible state spaces and start from any state space to explore possible moves. From these methods, a reward table of all state transitions was constructed with each transition having an initial reward of zero. If a transition resulted in a solved state, the reward value was set to 100.

The Q-learning algorithm takes five parameters. The reward table, the learning rate alpha, the discount factor gamma,the exploration vs exploitation rate epsilon, and the number of episodes to run. The reward table only changes if the original board changed, i.e. adding more or less pegs or disks. The learning rate, alpha, controls how much the Q-value is updated after each move. The discount factor, gamma, controls how much to value future rewards by looking at the max reward for the next state. The epsilon value controls how much the algorithm explores the state space at random, by randomly selecting a legal move, as opposed to selecting the move that has the greatest reward. Lastly, the user can specify the number of episodes to run before returning. An episode is defined as all the moves it takes for the algorithm to go from the start of the puzzle to the solved state. The number of moves per episode can vary depending on if it randomly choices the shortest path, if it cycles between states, or if it makes unnecessary moves.

The learning algorithm has a unique conditional to make sure that the algorithm runs enough to create a policy. It will run the number of episodes passed by the user, but will continue to run until the Q-value from the initial starting position of the puzzle to another state has at least one positive value. This prevents the algorithm from randomly selecting an illegal move since all the values would be zero. An alternative would be to make all illegal moves a negative value and all legal moves initialized to 0.

To test the data, the algorithm returns the Q-table and the number of episodes need to train the Q-table. A policy was then constructed from the Q-table and the puzzle is solved by using the policy. From this, the number of moves need to solve the puzzle is returned.

For experiment were run with multiple trials. For the experiment on alpha, the algorithm runs with 18 different values of alpha increasing by .05 from a range of .05 to .9. All other values are kept the same at gamma = .9, epsilon = .9 and the number of episodes until the Q-table had one non-negative value for each row. The algorithm runs in a loop 50 times so that the results can be averaged. The experiment for the gamma hyperparameter is the same as the alpha experiment except alpha is held constant at .9. For epsilon, the test was similar. Alpha and gamma were constant at .9 and the number of episodes was varied by the number of times the algorithm need to run to fill the Q-table. The only difference is epsilon ranged from 0 to 1 by .05 and was run 20 times.

The experiment for number of episodes was different. Instead of running only 18 or 20 times, the algorithm is tested 30 times with the minimum number of episodes ranging from 0 to 29. However, since the algorithm runs a minimum number of times to return a Q value for each row, the algorithm normally ran for a minimum of 7 episodes. While the values of alpha and gamma were kept constant at .9, the value of epsilon was .1 for one trial and .9 for another. This was done, to see the difference in the number of episodes need to train the model with different epsilon values.

All the data was then graphed to visually view it. The graphs and discussion of the graphs will follow in the next sections.

Results

The first experiment is to see how the number of episodes affected the number of moves needed to solve the puzzle. Two trials are run with the values of alpha and gamma set to .9 and epsilon either .1 or .9. The results are displayed in Figure 1.

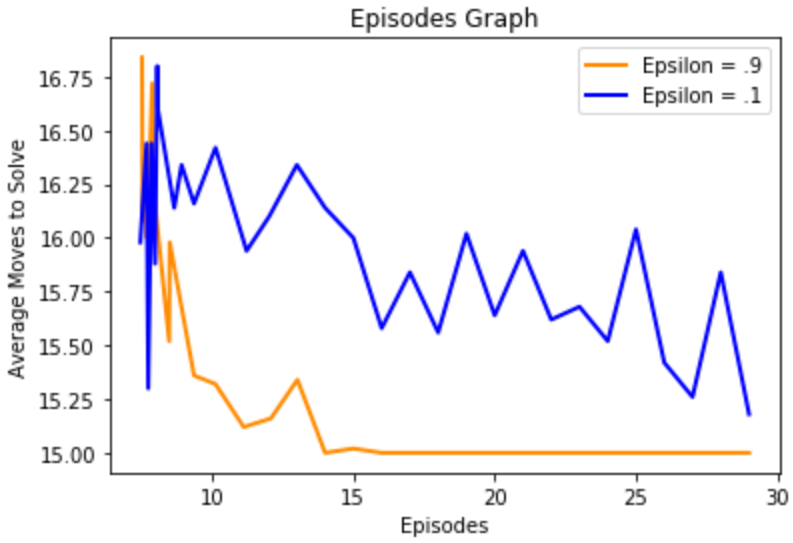

Figure 1

The next experiment is to see how the value of alpha affects the number of moves needed to solve the problem. Two trials were run with alpha ranging from .05 to .9 by an increase of .05. The difference in the trials was the minimum number of episodes needed. The first trial ran until the first row of the Q-table had at least one non-zero number. The average number of episodes

ranged from 7.38 to 8.24 with no clear pattern. The second trial ran when the first row of the Q-table had a non-zero number and at least 10 episodes were finished.

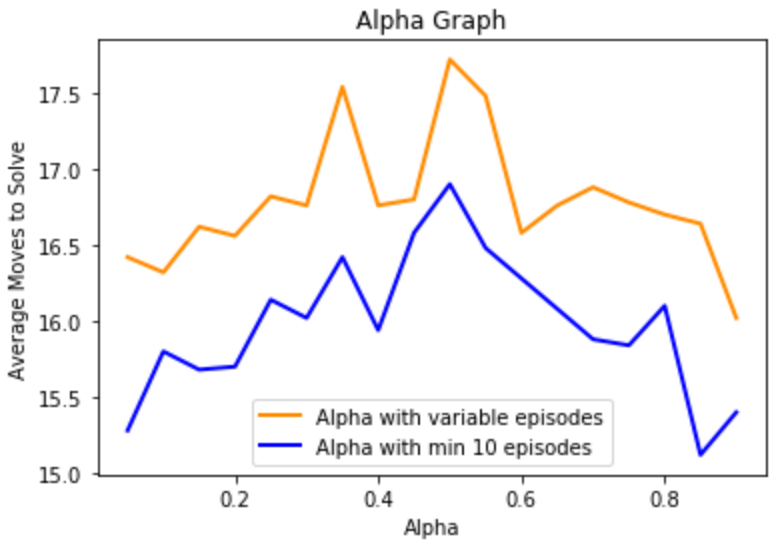

Figure 2

The gamma hyperparameter is tested in a similar manner and the results are included in Figure 3. The average number of episodes ranged from 7.68 to 8.22 and also had no noticeable pattern. For both alpha and gamma, when the minimum number of episodes was set to 10 the average number of episodes was between 10.02 and 10.55 (implying that it always took at least more than 10 trials in at least one of the 50 iterations to find a non-zero value in the first row of the Q-table).

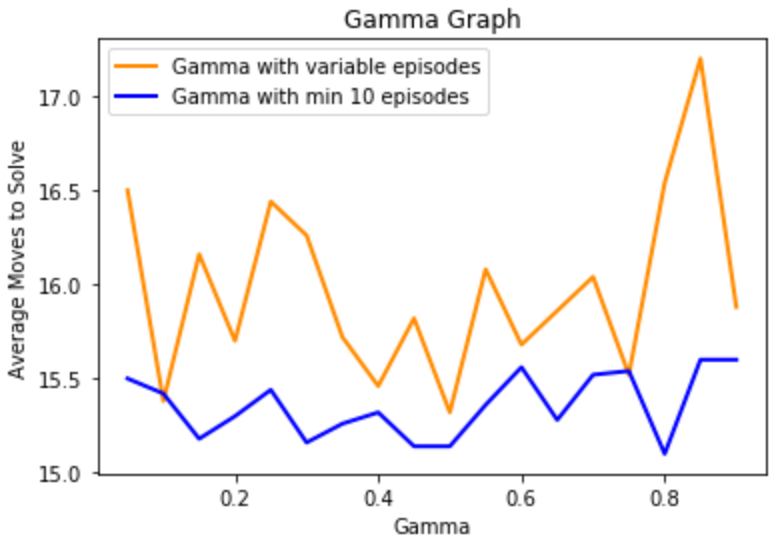

Figure 3

The last experiment is for the epsilon hyperparameter. The values ranged from 0 to 1 with an increase of .05 for each iteration. The number of episodes was between 7.18 to 8.3 for the variable episodes and 10.04 to 10.42 for the minimum of 10 episodes trial.

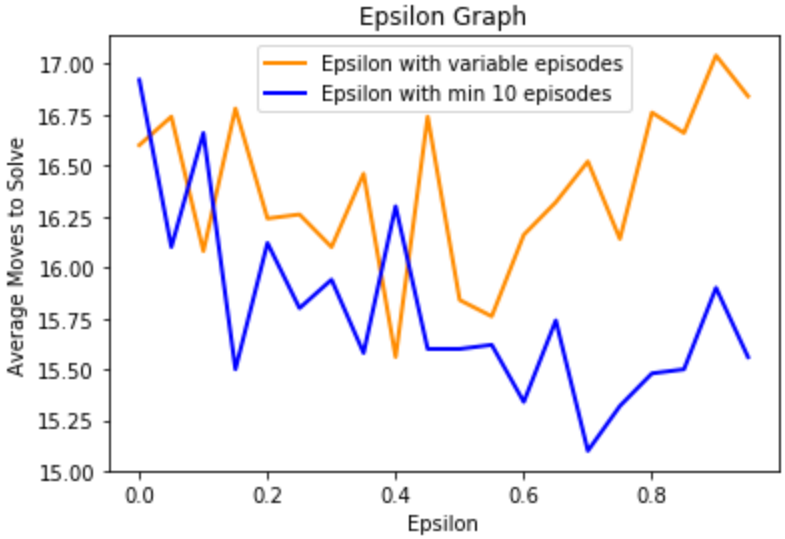

Figure 4

Discussion

The results show that as the number of episodes increases, the Q-table converges to find the minimum number of moves to solve the puzzle. It was interesting to see that when the epsilon value was set to .1 the Q-table did not converge even after 29 episodes. This shows that the algorithm benefits from exploring the state space more than exploiting previous rewards. This would seem to imply that the algorithm is learning ways to solve the problem that are not optimal, and following those reward too often if it is not given enough chances to randomly explore the possible moves.

The results for alpha are interesting. It appears that small and large alphas are more efficient than alphas in the middle. Both alphas that used the variable number of episodes and the minimum of 10 episodes experienced spikes when alpha was between .40 and .60. They also both had low points when alpha was very small and very large. In both cases the difference in the number of moves to solve the puzzle was about one less move for the best values of alpha compared to the worst. This is about a 6% increase in results.

The results for the gamma experiment show that gamma plays less of a role in finding the best solution than alpha does. While gamma with a variable number of episodes has a high peak for a value of .85, this is most likely an outlier as the standard deviation is 0.46 and the mean is 15.97. When gamma is run with a minimum number of episodes, the results are even tighter around a mean of 15.35 with a standard deviation of 0.166. Gamma appears to affect the results the least.

The results for the epsilon test show that as epsilon increases, more likely to make random choices, the algorithm becomes more likely to find the minimum number of moves to solve the puzzle. When the algorithm runs for a minimum of 10 episodes, the pattern is most noticeable. The worst result was at epsilon=0 with the policy taking 16.9 moves on average to solve the puzzle. The best result was when epsilon= .75 and the policy taking 15.1 moves on average to solve the puzzle. This is a 10% improvement in the number of moves to solve the puzzle. The results also confirm the results of the first experiment, the number of episodes with variable and minimum episodes. The algorithm performs best when it is less greedy and explores the state space instead.

Further Research

This paper explores the hyperparameters as a single variables. Further research could be done by exploring the hyperparameters as multivariables. For instance, a three dimensional graph with the number of episodes from 0 to 30 graphed against each hyperparameter could reveal more about the relationship between the hyperparameters and the number of episodes need to train the model. Taken to the extreme, it would be interesting to have a multidimensional graph of all the hyperparameters, however, this would require the algorithm to run 12,000,000 times with the current loops. On the current machine, a chromebook, this would take over 100 days to run!

The algorithm itself could also be changed. Currently, it checks to make sure that the first row of the Q-table has a non-zero value. This insures that the policy is deterministic and always has a path to the solution. Without this constraint, the policy can produce cycles. This can be avoided

if the policy has multiple options for a move and the play function randomly pick amongst the given moves. This would allow more disks to be used without worrying that the Q-table is fully filled.

Another possible change to the algorithm is to randomize the starting state and explore the next state instead of starting from the initial puzzle starting condition of all the disks stacked on the "A" peg. This would improve the first several episodes since the Q-values are mostly set to 0 and the only learning occurring is from the last move to the final state that solves the puzzle. Another possibility would be to start training the model backwards with the first episode looking at the moves that solve the puzzle. This has been proposed and implemented in another paper, specifically for the Towers of Hanoi[5].

Finally, a more robust statistical analysis could be used to better uncover the relationships between the hyperparameters. There appears to be a lot of randomness in the algorithm before it converges and this could be better teased out by statistical analysis. Since the paper only looks at a board with three pegs and four disks at most, this is less of a problem because it only takes a few seconds for the Q-table to coverage. However, with larger state spaces it can take significantly longer and the data would have to deal with Q-tables that are not complete to construct a policy.

As a final note, I learned how to implement Q-learning from scratch. I also learned how to create and make the state space for a puzzle in python. This was one of the hardest parts since getting

[5] Edwards, Ashley D., Downs, Laura, Davidson, James C. Forward-Backward Reinforcement Learning. https://arxiv.org/abs/1803.10227, March 2018

the state space right and storing the rewards and Q-values was one of the hardest parts of the project. I also learned that for the Towers of Hanoi, it appears more important to explore the state space than it is to follow the previous rewards. This makes sense since the state space is limited and the goal is to find the optimal number of moves.

My hypothesis was mostly correct. The epsilon value appears to be the most important and that it should not be set to be greedy, i.e. it should explore more than exploit. The alpha hyperparameter does appear to have some value, but surprisingly it looks like it has a inverted parable shape with high and low values working the best. And lastly, I did confirm that gamma has the least amount of impact on determining the optimal solution.