

CS181 Spring 2016 Practical 4: Reinforcement Learning — Team EXT3

Robert J. Johnson — Dinesh Malav — Matthew McKenna

May 1, 2016

Abstract

Swingy Monkey is a very basic game written in Python. Our project focused on using this game as a simplified context to develop and gain exposure to reinforcement learning algorithms. By implementing a variation of Q-Learning, our study showed that after 200 epochs, our score was able to reach 1,016, a vast improvement over random movements from the monkey.

1 Technical Approach

Initially, we conducted cursory research on different approaches that were available utilizing reinforcement learning. Based on the YouTube video included in the practical specification, we decided our main focus would be an implementation of a particular reinforcement learning algorithm, Q-Learning. We then coded up a solution in Python to develop a policy for the monkey to navigate the course. In basic terms, Q-Learning is an unsupervised learning algorithm that essentially stores value information regarding a particular approach to an outcome-based event in a matrix and uses this matrix to make further decisions in later epochs. We attempted to implement the process described at:

<http://mnemstudio.org/path-finding-q-learning-tutorial.htm>

The actual python file that contains our solution is `qlearn.py`.

Two major problems arose initially that constrained our implementation. The first major issue was how to interpret the incoming stream of objects in the jungle. In Swingy Monkey, we are faced with what is essentially a continuous input of data, and thus, we decided early on that the best way to for both ourselves and the algorithm to interpret the position space was to discretize the playing field into bins. The main parcels of information used were standardized by bin-size, which was variable for each trial. The second issue was the algorithm's ability to infer how the gravity changes during each epoch of the game. Since the pattern is random, we had to quickly find a way to deduce what the rate

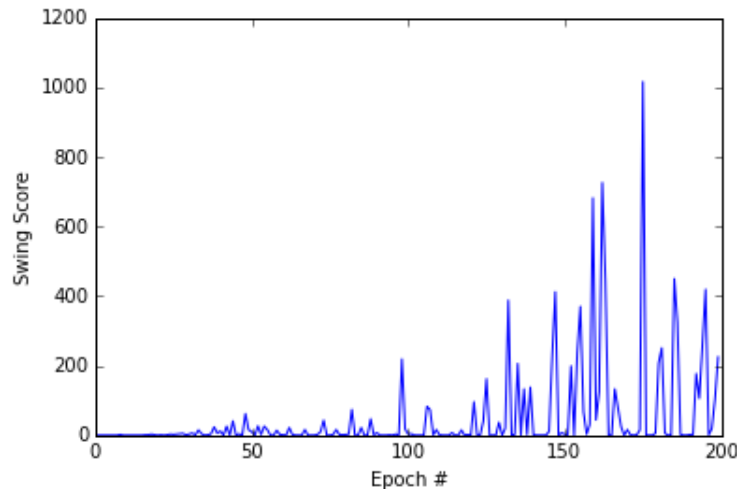
was in order to determine our policy for navigating the trees. Ultimately, we ended up explicitly ignoring the rate of gravity, as over the course of the learn-time, the Q-Matrix will store information seen for various gravitational pulls implicitly and based on the current state be able to make policy adjustments.

Our implementation of the algorithm begins by getting the current state of the monkey and the forest. The ultimate goal for the algorithm is to learn which actions have the greatest "reward" and use this information combined with the current input to maximize utility and make informed decisions regarding the next action to take. Our reward information is stored in matrix R and our information on learned rewards is stored in a similar matrix Q .

The function *action_callback* ended up doing the bulk of the work for the Learner object. In *action_callback* we begin by getting the current velocity and location of the monkey and the height of the next tree. At this point, we use the Q-Function and our values in the Q-Matrix to calculate the current maxima, and adjust the next maneuver based on this information. We can further change our model by using epsilon values to predict our next action.

2 Results

Our monkey was trained to reach an average score of 48.7 after 200 epochs, with a maximum score of 1,016. As expected, the average score increased dramatically as the number of epochs increased and the algorithm learned how to control the monkey based on the Q matrix. Around epoch 175, we saw a score above a thousand, and a few other scores above 600 after about 150 epochs:



We also tried changing the values of the hyper-parameters to see how they would affect the swing score. After trying various values of epsilon and bin size,

we found that a bin size of 50 and an epsilon value of .001 gave the highest average swing score:

Epsilon	Bin size	Avg. Swing Score
.01	10	0.63
.01	50	7.1
.001	50	48.7
.001	100	4.0
.0001	50	25.5

The bin size had a non-trivial effect on the swing score. Changing the bin size even slightly would produce results that were significantly worse compared to using a bin size of 50. When using a bin size of 10, the maximum score we saw was 9, compared to 1,016 with a bin size of 50.

3 Discussion

The implementation discussed here is not perfect, and there are other reinforcement learning approaches that we could have explored. Research in reinforcement learning is rapidly changing and evolving, so there will likely newer and more novel solutions developed for similar discrete problems such as this. Q-Learning was selected due to its ease of implementation and potential to effectively implement a policy for the monkey's traversal.

A natural first step for further research and experimentation would be to study how we can optimize the hyper-parameters of the model. There must exist an ideal epsilon/bin-size combination, but for the scope of the assignment we were not able to determine this value. We also only included height information for the top of the tree. It is our suspicion that if we could somehow learn information regarding the height of the bottom tree portion then the algorithm could be improved. Again, this was not able to be included during the time afforded to us on this particular practical. Lastly, given more time, we would have liked to have seen why certain earlier epochs performed better than later epochs. There may be some course/gravity pairings that give our model trouble, and de-bugging those could vastly improve our performance.

All code for this project can be found at:

<https://github.com/HarvardCS181Practical2016>

.