CS181 Reinforcement Learning Practical Report

Group Members: Danyang Chen, Yingzhuo (Diana) Zhang, Ruitao (Toby) Du

1. Swingy Monkey Exploration

The first step was to explore the game and see what parameters are available in the game as well as how to retrieve these parameters. As we first tried to play the game by hand, we noticed some available parameters - the position of the tree trunk, the speed of the monkey, and the position of the monkey. We observed that when the monkey jumps, the velocity of the monkey follows a Poisson distribution. Poisson distributed velocity complicates the situation a bit because we don't know how high the monkey will jump when we let it take the action of jumping, so the monkey might need to jump, but it could jump too high or too low. We also need to know the distance of the monkey from the tree trunk in order to determine whether or not to jump. In order for us to get a general picture of the game, we ran the given code with random jumps on Swingy Monkey for many times, and recorded the existing positions of tree trunk, positions and speed of the monkey, from which we can obtain the distance between the monkey and the tree trunk. Tree trunk position and monkey position both consist of two numbers, the top and the bottom, however, all we need from these parameters is whether or not the monkey survived. Therefore, we reduced the number of parameters by merely keeping the difference between the bottom of the tree trunk and the bottom of the monkey. By doing so, we acquired a starting range for the three parameters (Notice that all parameters are integers, so they are discrete in the range.):

1. Difference between bottom position of tree trunk and bottom position of monkey: (-70, 400)
2. Distance between tree trunk and monkey: (-465, 485)
3. Velocity of monkey: (-42, 42)

We continued to update these ranges as we train the monkey to perform better and explore more possible scenarios.

The reward system of the game is quite simple, the monkey stays alive and get 1 point if the it passes a tree trunk successfully and the monkey dies when if fails to pass the tree trunk. There are three possible scenarios for the monkey to fail:

1. The monkey can hit the tree
2. The monkey can fall off the bottom of the screen
3. The monkey can jump out the top of the screen when it jumps too much.

2. Reinforcement Learning

Now that we have a general idea of the game, we can redefine the game as a reinforcement learning task. We know that for reinforcement learning, the inputs are states, actions, and reward function. For the Swingy Monkey game, we have a set of three

states, which are the three parameters we described in the previous section. There are two available actions, jump or no action. The reward function should be designed to consider the possible outcomes of the monkey, that is, the agent (which is the monkey) should get rewarded when it successfully passes through a tree trunk, and it should be punished for the three possible failures described above. We designed the reward function as follows:

- Reward of +1 for passing a tree trunk.
- Reward of -5 for hitting a tree trunk.
- Reward of -10 for falling off the bottom of the screen.
- Reward of -10 for jumping out the top of the screen.
- Reward of 0 otherwise.

2.1 Discretization/Dimensionality reduction

Since the possible values for the states are all integers, we can calculate the total number of possible states using the initial range for the states. The total number of possible states is product of the number of each state. However, even with the initial range, this total number is extremely large, so we must first try to reduce the total number of possible states. The intuition here is that we can group the states into a small subset. For example, the initial range for the monkey's velocity is (-42,42). Say we want to group the velocities into three subsets, fast, medium, and slow. Given an arbitrary velocity v, the groups can be defined by

$$K = (v - (-42)) / n,$$

where n is determined by how many groups we need. In this particular example we have three groups, so n = 28. Notice that we are doing integer division here, so we will have 3 possible integer values for K that represents the groups. Grouping the distance between the monkey and tree trunk and the difference of the bottom position of tree trunk and monkey are done in a similar manner. We tried a few different settings of Ks, and they are summarized in the table below.

	Setting 1	Setting 2	Setting 3	Setting 4
Bottom Position Difference	5	10	5	10
Monkey-Tree Trunk Distance	3	3	6	6
Velocity of Monkey	5	10	5	10

Table 1. Different settings for grouping the stats

2.2 Q-learning

We choose to implement Q-learning on this problem. Q-learning is one method of model-free reinforcement learning, so instead of transition functions, we have the update rule for the Q value, which is defined as:

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{\alpha'} Q(s',a') - Q(s,a)] ,$$

where α is the learning rate, γ is the reward for taking action a in state s. As for any reinforcement learning task, we are concerned with exploration vs exploitation, hence we incorporated ε-greedy into Q-learning. Recall that with ε-greedy, when deciding the next action to take, the agent goes to the best action with probability 1- ε to exploit, and with probability ε the agent takes a random action to explore.

Now we need to tune the parameters α, γ, and ε to improve performance of Q-learning. We used the four different settings for grouping the states described in Table 1, and we tuned the three parameters separately for each setting and compared the performance for each setting. The algorithm for tuning the parameters is as follows:

```
For each Setting:

      Initialize the parameters as α = 0.001, γ = 0.4, ε = 0.1.

      Run Q-learning by keeping γ and ε constant and altering the value
      of α.

      for each value of α:
            for i in number_of_trials(=5) (i.e. Run Q-
            learning 5 times):
                  Record the cumulative average score of the
                  game achieved by Q-learning.
            Take the average of the 5 cumulative average
            scores.
      Choose the α that gives the highest average score.

      Use the best α from the last step, and alter the
      value of γ.
      for each value of γ:
            for i in number_of_trials(=5) (i.e.Run Q-learning
            5 times)
                  Record the cumulative average score of the
                  game achieved by Q-learning.
            Take the average of the 5 cumulative average
            scores.
      choose the γ that gives the highest average score.

      Use the best α, best γ from the previous steps and alter the
      value of ε.
      for each value of ε:
            for i in number_of_trials(=5) (i.e.Run Q-learning
            5 times)
                  Record the cumulative average score of the
                  game achieved by Q-learning.
            Take the average of the 5 cumulative average
            scores.
      choose the ε that gives the highest average score.
```

The lists of α, γ, and ε we tried are as follows,

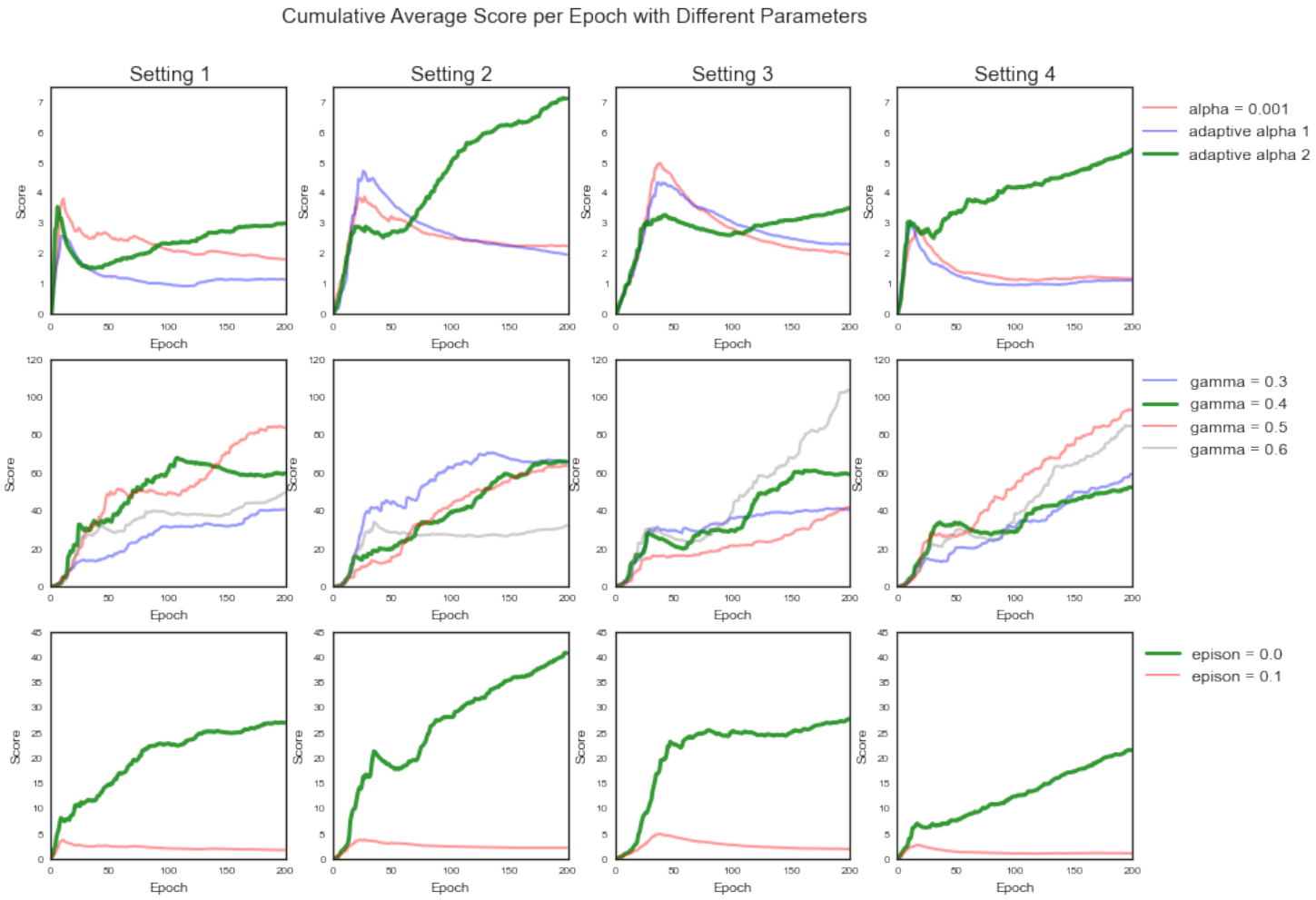
$\alpha = \{0.001, \min(0.001, 1/\min(\text{number of iteration}^{1})), 1/\text{number of visit}^{2}\}$

1. number of iterations here refers the iterations of Q-learning, i.e. the number of times Q value was updated. (This α is denoted as adaptive α 1 in later plots)
2. There is a number of visit for each (state, action) pair, and the visit is the number of times this (state, action) pair is visited. (This α is denoted as adaptive α 2 in later plots)
3. Both 1 and 2 are adaptive approaches for α.

$\gamma = \{0.3, 0.4, 0.5, 0.6\}$

$\varepsilon = \{0, 0.1\}$

Below are the plots that demonstrate the tuning process. The plots in each column represent tuning for each setting, and the rows show the results for choosing α, γ, and ε respectively.



3. Summary and Conclusion

In terms of parameters α, γ, and ε, we see that adaptive alpha 2 which is a function of number of visits performs the best for all settings, $\varepsilon = 0.0$ performs the best for all settings. That is, we shouldn't let the agent take any random action; all actions should be decided by the Q value. The reason is that for this particular game, the monkey dies whenever it fails to pass a tree trunk, meaning that one action can lead to death of the monkey, so taking random actions will increase the probability of the monkey dying therefore reducing average score. The best value for γ varies from setting to setting, because for each setting the number of (state, action) pair is different, so the reward parameter γ differs. However, we notice that $\gamma = 0.4$ is rather stable across all settings, so we used this value in final algorithm.

In terms of settings, setting 2 has the best overall performance because it gives the highest average score.

After selecting the best parameters, we used the set of the best parameters to train the monkey for 200 epochs and we saved the Q value, below is the result of the Swingy Monkey when we run the game with the saved Q value. We also made a recording of the money that is available in the this link:

https://www.youtube.com/watch?v=claunbjlv40&feature=youtu.be