Practical 4 Report

Andong Liu, Yulei Wang and Xiaoran Zhang

April 29, 2016

1 Game Description

In the Swingy Monkey game, the user control a monkey that is trying to swing on vines and avoid tree trunks, where the gravity and the distances between trees vary from game to game. In this practical, we developed an agent that plays this game on its own through Reinforcement Learning. The reward for the agent is given 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 off the top of the screen;
reward of 0 otherwise.

2 Model Selection

2.1 Markov process

We began with considering the possibility of taking Markov process, a stochastic model that has the Markov property, as our potential model. It is usually used to model a random system that changes states according to a transition rule which only depends on the current state. However, after taking a closer look at the features of this task, we rejected to use Markov model or the model-based approach for the following reasons. First, our objective is to maximize the expected rewards R and find the corresponding optimal policy π^* that

$$\pi^* = arg\max_{\pi} E\Big(\sum_{s\in\mathbf{p}} R(s,\pi(s))\Big|\mathbf{p}\Big) \tag{1}$$

where $\mathbf{p}$ is a possible path of $\{S_t\}$.

The information available to us including S and A space, while unknown information including both the transition probability $P(s_{t+1}|s_1,\dots,s_t,a_1,\dots,a_t)$ and the reward function $R: S\times A\to R$. Specifically, each state S is a continuous vector, with $|S|=\infty$, and each action A is one of the two possible actions, meaning that $|A|=2$. Under Markov process and model-based approach, we will have to assume that the transition probability is independent from all the previous states and actions, i.e. $P(s_{t+1}|\{s_i\}_{i=0}^t,\{a_i\}_{i=1}^t) = P(s_{t+1}|s_t,a_t)$. Second, since the efficiency of Markov model heavily relies on an accurate estimation of the transition matrix $\mathbf{P}$ (which is a $|S|\times|S|\times 2$ matrix for this task), we cannot avoid dealing with an overly large space of $S\times A$ when estimating $\mathbf{P}$. We expect the algorithm to update each element of the matrix for a sufficient number of times before receiving a reliable $P(s'|s,a)$ estimate. It is obviously computationally difficult for practice. Therefore, in order to not making any incorrect and over-conservative functional assumptions as well as rising complexity of computing and storaging, we decided not to use Markov model.

2.2 Q-learning

Instead, the available and unavailable information in this task well matches the setting of reinforcement learning since we only be able to observe the action and corresponding reward at each state. Thus, the first model we built is Q-learning.

However, there are many challenges beyond selecting a model. First, we need to define a state space for the Q function. The monkey is swinging in a forest (background picture with pixels 400 x 600) continuously. The state will depend on our observation of where the monkey is. The second challenge is that the game has random gravity, so the state needs to depend on gravity at each epoch.

We have the freedom of choosing the observations we want, we first narrow our observation to dh = the horizontal distance from the monkey to the next tree, dv = the distance between his head and the next tree gap's top boundary, and v = how fast the monkey is moving (this accounts for gravity). To simplify the model, we chose to define a discrete state space by dividing the possible values of the observations into bins, that is, each state is indexed to be s_t{dh_bin_number, dv_bin_number, v_bin_number}. We observe the history of the monkey as a sequence of observations and rewards $\{\ s_1,\ action_1,\ reward_1,\ s_2,\ action_2,\ reward_2,\ \cdots\}$.

With the Q-learning approach, the complexity is greatly reduced by only considering updating the Q matrix, without extra assumptions of probabilistic model or the transition probabilities. Below we start with stating the Bellman's equations and its expanding version which we used in the analysis.

$$\begin{aligned} Q(s,a) &= R(s,a) + \gamma \sum P(s'|s,a) max_{a' \in A} Q(s',a') \\ &= R(s,a) + \gamma E_{s'}[max_{a' \in A} Q(s',a')] \\ &= E_{s'}[R(s,a) + \gamma max_{a' \in A} Q(s',a')] \end{aligned}$$

Through Stochastic Gradient Decent, we estimate the target Q by learning the difference between current Q and the Q we previously expected, where $\hat{Q}_{\text{target}} = r^{obs}_{s,a} + \gamma max_{a' \in A} Q^{old}(s',a')$. Thus the updating rule for Q is as the following,

$$Q(s,a)^{new} \leftarrow Q(s,a)^{old} + \alpha[r + \gamma max'_a Q(s',a')^{old} - Q(s,a)^{old}] \tag{2}$$

where $\gamma \in (0,1)$ is the discount factor, and $\alpha \in (0,1)$ is the learning rate.

Specifically, we used ϵ-greedy algorithm for choosing the new action, that is, at each time t, we choose a random action as our next move with probability ϵ, and we choose an action that has the highest Q value from the Q table with probability (1-ϵ). Tuning the value of ϵ changes the exploitation and exploration ratio. We also changed the learning rate α to 0.1* original α if the number of iterations exceed 100.

After discretizing S as mentioned earlier, we also decide what value of the learning rate (α), the discount rate (γ) and the ϵ-greedy factor to use, since they determine the convergence speed of the program. These parameters are involved in the learning process mainly in two ways. For one, the program will visit every state for an unlimited number of times, and the learning rate will become smaller as epoch increases, and eventually leads the Q values to converge. For the other, at the point of convergence, the policy that brings us the Q values will the optimal polity selected by learning. In this analysis, we choose some large values of γ, i.e. from 0.6 to 0.9, for it's reasonable to preserve the impact from previous states and actions relatively longer for the learning afterwards. In the next section, we present the scores with different discount rates at given learning rate and ϵ-greedy factor, then conclude the impact from tuning these parameters. Similarly, we also compare the learning performance with different α and ϵ inputs given the other parameters fixed.

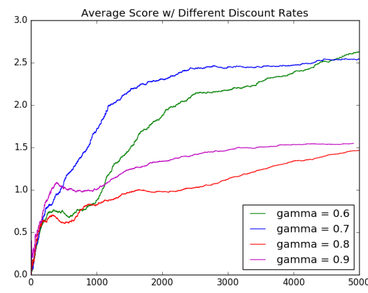
2.3 Q-learning 2.0

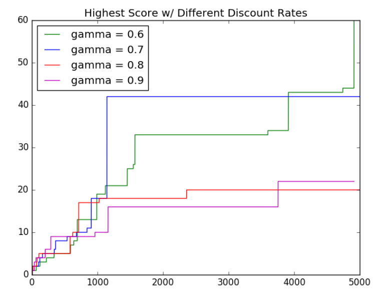
After performing several versions of this Q-learning model with different learning rate, discount rate and ϵ values, we built a more complex model which includes a bigger state space and more customized learning rate value.

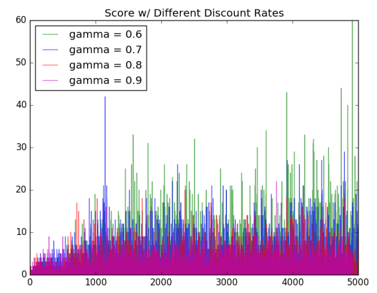
We added another dimension to our state observations. The new states are defined by bins in dh = the horizontal distance from the monkey to the next tree, $dhead$ = the distance between the monkey's head and the next tree gap's top boundary, v = how fast the monkey is moving (this accounts for gravity) and $tbot$ = the distance between the bottom of the tree gap to the ground. The new dimension $tbot$ is added because we notice that our old states only tell us where the monkey is to the tree, but do not tell us how low the monkey is relative to the ground. The α values in this new model depends on how many times the corresponding $Q(s, a)$ has been filled, i.e. the number of times the monkey finds itself taking action a in state s $k(s, a)$. We set the learning rate for each state to $k(s, a) = 1/k(s, a)$. In this way, the higher the number of times the monkey visits $Q(s, a)$ the lower α will be, and the learning rate will reach the point with the value of k increases leading to convergence of Q.

3 Performance Evaluation

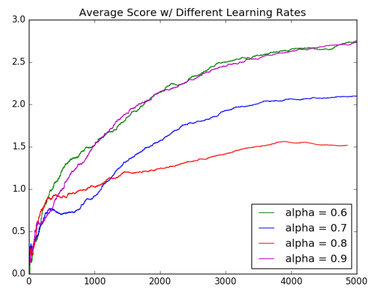
3.1 Q-learning

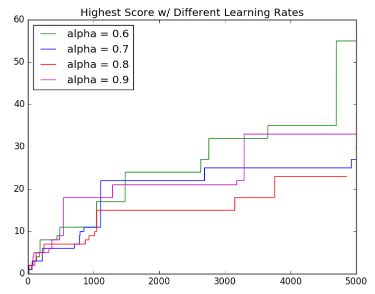


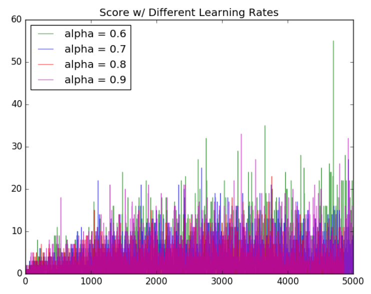




Above three plots show the average, highest, and raw scores from the Q-learning process with different discount (γ) rates input, given the learning rate (α) maintains the same. Combining results from all three plots, we may conclude that, with lower values of γ, i.e. 0.6 and 0.7 vs. 0.8 and 0.9, the learning performance is consistently better than it with higher γ. Specifically, both the highest scores and average scores are nearly double those from learning with the higher rates as the number of epoch increases. This indicates that the states and action in the future has moderate relationship with the previous ones.



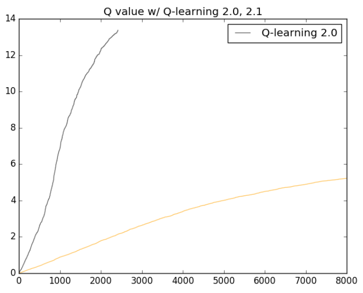




Above three plots show the average, highest, and raw scores from the learning process with different learning rates (α) input, given the other parameters maintain the same. Combining results from all three plots, we may conclude that, with lowest and highest values of α, i.e. 0.6 and 0.9 vs. 0.7 and 0.8, the learning performance is consistently better than it with α's that are not so extreme. This may reveal some nonlinear relationship between the learning rate and scores, suggesting us to try some nonlinear models of α in Q-learning 2.0.

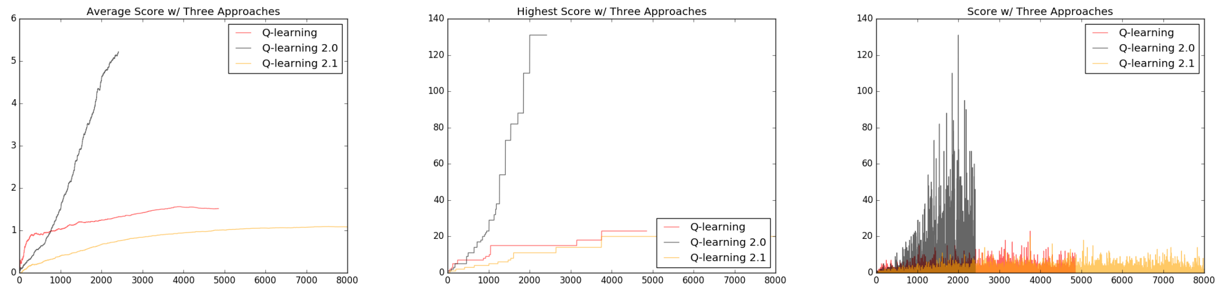
3.2 Q-learning 2.0

Q-learning 2.0 has a noticeably slower learning speed compared to the simplified Q-learning model. This is likely due to the fact that an additional dimension is included to the state space. The number of dimensions in



Q-learning 2.0 is scaled by the number of bins in this extra dimension, meaning it will take more iterations to fill out the Q matrix. Due to the time constraint of this task, we were not able to test it with a larger number of iterations such as 10,000, however, we expect the Q-matrix to be filled more completely, which will result in a better-performing model.

To test the performance of the model with only α updated according to $k(s, a)$ with state space remain the same, we have the following plot. The two models have the same initial α and γ values. Notice that this model is much better than the previous one. The average score starts off increasing slower than the Q-learning model, but it still increases at a reasonable speed when the Q-learning's average speed goes into a steady state. The plot of the highest score and the current scores are both much better than the previous model.



Next, we added the *tbot* dimension to the model. Notice the proportion of filled Q matrix values are very low due to the increased number of dimensions.

4 Discussion

To improve our learning process, one possible approach is to redefine the state space. Our current state space has discrete grids, in the future we may instead use some function approximation, such as neural network, to represent the value function or the Q-function.

Second, we desire to increase the number of iterations. This will ensure that more Q-matrix values to be filled. The more complete the matrix is, the better the model will perform.

Finally, we can also implement a function that uses grid-search to find the best hyper-parameter values.

5 References

Markov Process. $https://en.wikipedia.org/wiki/Markov\_process$

6 Github

https://github.com/aannieliu/monkeymonkeygogogo