Practical 4: Reinforcement Learning

Bolei Deng, Matheus C. Fernandes, and Nathan Wolfe
boleideng@g.harvard.edu, fernandes@g.harvard.edu, nwolfe@college.harvard.edu
Canvas Group Name: P4 Peer Group 10

April 28, 2017

1 Technical Approach

The approach we took for this practical was to implement a Q-learning, off-policy, Reinforcement Learning (RL) algorithm that takes in the state of the game's 4 features and predicts the best move to achieve a high score. For the linear Q-learning update we used the following equation and also used an ϵ-greedy algorithm for exploration-exploitation balance. More specifically, we predefine a constant probability ϵ. With probability $1 - \epsilon$, we choose the action $a' = \text{argmax}_{a' \in A} Q(s', a'; \boldsymbol{w})$; otherwise we choose a' randomly to explore the world.

$$w_{s,a} \leftarrow w_{s,a} - \eta \left(w_{s,a} - [r + \gamma \max_{a' \in A} Q(s', a'; \boldsymbol{w})] \right)$$

We developed the algorithm from scratch and did not end up using any off-the-shelf tools such as SciKit Learn. To play the game we took the following states into consideration: 1. monkey vertical velocity; 2. monkey's horizontal distance from the tree; 3. monkey's vertical height difference from the tree's bottom stump; 4. monkey's gravitational acceleration. We attempted multiple approaches to tuning and configuring the above equation's parameters to ensure that the learning happens at an optimal rate and the player is able to score the highest requiring the least amount of epochs.

In order to obtain the gravity for our state space, we have included a function that evaluates the difference between velocities in each time step. Given that we know that the gravity will vary between 1 and 4, we make sure that values falls in either one of these binary options. To ensure that we are obtaining the correct reading for the gravity and not the impulse of an action, take the reading for the gravity at the beginning of the simulation and enforce a no-action policy for the first time step as well as adding a condition making sure the value difference between velocities is negative; whereas, if it is positive we know that the acceleration reading is due to the an action impulse. The gravity reading is only set to reset during every new sequence (every new epoch).

1.1 Feature engineering

Additionally, to make the code more efficient for learning, we discretized the pixel-space into agglomerates of 50 pixels, as well as discretizing the velocity into 6 segments (including 0) with a maximum value of 40. Anything velocity above 40, is considered to be equivalent to 40. Also we

limited the distance between the monkey and tree to 0, such that once the tree passes the monkey, it is considered done – this allows the monkey to behave the same way once a tree has passed and another tree has not yet appeared. We implemented the height difference of the monkey and the lower tree stump to be discretized from the screen space and also half of the screen space below the 0 difference mark, meaning that it accounts for negative distances. This was a fundamental improvement to assure that we were not accessing negative indexes, and ultimately not affecting the learning algorithm that is happening along the top of the screen.

To ensure that we are taking accurate readings for every time step, we save all of the actions and last state into a last state and last action variable that is globally saved across every iteration. For the first time step of the sequence (epoch), we check to see if there is an existing last action, and if not, then we just do not do any actions such that we can obtain the accurate gravity readings. Otherwise, if the code has iterated into the next time step after the very first step, we begin the learning and action prediction algorithm.

1.2 Revised epsilon-greedy method

When we use ϵ-greedy method, we are trying to add uncertainty into actions. While if we have already taken this particular actions for many times, we would expect this action to be less uncertain. As for the learning rate η, we would like it to be large at first, so that it would converge earlier. While upon getting closer to the optimal value, we would like η to be smaller to prevent overshooting. To dynamically modify ϵ and η, we improved our algorithm by tracking the history of actions. We count the number of times action a is taken at state s as $C(s, a)$, where C is a counting matrix with the same shape as Q. Initially, we predefine only ϵ_0 and η_0, and then define ϵ as

$$\epsilon \leftarrow \frac{\epsilon_0}{\max\{C(s', a'), 1\}}$$

define learning rate η as

$$\eta \leftarrow \frac{\eta_0}{C(s, a)}.$$

2 Results

In this part, the results for Q-learning is presented. Two methods are used, where method 1 refers tothe ϵ-greedy method with constant ϵ and η and method 2 refers to that with dynamic ones (as mentioned earlier). Three hyper-parameters are considered: ϵ_0, γ and η_0. We analyzed the influence of these parameters for both methods via perturbing terms from a base state, (ϵ_0, γ, η_0) = (0.01,0.9,0.2). More specifically, nine different parameter sets are studied as illustrated below.

No.	ϵ_0	γ	η_0	method
1	0.001	0.90	0.20	1&2
2	0.005	0.90	0.20	1&2
3	0.01	0.90	0.20	1&2
4	0.05	0.90	0.20	1&2
5	01	0.90	0.20	1&2
6	0.01	0.80	0.20	1&2
7	0.01	0.95	0.20	1&2
8	0.01	0.90	0.10	1&2
9	0.01	0.90	0.50	1&2

We conducted 18 simulations with 300 epochs each. For each simulation the score for every trial is recorded. We plot, in Fig.1, the 50-epoch rolling average score versus the epoch number conducted.

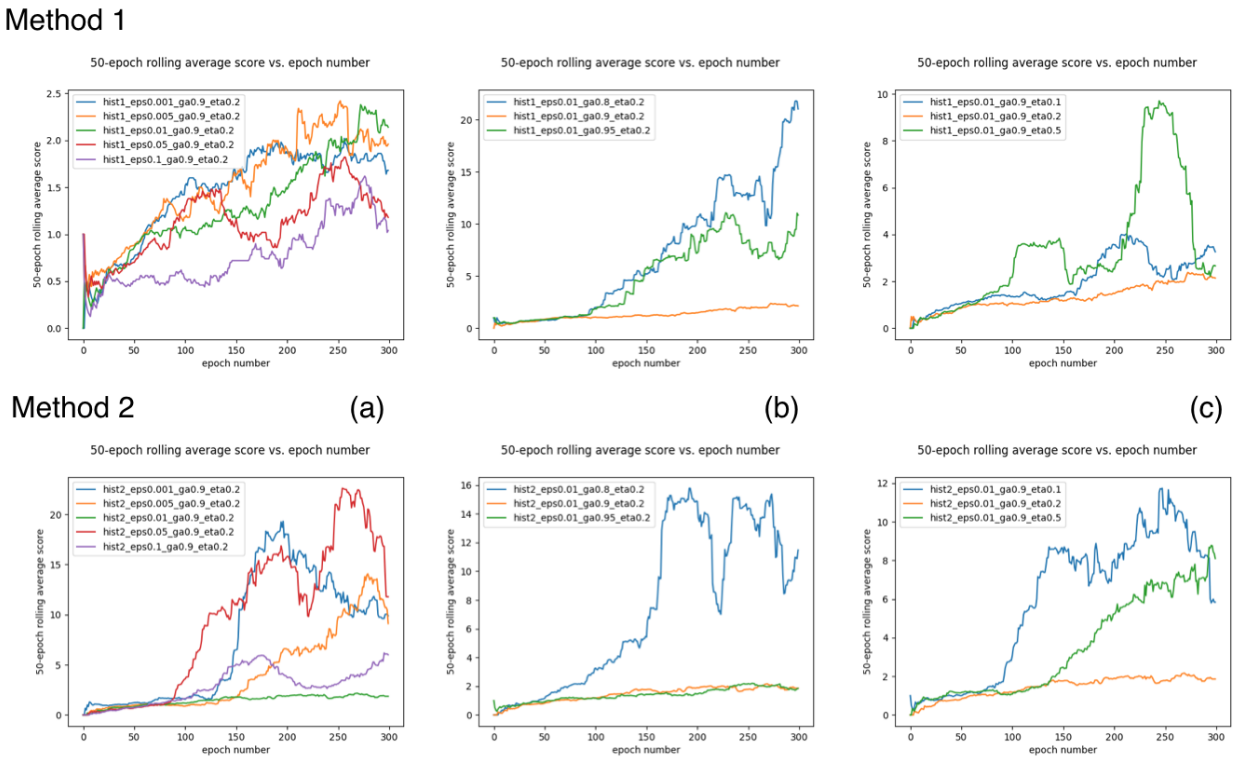

Figure 1: 50-epoch rolling scores versus epoch number for both methods with different values of (a) ϵ_0, (b) γ and (c) η_0.

3 Discussion

Overall method 2 gives a better performance comparing to method 1. It seems that the learning performs better with smaller γ. For method 1, when the learning rate η_0 is large, the performance fluctuates a lot (shown by green line of Fig. 1(c) for method 1) due to the overshooting problem.

Meanwhile, in method 2, there is no such issue as we keep modifying the learning rate to prevent that. We are not sure the relation between performance and parameter ϵ_0, it seems that there is some randomness here, especially for larger ϵ_0. That make sense because by trying larger ϵ_0 we are actually randomly exploring the world. Sometimes, if we are lucky, we explore to the right direction and get nice performances, and sometimes not.

Although the average performance of Q-learning is generally around 20 to 30 points, the single highest score is always surprisingly high. In another words, the potential of getting a great score is very significant, particularly when we start in the state where we have learned a lot.

Figure 2: Game snapshot.

Github Repository

`https://github.com/matheuscfernandes/cs181-s17-practicals/tree/master/p4`