# Practical 4: Reinforcement Learning

Group 11: Rohan Thavarjah / Jonne Saleva / Anthony Soroka
rthavarajah@g. / jonnesaleva@college. / atsoroka@g.

April 28, 2017

## 1 Technical Approach

The objective is to build an agent that learns to play *Swingy Monkey* - a game similar to the 2013 hit *Flappy Bird*. The agent requires a framework to learn and act in its world. We naturally use a Markov Decision Process (MDP) as the framework, with the MDP consisting of:

- A set of states $S$, and a set of actions $A$
- A reward function $r : S \times A \rightarrow \mathbb{R}$
- A transition model $p(s'|s,a), \forall s, s \in S, a \in A$

From this framework the agent learns a policy $\pi$, mapping states to actions, i.e. $\pi : S \rightarrow A$.

If we knew the reward function and transition model (i.e. understood the dynamics of the world), we could consider using value or policy iteration to determine an optimal policy (though convergence would take a significant amount of time given the size of the state space). Similarly, model-based RL is not an option given that the transition model for the world would be too large.

Hence we use model-free reinforcement learning. We specifically consider both the on-policy SARSA (State-Action-Reward-State-Action) approach and off-policy Q-Learning approach, with update functions as per below:

$$\text{SARSA: } w_{s,a} \leftarrow (1-\eta)w_{s,a} + \eta[r + \gamma Q(s', \pi(s'); \boldsymbol{w})]$$

$$\text{Q-Learning: } w_{s,a} \leftarrow (1-\eta)w_{s,a} + \eta[r + \gamma \max_{a' \in A} Q(s', \pi(s'); \boldsymbol{w})]$$

While the above explains how and why we utilize the MDP framework, we still need to determine how to represent the world. The states space, after all, is effectively continuous. To reduce this complexity, we discretize the positional state representation. We break the 600 x 400 pixel screen into square bins using the parameter *PIX_BINSIZE*. We limit the positional components of the state space to a horizontal and vertical component. The horizontal positional component records the horizontal bin distance of the monkey to the upcoming tree (see Figure 1):

```
state['tree']['dist'] // PIX_BINSIZE
```

The vertical positional component captures the vertical bin distance between the monkey's feet the bottom tree stub:

```
(state['monkey']['bot'] - state['tree']['bot'])// PIX_BINSIZE
```

We acknowledge this ignores 1) how close the monkey is from the top or bottom of the screen regardless of the tree's position 2) the location of the top tree stub. We hypothesize the ideal policy has the monkey hovering at some height above the bottom trunk. Additionally we believe there would be marginal benefit of expanding the vertical state representation given 1) our vertical position estimate does offer some proxy for the other vertical positional information 2) a larger state space requires more exploration (i.e. train time).

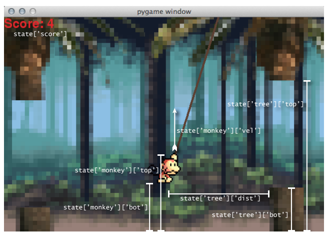


Figure 1: Positional Components of State Representation

Next we incorporate the dynamics of the game into the state space. The monkey's vertical velocity can generally range in integer values between -50 and 50. In aims of limiting the number of states so we can learn more a generalizable policy, we discretize the velocity states into 5 bins:

| **Velocity Bin** | -2 | -1 | 0 | 1 | 2 |
|---|---|---|---|---|---|
| **Velocity Range** | $\leq$ -15 | (-15,0) | [0,15) | [15,30) | $\geq$ 30 |

Table 1: Discretizing Velocity

Unlike in the original MDP *Model 1*, *Model 2* incorporates gravity in the state representation. Recognizing that the the gravity parameter randomly rotates between two values (either 1 or 4), *Model 2*'s agent learns gravity by taking no action (not jumping) on the first time step and interpreting its velocity after the first time step to deduce the gravity parameter.

With the set of states now defined, the MDP next requires a set of Actions $A$ and reward function $r$. These are essentially provided in the game as:

$$A = \{0 : \text{Do Nothing}, 1 : \text{Jump}\}$$

$$r(s,a) = \{\text{Pass Tree} \rightarrow +1, \text{Hit Tree} \rightarrow -5, \text{Hit Top/Bottom} \rightarrow -10, \text{Otherwise} \rightarrow 0$$

Lastly, we set the remaining reinforcement learning parameters: the discount rate, the learning rate, and the exploration rate. $\gamma$ is the discount rate which we originally set to .9. $\eta$ is the learning rate, weighting how much to adjust previous learned Q-values due to recent experiences. When the agent hasn't experienced state $s$ and action $a$ many times before, we want to amend the $Q(s,a)$ significantly, and less so as the agent gains experience. Hence, we set $\eta$ to:

$$\eta = \frac{1}{K(s,a)}, \text{where } K(s,a) = \text{Counts of experiencing } s,a$$

Q-Learning uses $\epsilon$-greedy policy to balance exploitation (choosing believed best course of action) with exploration (trying new actions out). Similar to $\eta$, we reduce the exploration rate $\epsilon$ based off how many times the agent has encountered the greedy state action pair. Specifically we use:

$$\epsilon_{util} = \frac{\epsilon_{param}}{K(s,a)}, \text{where } K(s,a) = \text{Counts of experiencing greedy } s,a$$

$\epsilon_{\text{param}}$ is our parametrization (originally set to .01), and $\epsilon_{\text{util}}$ the actual probability of exploration.

## 2 Results

In order to find optimal hyperparameters, we conducted several experiments. Specifically, we varied the gamma values, the size of the bin in pixels, as well as the learning method (Q-learning vs. SARSA) and gravity (on/off). We limited ourselves to these parameters with computational concerns in mind, since we were optimizing with plain grid search. We noticed several patterns that hold throughout regardless of parameter settings. For example, most runs seem to end in low scores, but the maximum scoring ability of Swingy Monkey seems to increase. This creates a "funnel" pattern in the score traceplots, visualized in Figure 2. Swingy Monkey also seems to reach scores greater than 100 in everything except the SARSA - No gravity condition, which speaks to the strength of the methodology used.

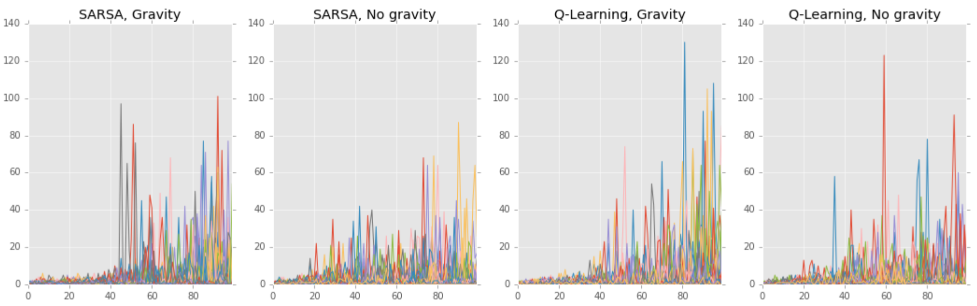
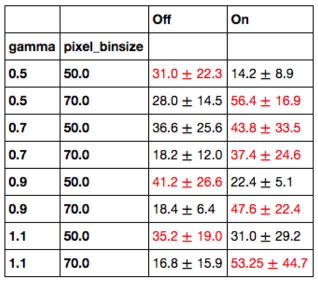
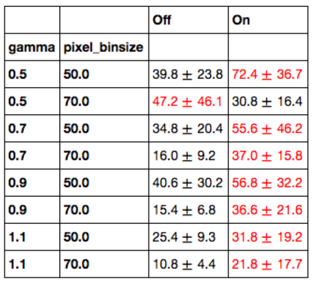


Figure 2: Score of Swingy Monkey vs. Iteration. The different plots correspond to different learning type / gravity settings.

Overall, there is a lot of noise in the results for each hyperparameter setting, as can be seen in the Figures 3 and 4 below. It is clear, however, that with gravity activated, Swingy Monkey seems to do better on average, at least in terms of the maximum score attained in 5 epochs of 100 runs each.

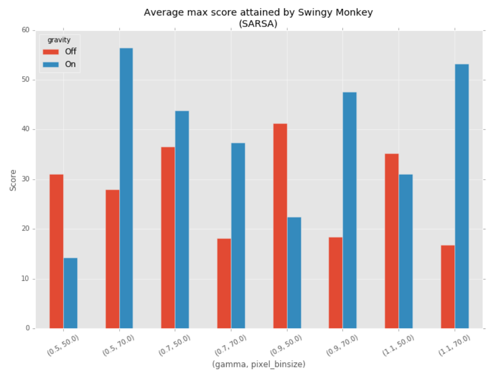


| | | Off | On |
|---|---|---|---|
| gamma | pixel_binsize | | |
| 0.5 | 50.0 | 31.0 ± 22.3 | 14.2 ± 8.9 |
| 0.5 | 70.0 | 28.0 ± 14.5 | 56.4 ± 16.9 |
| 0.7 | 50.0 | 36.6 ± 25.6 | 43.8 ± 33.5 |
| 0.7 | 70.0 | 18.2 ± 12.0 | 37.4 ± 24.6 |
| 0.9 | 50.0 | 41.2 ± 26.6 | 22.4 ± 5.1 |
| 0.9 | 70.0 | 18.4 ± 6.4 | 47.6 ± 22.4 |
| 1.1 | 50.0 | 35.2 ± 19.0 | 31.0 ± 29.2 |
| 1.1 | 70.0 | 16.8 ± 15.9 | 53.25 ± 44.7 |



| | | Off | On |
|---|---|---|---|
| gamma | pixel_binsize | | |
| 0.5 | 50.0 | 39.8 ± 23.8 | 72.4 ± 36.7 |
| 0.5 | 70.0 | 47.2 ± 46.1 | 30.8 ± 16.4 |
| 0.7 | 50.0 | 34.8 ± 20.4 | 55.6 ± 46.2 |
| 0.7 | 70.0 | 16.0 ± 9.2 | 37.0 ± 15.8 |
| 0.9 | 50.0 | 40.6 ± 30.2 | 56.8 ± 32.2 |
| 0.9 | 70.0 | 15.4 ± 6.8 | 36.6 ± 21.6 |
| 1.1 | 50.0 | 25.4 ± 9.3 | 31.8 ± 19.2 |
| 1.1 | 70.0 | 10.8 ± 4.4 | 21.8 ± 17.7 |

Figure 3: Average max score ± standard deviation. Left: SARSA, Right: Q-Learning



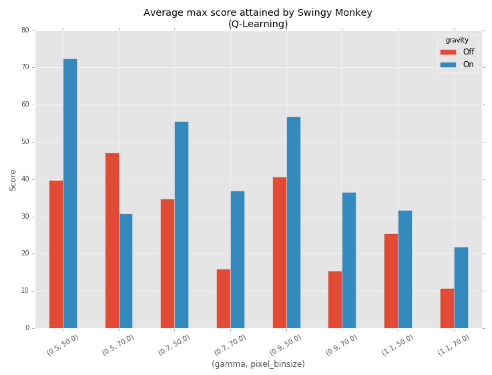


Figure 4: Bar plot of average Max Score reached by Swingy Monkey. Note how the effect of gravity is nearly uniformly positive.

Given more time, we would have wanted to explore the additional effect of running Swingy Monkey for more than 100 iterations.

## 3 Discussion

With more time, we would have liked to study balancing train time versus maximizing eventual performance. Specifically, we hypothesize that an agent in a larger, more complicated state space will take longer to reach reasonable performance, but after significant training will eventually outperform an agent using a simpler state representation. We used both experimentation and reasoning to decide on our state representation, balancing how quickly we wanted the agent to learn and maximizing eventual performance. Formally analyzing how adding complexity to the state representation affects these two outcomes could be very insightful.

Moreover, we would have liked to use neural networks for function approximation to represent the value function or the Q-function. DeepMind Techologies illustrated how Convolutional Neural Networks can be applied to Atari games, and their research encourages applying a similar approach to *Swingy Monkey*.