Practical 4: Reinforcement Learning

Linying Zhang, Yi Ding, Danny Zhuang
Usual kaggle group: LYD

1 Introduction

This practical focuses on using reinforcement learning method to develop an agent to play the game Swingy Monkey, a game similar to Flappy Bird. The agent can make the monkey jump to a new vine, or swing down the current vine he is holding, in order to pass the tree trunks coming from both the top and the bottom of the screen. Positive reward is given when the monkey successfully passes tree trunks without hitting them, and different negative rewards are given when the monkey falls off the bottom of the screen, jumps off the top, or hit a tree trunk. A better agent is defined as an agent that can achieve higher average score and max score within a certain number of game iterations. Our goal is to train a good agent through various reinforcement learning method (SARSA and Q-learning), and compare their performance with different policy and hyper-parameters on discretized space.

2 Technical approach

2.1 Space Discretization

Because the original state space is in high-dimension (600x400 pixels), almost continuous, we tried to reduce the state space by discretizing the position space into bins with width 100. (any work done on choosing this bin size?). With slight modification on the original states, our states are:

a. monkey-to-next-tree: monkeys horizontal distance to next tree

b. monkey-to-tree-top: the vertical distance from the top of the monkey to the top tree trunk

c. monkey-to-tree-bottom: the vertical distance from the top of the monkey to the top tree trunk

d. gravity: gravity is randomly initialized in each epoch and remains the same in an epoch. This is not given directly but can be calculated easily through the change of monkeys vertical velocity. Models were tested with and without this state

2.2 Model Building

2.2.1 Rationale for model-free approach

Model-based approach only shows good performance when there is good prior knowledge on the reward model $r(s,a)$ and transition model $p(s'|s,a)$, which are hard to estimate for a high-dimensional problem like this. Model-free approach skips this estimation step, and directly learns the policy from samples of the world. So we chose to evaluate the performance of some model-free approaches, eg. SARSA and Q-learning, in tackling this problem.

2.2.2 Overview of model-free methods

Both SARSA and Q-learning are model-free RI methods, which are differed by the way they select the a' to update towards. Recall the alternate form of Bellman equation is used to construct the loss function:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{s,a} ||Q(s,a;\mathbf{w}) - [r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) \underset{a'}{max} Q(s' \mid a';\mathbf{w})]$$

where γ is a discounting factor 0¡γ¡1 After taking the gradient of the sampled loss and approximating the expection $s' \sim p(s' \mid s,a)$, the update rule for a given state action pair is

$$\frac{\partial \mathcal{L}^{(s,a)}(\mathbf{w})}{\partial w_{s,a}} \approx Q(s,a;\mathbf{w}) - [r(s,a) + \gamma \underset{a'}{max} Q(s',a';\mathbf{w})]$$

Based on the above update rule, every time the monkey starts at a state $sinS$ select action a based on the policy, collect $r \leftarrow r(s,a)$ and next state s' from environment, consider new action a' the step where SARSA and Q-learning differs, update the estimate of $Q(s,a)$ function through $w_{s,a}$, repeat at s.

2.2.3 SARSA

SARSA is an on-policy update. Instead of max a, it uses action from policy π to update $Q(s,a)$.

$$\frac{\partial \mathcal{L}^{(s,a)}(\mathbf{w})}{\partial w_{s,a}} \approx Q(s,a;\mathbf{w}) - [r(s,a) + \gamma Q(s',\pi(s');\mathbf{w})]$$

$$w_{s,a} \leftarrow Q(s,a;\mathbf{w}) + \eta[r + \gamma Q(s',\pi(s');\mathbf{w}) - Q(s,a;\mathbf{w})]$$

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

where where η is learning rate, 0< η <1 Notice that the update of $w_{s,a}$ can be seen as a weighted average of current reward and discounted future reward with future action a determined by the policy π. The advantage of using SARSA is that it converges faster but disadvantage is that it updates towards worse policy.

2.2.4 Q-Learning

Q-learning is an off-policy update rule. The future action a is always the action at s that maximize Q. The update at each step is

$$w_{s,a} \leftarrow Q(s,a;\mathbf{w}) + \eta[r + \gamma \underset{a'}{max} Q(s', \pi(s'); \mathbf{w}) - Q(s,a;\mathbf{w})]$$

The off-policy character makes Q-learning worse at convergence, but can learn better policy.

2.2.5 Policy

Exploitative policy
When ϵ = 0, the policy is exploitative policy, under which SARSA and Q-learning are technically using the same rule at selecting a, and thus their performance were not expected to differ. We implemented both approaches under optimal policy to confirm our hypothesis and contrast it to the ϵ-greedy policy.

ϵ-greedy policy
Because we only have limited access to the world, the agent has to balance between getting current maximum reward (exploitation) versus learning the world to improve estimation (exploration). We find this balance by using epsilon-greedy policy, under which the optimal policy is taken with probability $1 - \epsilon$, and a uniformly random action is taken to explore the world with probability ϵ. We first tested models with fixed epsilon, and then changed to decreasing epsilon by decreasing epsilon by half after each random action. in each training cycle. So as the agent learns more and more about the world, its policy converges to optimal policy.

3 Result

A total of five models were implemented including:

1 Model with Exploitative Policy

2 Q-Learning with constant ϵ-greedy Policy

3 SARSA with constant ϵ-greedy Policy

4 Q-Learning with decreasing ϵ-greedy Policy

5 SARSA with decreasing ϵ-greedy Policy

where $\epsilon = 0.1$for initial setting across all implementaion.

We tune learning rate η and future discount factor γ with η = [0.2, 0.4, 0.6, 0.8] and γ = [0.2, 0.4, 0.6, 0.8]. Best score and Highest score for each model under each parameters setting were presented with heatmap in Figure 1-3. Max Best score and Highest score and correspongidng parameters setting were summarized in Table 1.

Table 1: Max best and average score for 5 models, η : Learning rate, γ : Discount

Model	Max Best Score	Max Average Score
EXPLOITATIVE POLICY	1177 $\eta = 0.2, \gamma = 0.6$	24.8 $\eta = 0.4, \gamma = 0.8$
CONSTANT ϵ-GREEDY POLICY: Q-LEARNING	25 $\eta = 0.2, \gamma = 0.2$	1.1 $\eta = 0.2, \gamma = 0.8$
CONSTANT ϵ-GREEDY POLICY: SARSA	19 $\eta = 0.2, \gamma = 0.2$	0.9 $\eta = 0.2, \gamma = 0.8$
DECREASING ϵ-GREEDY POLICY: Q-LEARNING	1573 $\eta = 0.2, \gamma = 0.8$	57.2 $\eta = 0.2, \gamma = 0.6$
DECREASING ϵ-GREEDY POLICY: SARSA	1317 $\eta = 0.2, \gamma = 0.8$	52.3 $\eta = 0.2, \gamma = 0.8$

Model Comparison
We found that Q-Learning with decreasing ϵ-greedy policy outperformed all other methods with highest best score and highest average score (Table 1). Comparing ϵ-greedy policy with fixed or decreasing ϵ, we found that both SARSA and Q-learning models with decreasing ϵ-greedy policy performedand much better than those with fixed ϵ: the highest average score from parameter tuning is around 50 for models with decreasing ϵ, almost 50 times higher than those with fixed ϵ. Models with exploitative policy were slightly worse than models with decreasing ϵ-greedy policy. Comparing SARSA and Q-Learning, we didn't find any significant difference between these two approaches in terms of max score and average score, which can be explained by the policy we used that created balance between exploitation and exploration for both approaches. Due to the limited time, tuning was done only once for each parameter pair and we were aware of the variations that existed due to the stochastic process of training.

Impact of Learning Rate and Future Discount Factor
In general, we found that large future discounting factor and small learning rate led to the best performance of each model tuned, which was demonstrated by the darker color aggregated on the top right corner of the heatmap (Figure 2). Although only showing the heatmap corresponding to the tuning of of SARSA and Q-Learning with decreasing

ϵ-greedy policy, we found that large future discounting factor and small learning rate remained to be the best combination across all other models with different policy.

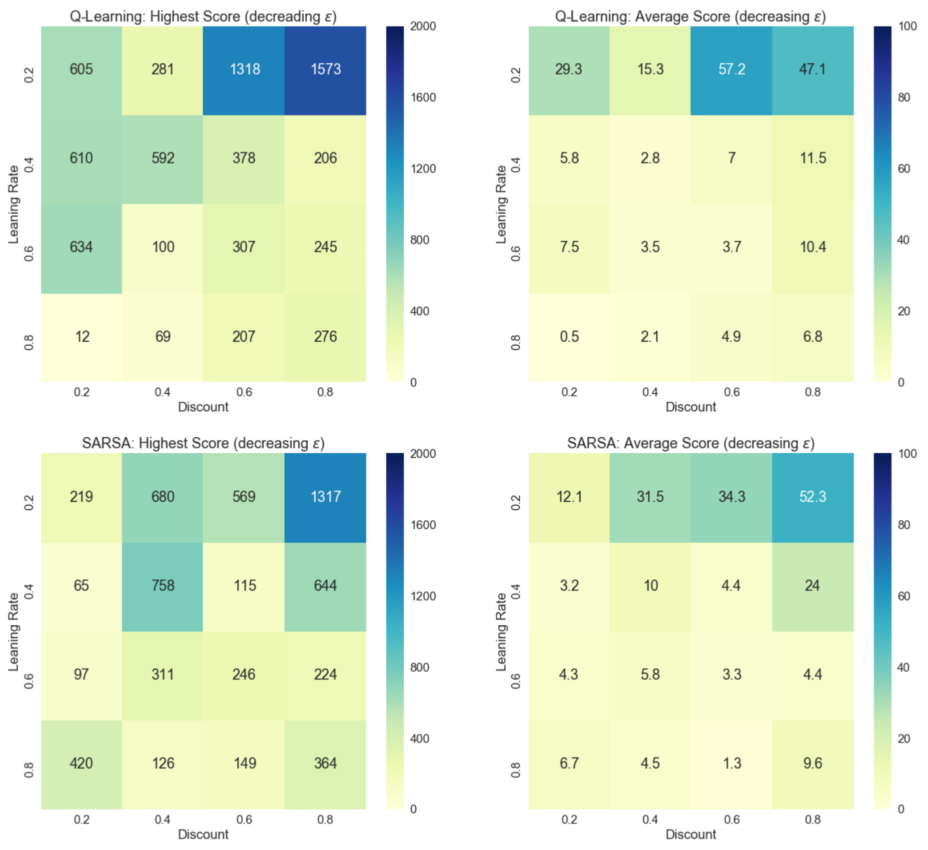

Figure 1: Q-Learning and SARSA with decreasing ϵ, initial $\epsilon = 0.1$

Distribution and trend of score from our best model
Based on the score histogram from 500 epochs of Q-Learning training, we found that majority of the scores (80 percent) were less than 25, 1-percent of the times the agent scored higher than 600. (Figure 3). When we plotted the average score for each 20 epochs, the scores were highly fluctuating, but the general trend of the score was increasing as training processed. If we had more epochs or increased the size of the bin, the overall trend should be more clearer with less fluctuations towards the end to better show the convergence.

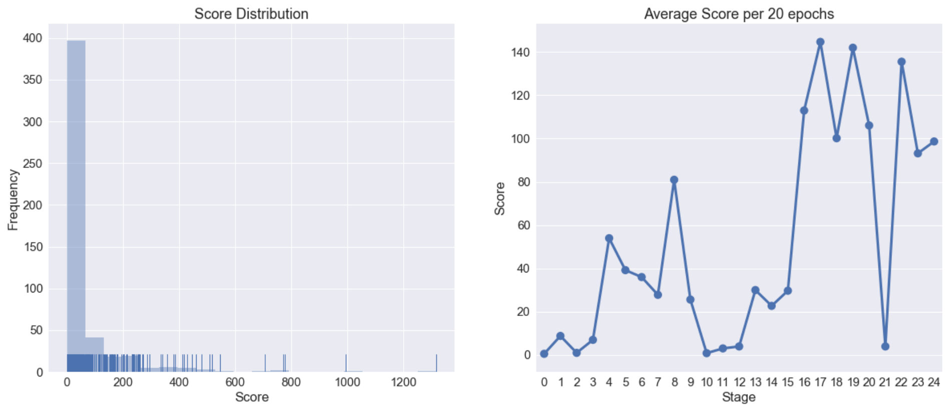

Figure 2: Q-Learning with decreasing ϵ, initial $\epsilon = 0.1$, $\eta = 0.2$, $\gamma = 0.8$

4 Discussion

Decreading ϵ vs. Constant ϵ Here we observed that model with decreasing *epsilon* is substatially better than constant ϵ. Large ϵ is desirable in the early stage, because it allows us to explore more state-action pairs to have better estimated $Q(s, a)$. However, in the later stage after we have got good $Q(s, a)$ estimation, it makes better sense to be exploitative to get high score. Keeping using random choice of *Swing* or *Jump* will lead to failure of the game.

SALSA vs Q-Learning We didn't observe big discrepancy in performance between SALSA and Q-Learning under decreasing ϵ-greedy setting. This is due to the fact that our ϵ became very small after several epochs. At that time, ϵ-greedy became a totally greedy policy, and $Q(s', \pi(s'))$ for SARSA is equal to $\underset{a'}{max} Q(s', a')$ for Q-Learning.

Exploitative Policy vs decreading ϵ-greedy Policy Also due to the decreading ϵ, when ϵ is close to zero, ϵ-greedy policy implemented in SARSA, Q-Learning became exploitative, that's why the three models had close performance. However, when Q-Learning and SARSA would be able to explore more state-action pairs to improve $Q(s, a)$ estimation in early epochs when ϵ was still large, which made them perform better than model with exploitative policy in later epochs.

5 Appendix

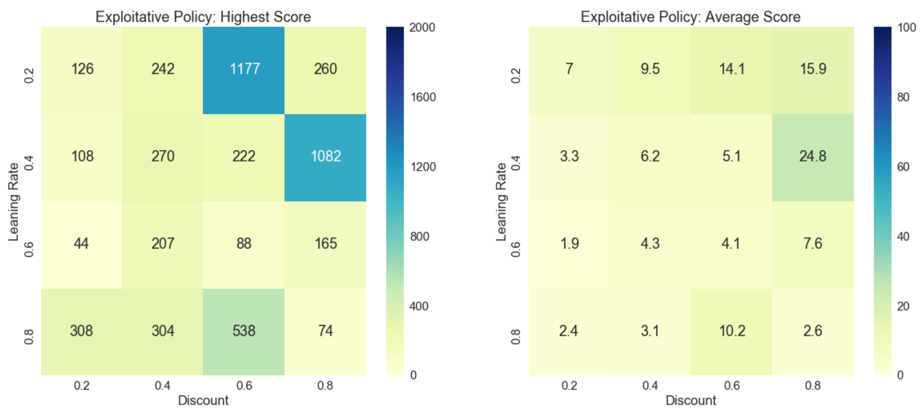

Figure 3: Exploitative Policy

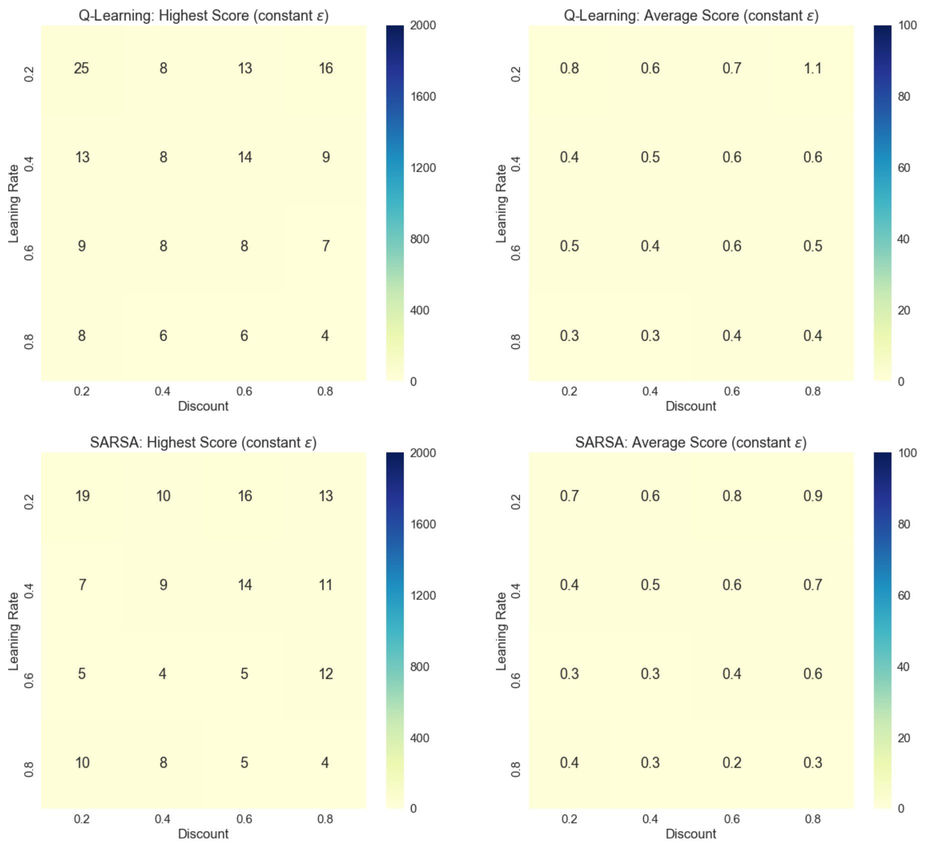

Figure 4: Q-Learning and SARSA with constant $\epsilon = 0.1$