# ECE5242 Project4: Reinforcement learning

SIJIA GAO

## 1 Introduction

This project specifies reinforcement learning algorithms for simple and complex MDP environments. For a simple Maze MDP, we implement value iteration and Q learning. For continuous state space MDP in OpenAI Gyms, we implement REINFORCE, Hill Climbing and Q learning with discretized state space.

## 2 Simple Maze Problem

We start from a simple Maze problem (see Fig.1(a)): the agent starts from "S" and its goal is to collect flags in "F" and escape the maze through "G" as soon as possible. The agent only receives a reward equivalent to the number of flags it has collected at "G". Agent's state contains cell position and flag status and its action is chosen from $\{up, down, left, right\}$. To learn the optimal policy, we implement value iteration and Q learning described as follows.

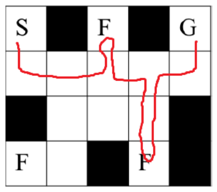
### 2.1 Value Iteration Algorithm

The value iteration algorithm flows as: Choose the number of iterations $N$ (typically large). For iterations $k = 1, 2, \dots N$, compute

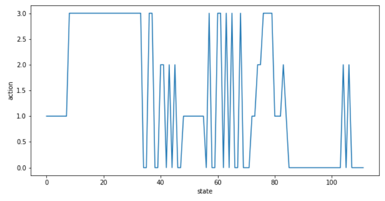
$$\begin{aligned} Q_k(i,u) &= r(i,u) + \gamma \sum_j P_{i,j}(u) V_{k-1}(j) \\ V_k^i &= \min Q_k(i,u), \mu_k(i) = Q_k(i,u) \end{aligned} \tag{1}$$

Here, $i$, $u$ denote state and action, respectively. $r(i,u)$ denotes the reward associated with state $i$ and action $u$. $\gamma$ is the discounted factor. $\mu_N(i)$ is the optimal policy.

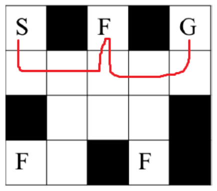
**Results and Discussion** Fig.1 illustrates performance of value iteration algorithm for Maze problem. Results are shown with two difference discounted factors $\gamma$. Agent picks up fewer flags when $\gamma$ goes lower. The trade off in this problem is spending more time in maze for more flags and escaping the maze. Since agent can only get reward at $G$, lower $\gamma$ will force the agent to make a bias towards escaping the maze.



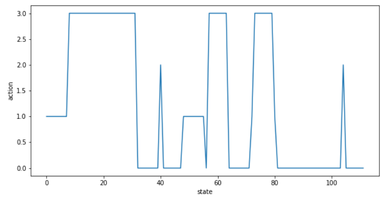
(a)



(b)



(c)



(d)

Figure 1: Discounted factor=0.9: (a) Maze and optimal path (b) Optimal policy using value iteration. Discounted factor=0.5: (c) Maze and optimal path (d) Optimal policy using value iteration

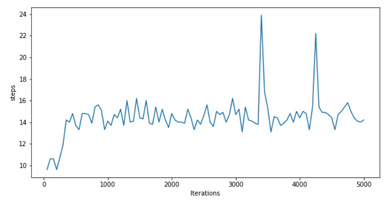
## 2.2 Q learning

Q learning is a model-free reinforcement learning algorithm that learns a better policy for the agent to take. It operates by stepping forward in the MDP and refreshing the Q table. Q table includes a set of $Q(s,a)$ where $s$, $a$ denote the state and action, respectively. Q learning is described as follows

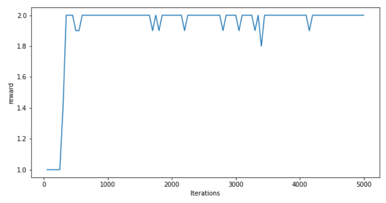
$$\begin{aligned} &\text{Sample } a_t \text{ using } \epsilon - \text{greedy} \\ &Q(s_t,a_t)^{new} = (1-\alpha)Q(s_t,a_t)^{old} + \alpha(r_t + \gamma \max_a Q(s_{t+1},a)) \end{aligned} \tag{2}$$

Here, $\epsilon$-greedy action selection tells that with $\epsilon$ probability we utilize random selection from action set; with (1-$\epsilon$) probability, we set $a_t = \arg\max_a Q(s_t,a)$. $\alpha$ denotes the learning rate. $\gamma$ denotes the discounted factor.

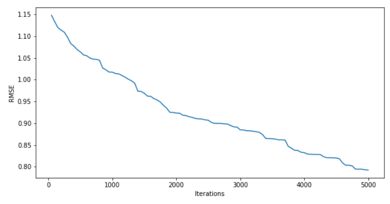
**Results and Discussion** Fig.2 presents results for Q learning implementation on Maze problem. Results are evaluated using average steps, reward and RMSE to optimal Q computed in Sec.2.1. Fig.3 illustrates results under different hyperparameters: action selection $\epsilon$ and learning rate $\alpha$. We can see that higher $\epsilon$ results in faster convergence and lower RMSE. The reason is that higher $\epsilon$ gives agent more random choice to explore the environment. Higher $\alpha$ gives lower RMSE but slower convergence. $\alpha = 0.1$ results in fastest convergence. Higher $\alpha$ means we take into account more TD error to refresh Q table rather than a trivial perturbation on the old Q table. This gives more accurate result but also slower convergence.



(a)



(b)



(c)

Figure 2: Results for Maze problem using Q learning with learning rate $\alpha = 0.2, \epsilon = 0.5$ and $\gamma = 0.9$.

## 3 Continuous state space

In this project, we work with Acrobot-v1 and MountainCar-v0 environment in OpenAI GYm with continuous state space and discrete action space.

### 3.1 REINFORCE

REINFORCE is a policy gradient approach to find the optimal policy. The policy $\theta$ is optimized iteratively as

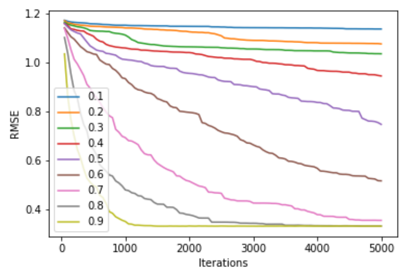
$$\theta = \theta + \alpha \nabla_\theta log \pi_\theta(s_t, a_t) G_t$$

Here, $G_t$ is the discounted reward (advantage). REINFORCE is achieved using 3-layer neural network in Pytorch. The size of hidden layer is 16. We utilize rectifier and softmax as activation function. Neural network takes state as the input and gives soft estimates of actions.

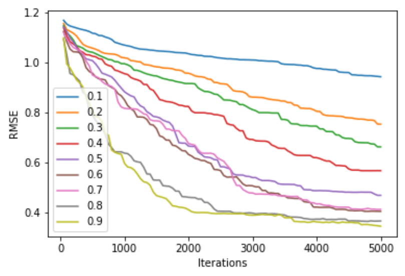
**Results and Discussion** Results under Acrobot-v1 environment are shown in Fig.4. Debugging size of hidden layer is important in this problem.

### 3.2 Hill Climbing

Hill climbing is a random search technique that starts from an arbitrary solution and find a better one by making increments to the current solution. We implement Hill Climbing in both Acrobot-v1 and MountainCar-v0 environment. The

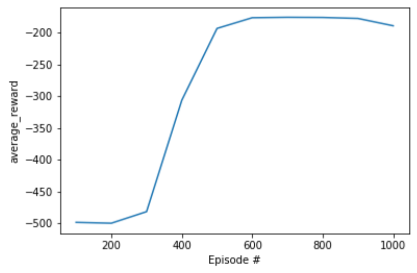


(a)

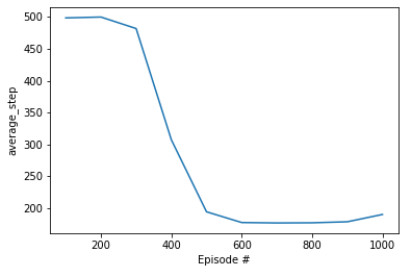


(b)

Figure 3: RMSE under a sweep of $\epsilon$ (a) and learning rate $\alpha$ (b)



(a)



(b)

Figure 4: Performance of Acrobot-v1 environment using REINFORCE.

key is to estimate a weight matrix $w$ with rows representing state and columns representing actions. Hill Climbing is illustrated as follows:

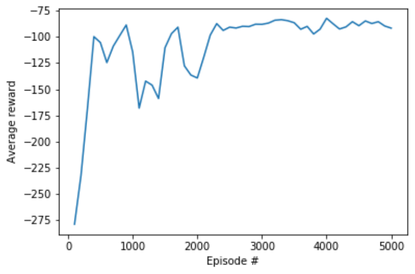
Initialize $w$, for each iteration do:

1. Initialize state
2. Choose $i$th action if $i = \arg\max_i exp(c(i))/\sum exp(c(i)), c = state * w$, get reward and step reward until termination
3. Compute discounted reward $R$ and update policy: if $R$ is higher than up-to-date R,

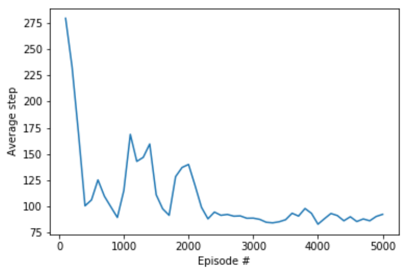
**Results and Discussion** Results using Hill Climbing are shown in Fig.5. Be careful with noise scale selection.

## 3.3 Discretize state space

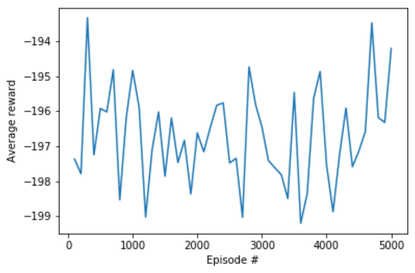
A natural way to solving continuous state space is to discretizing. Under MountainCar-v0, the state space is continuous in 2-dimension and ranges from



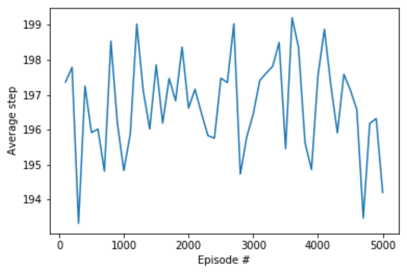
(a)



(b)

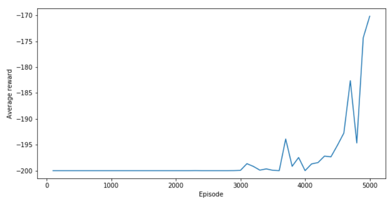


(c)

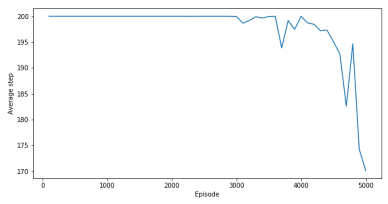


(d)

Figure 5: Performance for Acrobot-v1 (a)(b) and MountainCar-v0 (c)(d) using Hill Climbing.



(a)



(b)

Figure 6: Performance for MountainCar-v0 by discretizing state space and run Q learning.

[-1.2, -0.07] to [0.6, 0.07]. Therefore, we consider discretizing the 1st dimension in 0.1 stepsize and second dimension in 0.01 stepsize. Other procedures are similar to those in conventional Q learning. Epsilon is reduced (bias towards action selection from Q table rather than random selection) by a stable amount every episode because we are approaching the optimal policy. Results are shown in Fig.6.