# ECE 276C Assignment 1 Report

Mingwei Xu A53270271

Fall 2019

## 1 Question 1 - Model based methods

### 1.1 Describe Environment

The environment can be described as below:

- State space: $s \in \{0, 1, 2, 3, ..., 15\}$ representing the index of grid from top-left to bottom-right, each state can be either a $\{S, F, H, G\}$ representing start, frozen space, hole or goal. Terminal state is $s(t) = H$ or $G$.
- Action space: $u \in \{0, 1, 2, 3\}$ that enumerates $\{left, down, right, up\}$
- Reward function: $reward = 1$ if $s = 15$ $(G)$ else $reward = 0$

State transition is not deterministic in this environment, since given a state and action, the probability of getting into one desired next state is not 1.

### 1.2 Derive Value Function

Derivation of value function given a deterministic policy is given below:

$$\begin{aligned} &v(s) = \mathbb{E}_\pi[\textstyle\sum_{k=t}^{\infty} \gamma^{k-t} R_k | S_t = s] \\ &= \mathbb{E}_\pi[R_t + \gamma \textstyle\sum_{k=t+1}^{\infty} \gamma^{k-t} R_k | S_t = s] \\ &= \textstyle\sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma \mathbb{E}_\pi[\sum_{k=t+1}^{\infty} \gamma^{k-1} R_k | S_{t+1} = s']] \\ &= \textstyle\sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v(s')] \end{aligned} \tag{1}$$

### 1.3 Test Policy

Given the policy $\pi(s) = (s + 1)\%4$, success rate is about 0.02. It shows that using some naive policy can hardly let the agent reach the goal.

### 1.4 Learn Model

This function estimates the transition probability and reward function using $10^5$ random samples.

Here I store them into numpy matrices, where the transition probabilities and reward function are stored in two $16 \times 4 \times 16$ matrices separately, where the 3 dimensions represents $(s, a, s')$

Since the matrices are too large, please see **p_mat** and **r_mat** in **p1_main.py**

**Policy iteration (using iterative policy evaluation)**

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   *policy-stable* $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      *old-action* $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

## 1.5 Policy Iteration

The policy iteration algorithm is defined below:

The results of 50 iterations are shown in "Fig. 1"

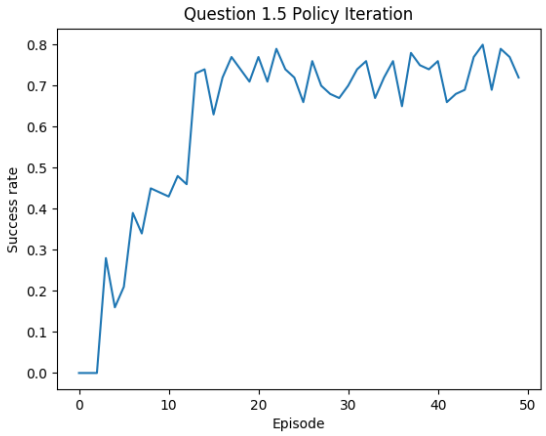


Figure 1: Policy Iteration Result

From the results we can see the policy converges pretty well, and the final success rate is about 0.7 0.8.

## 1.6 Value Iteration

The value iteration algorithm is defined below:

The results of 50 iterations are shown in "Fig. 2"

From the results we can see the policy converges pretty well, and the final success rate is about 0.8. More over, the plot looks very similar to policy iteration, which means they have similar performance in this environment.

**Value iteration**

Initialize array $V$ arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

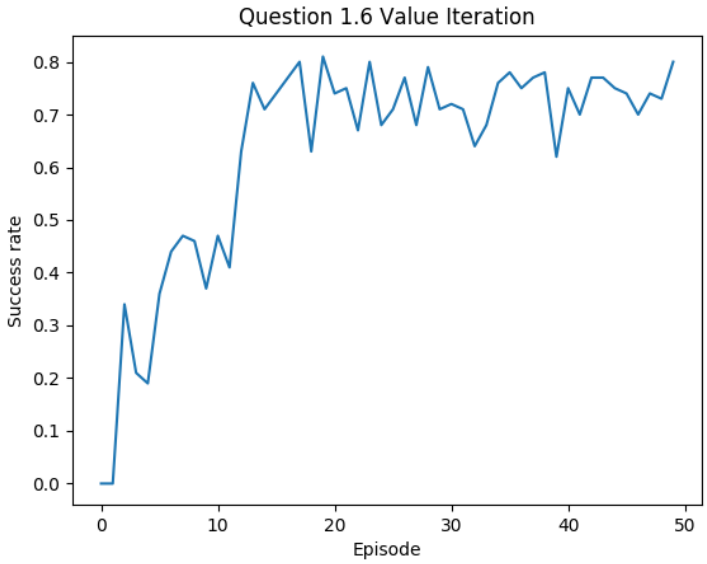


Figure 2: Value Iteration Result

# 2 Question 2 - Model free methods

## 2.1 Q-learning

Here we use Q-learning for model free learning. The Q-learning algorithm is shown below:

**Q-learning: An off-policy TD control algorithm**

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize $S$
  Repeat (for each step of episode):
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Take action $A$, observe $R$, $S'$
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until $S$ is terminal

At first, we do exploration using linearly annealed $\epsilon - \textit{greedy}$ method, that takes random action with probability $1 - e/5000$ where $e$ is the number of current episodes.

**Fix discount factor** With fixed discount factor $\gamma = 0.99$, the results are shown in "Fig. 3"

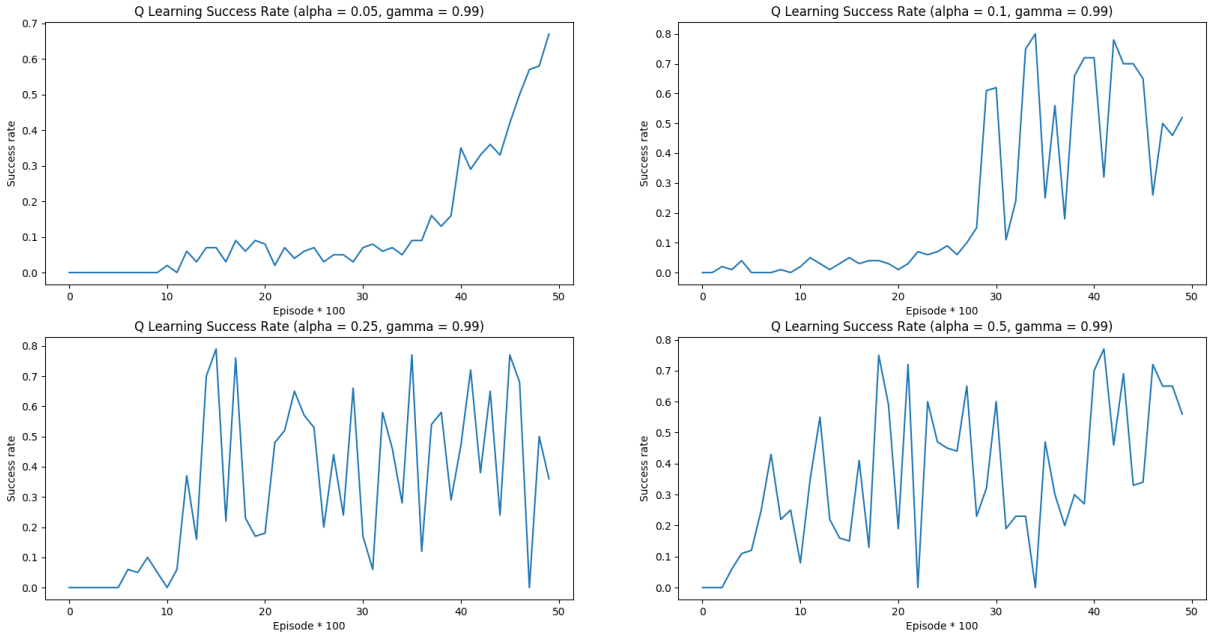


Figure 3: Q Learning Result $\gamma = 0.99$

From the results we can see as learning rate $\alpha$ increases, the agent may learn faster, but often has more vibration and less consistent result. The result tends to be more robust when $\alpha$ is as low as 0.05.

**Fix learning rate** With fixed learning rate $\alpha = 0.05$, the results are shown in "Fig. 4"

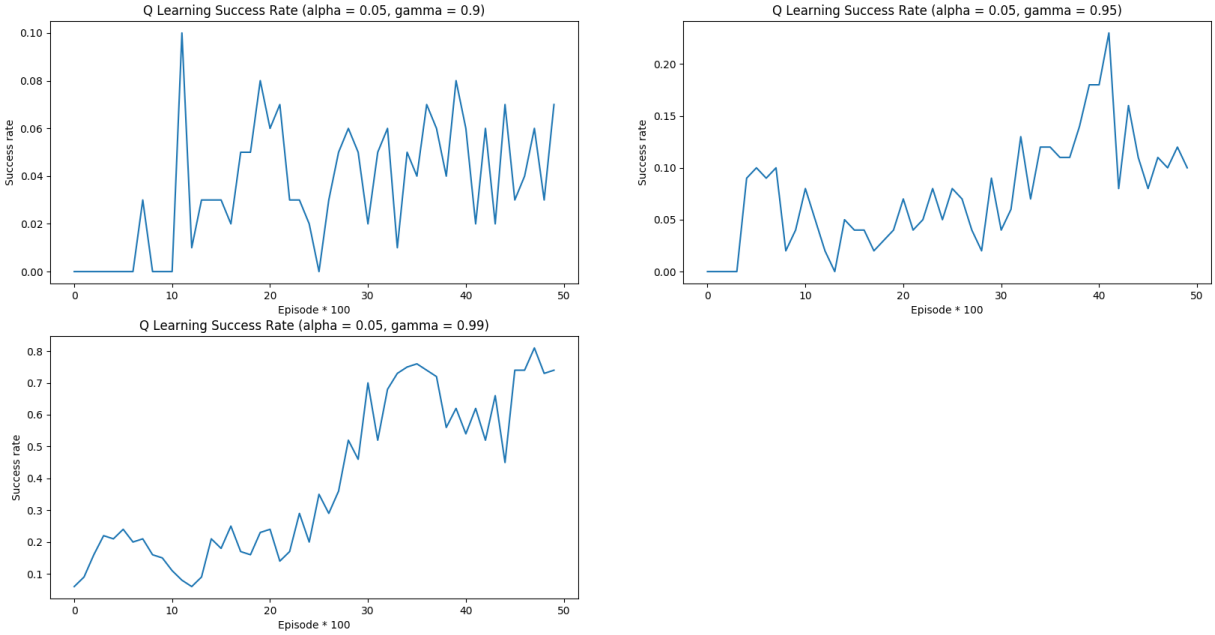


Figure 4: Q Learning Result $\alpha = 0.05$

From the results we can see as discount factor $\gamma$ increases, the result tends to be better. The best result happens when $\gamma = 0.99$

When discount factor is large, the agent will foresee the situation more and emphasize more on the long-term reward. When discount factor is small, the agent will be more greedy and focus on short-term reward.

Since in this environment, the agent can only get reward when it reaches the final goal, so it makes since that large discount factor has better result.

## 2.2 Different exploration strategy

Here, we implemented a different exploration strategy: Adaptive $\epsilon - Greedy$, proposed by Michel Tokic [1]. The exploration probability $\epsilon(s)$ here is dependent to the value-function error, enables the agent to adapt the exploration rate based on the value function changing behavior. $\epsilon(s)$ is regularized by a softmax function of value function errors, as shown in the equation below:

$$f(s,a,\sigma) = \left| \frac{e^{\frac{Q_t(s,a)}{\sigma}}}{e^{\frac{Q_t(s,a)}{\sigma}} + e^{\frac{Q_{t+1}(s,a)}{\sigma}}} - \frac{e^{\frac{Q_{t+1}(s,a)}{\sigma}}}{e^{\frac{Q_t(s,a)}{\sigma}} + e^{\frac{Q_{t+1}(s,a)}{\sigma}}} \right| = \frac{1 - e^{\frac{-|Q_{t+1}(s,a) - Q_t(s,a)|}{\sigma}}}{1 + e^{\frac{-|Q_{t+1}(s,a) - Q_t(s,a)|}{\sigma}}} \tag{2}$$

Then the exploration probability can be updated using:

$$\varepsilon_{t+1}(s) = \delta \cdot f\left(s_t, a_t, \sigma\right) + (1 - \delta) \cdot \varepsilon_t(s) \tag{3}$$

Using $\alpha = 0.1$ and $\gamma = 0.99$, which is determined to be the best parameter using grid search, we compare the previous algorithm and new algorighm using adaptive $\epsilon - Greedy$. For adaptive $\epsilon - Greedy$, we use $\delta = 0.25$ and $\sigma = 0.6$. The results of previous algorithm is shown in "Fig. 5", and the results of adaptive $\epsilon - Greedy$ is shown in "Fig. 6".

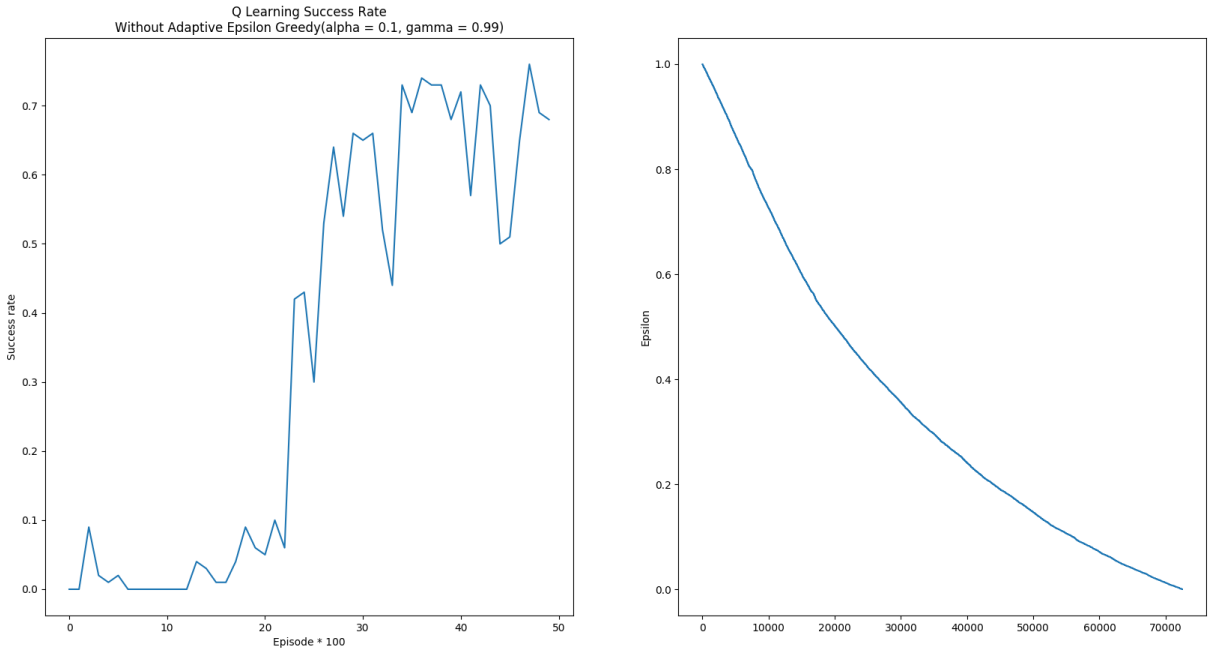


Figure 5: Q Learning Without Adaptive $\epsilon - Greedy$

From the results we can see the results of adaptive $\epsilon - Greedy$ is slightly better, with final success rate around 0.7 0.8. $\epsilon$ reduces pretty fast among the first several episodes, however oscillate a lot as training goes on. This behavior may due to the very noisy gym environment which causes relative large change of value function, and the adaptive method overshoot to tune $\epsilon$.

As a consequence, the new exploration method results in better peak success rate, but less consistent performance.

# References

[1] Tokic M. (2010) Adaptive -Greedy Exploration in Reinforcement Learning Based on Value Differences. In: Dillmann R., Beyerer J., Hanebeck U.D., Schultz T. (eds) KI 2010: Advances in Artificial Intelligence. KI 2010. Lecture Notes in Computer Science, vol 6359. Springer, Berlin, Heidelberg

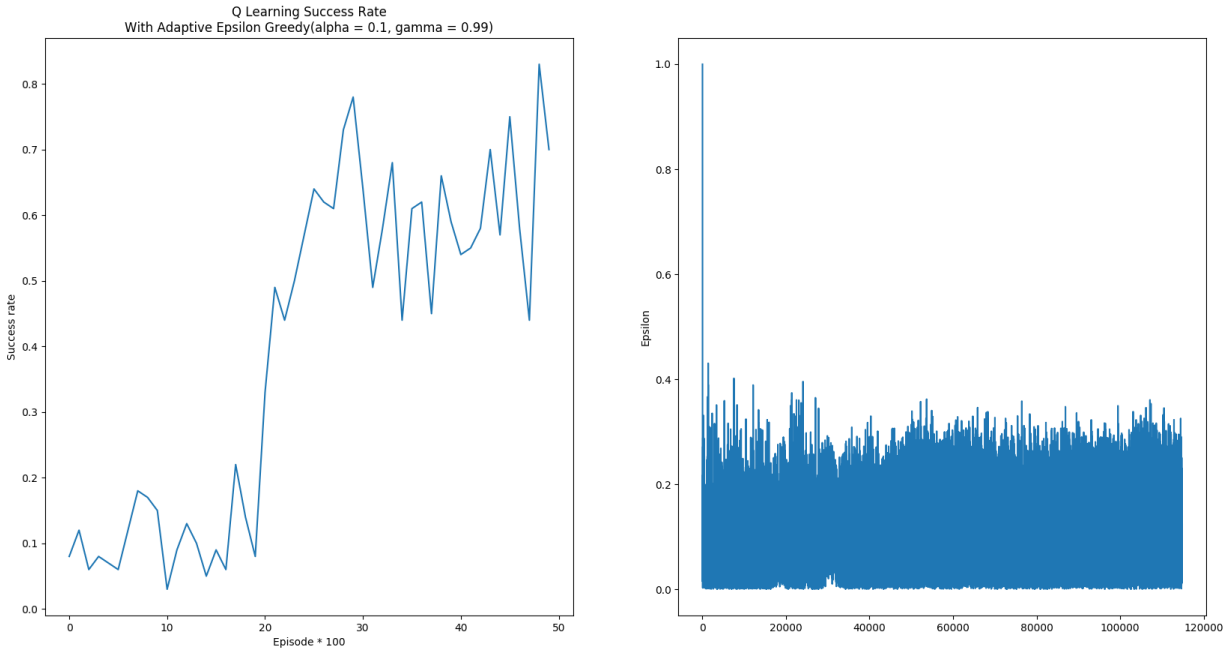


Figure 6: Q Learning With Adaptive $\epsilon - Greedy$