ECOLE
POLYTECHNIQUE
DE BRUXELLES

UNIVERSITÉ LIBRE DE BRUXELLES

Learning dynamics INFO-F-409
Assignment 3

Authors:
Liyuan WU
Student No. 000454520

Academic year 2018 - 2019

Contents

1 N-Armed Bandit

My table is number 1. Rewards of each arm follows normal distribution, I also accept negative reward, because considering practical situations, player could also loose money, this situation is when reward is negative.

1.1 Original standard deviations experiments

1.1.1 Average reward for each algorithm

In order to avoid randomness, I ran each algorithm 100 times, and plot its corresponding average reward in Figure 1. Corresponding algorithms are: Random, Greedy $\epsilon = 0$, Greedy $\epsilon = 0.1$, Greedy $\epsilon = 0.2$, Softmax $\tau = 1$, Softmax $\tau = 0.1$.

Average rewards are: [1802.16, 1835.00, 2203.50, 2163.08, 1972.17, 1932.39], shown as green points in Figure 1.

Standard deviations are: [40.98, 405.31, 68.28, 59.29, 38.55, 383.63]

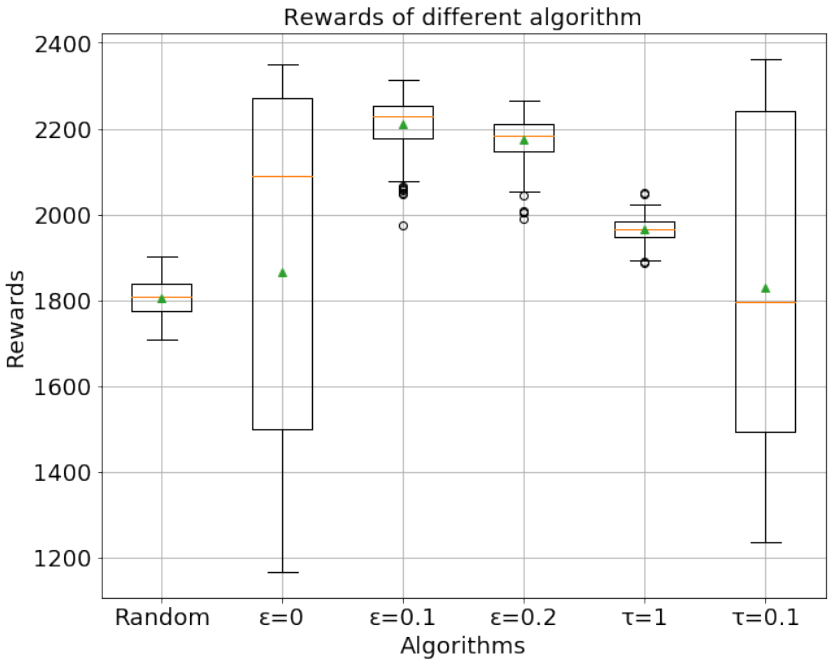

Figure 1: Box-plot of Average reward for each algorithm

1. The biggest average reward is greedy with $\epsilon = 0.1$, the worst algorithm is Random.
2. Standard variance of Greedy $\epsilon = 0$ and Softmax $\tau = 0.1$ are very big. This is because, for Greedy $\epsilon = 0$ algorithm, the first step, rewards of each arm are all zeros, thus, player will choose one arm randomly, if he obtains positive reward from this arm, on account of $\epsilon = 0$, only exploitation, no exploration, therefore, the player will continue choosing the arm which he chose at the first step. But sometimes, if the reward of first step is negative, player will choose the other

arms randomly at the second step.

3. For Softmax $\tau = 0.1$, the probability of choosing one arm is

$$Softmax(Q/\tau)$$

where

$$Softmax(x) = exp(x)/sum(exp(X))$$

when $\tau = 0.1$, it amplifies the probability of choosing the biggest reward arm, which means it almost only exploitation, no exploration. Therefore, its reward variance is as big as greedy with $\epsilon = 0$.

1.1.2 Arms average reward estimation over time

Arms average reward estimation over time plot is Figure 2.

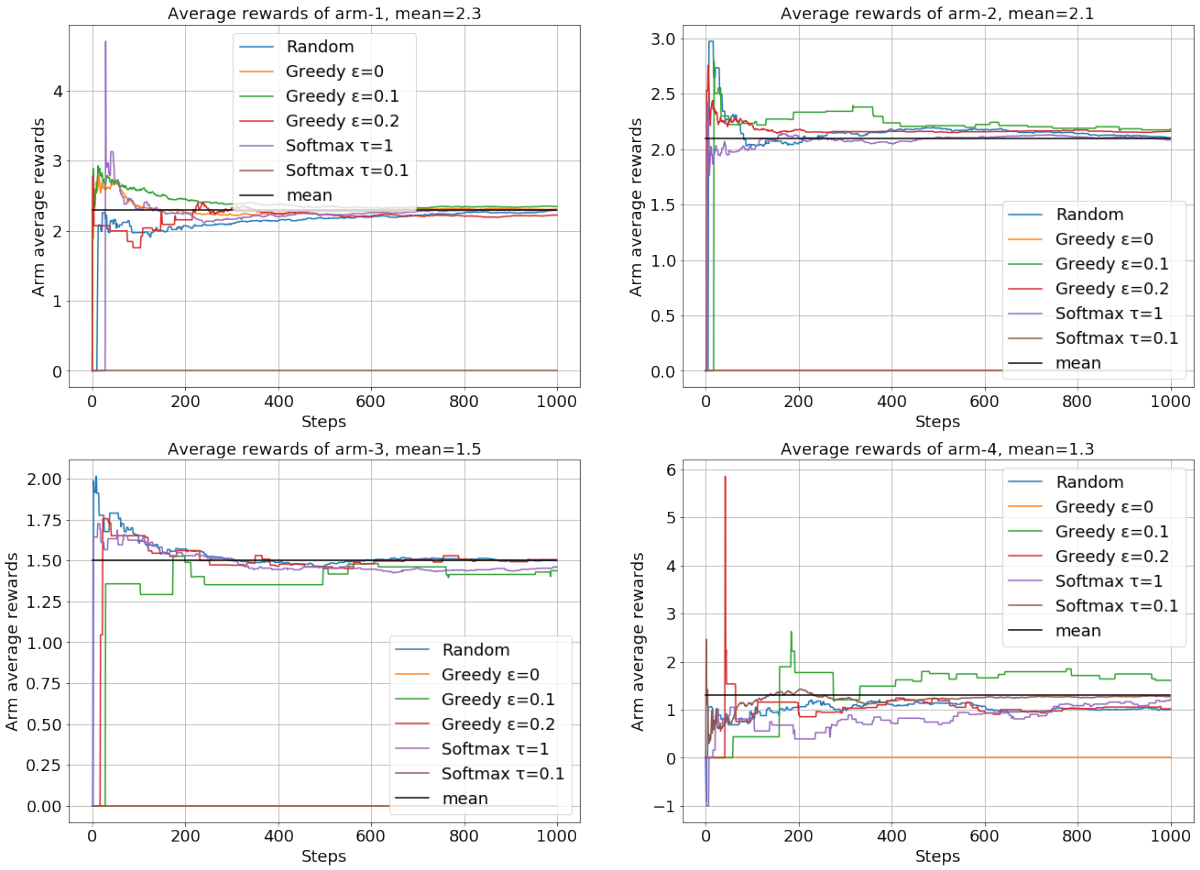

Figure 2: Arms average reward estimation over time

1. Generally, average estimation of arm 1, 2, 3 are good, but arm 4 is bad. This is because their corresponding deviation are [0.9, 0.6, 0.4, 2], arm 1, 2, 3 have small deviations but arm 4 has a big deviation.

2. For one experiment, random estimates mean value well because it chooses each arm with equally count, approximately 250 when steps is 1000. Greedy $\epsilon = 0$ and Softmax $\tau = 0.1$ can only estimate mean of their chosen arm. Greedy $\epsilon =$

0.1, 0.2 estimate arm 1 well, but the others bad, because most of the time, they choose arm 1, in Figure 3. Softmax $\tau = 1$ estimate arm 4 bad because it chooses less times on arm 4.

1.1.3 Number of times each action is selected

For each algorithm, its action selection has randomness, so I ran each algorithm 100 times, and result is Figure 3. The distribution standard deviation of algorithm Greedy $\epsilon = 0$ and Softmax $\tau = 0.1$ are bug, the others are small. The reason is the same as average reward for each algorithm. For these two algorithms, they only exploitation no exploration. Therefore, for each experiment, they almost always choose one arm, and for different experiment, they choose different arms.

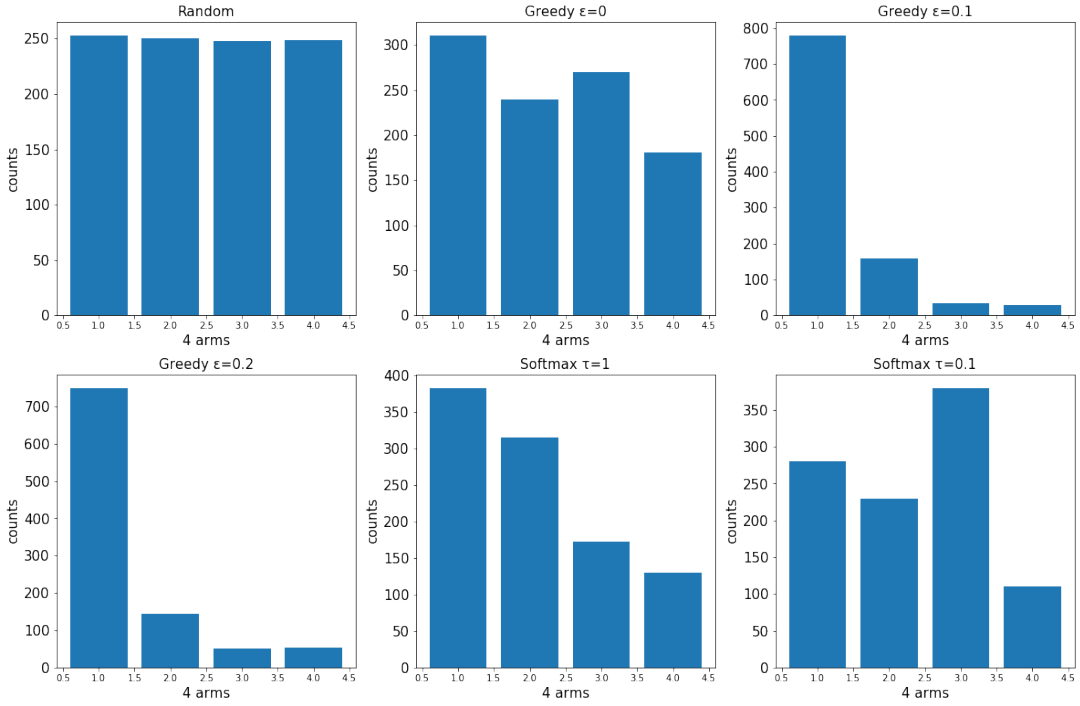

Figure 3: Number of times each arm is selected histogram

1.1.4 Best arm

My table is table 1, the best arm choice is arm 1. In order to answer: Which algorithms arrive close to the best arm after 1000 iterations? Which one arrives there faster? I draw the arm-1's chose percentage over time. But because every experiment is different, I posted 2 experiments result, as shown in Figure 4, and 5.

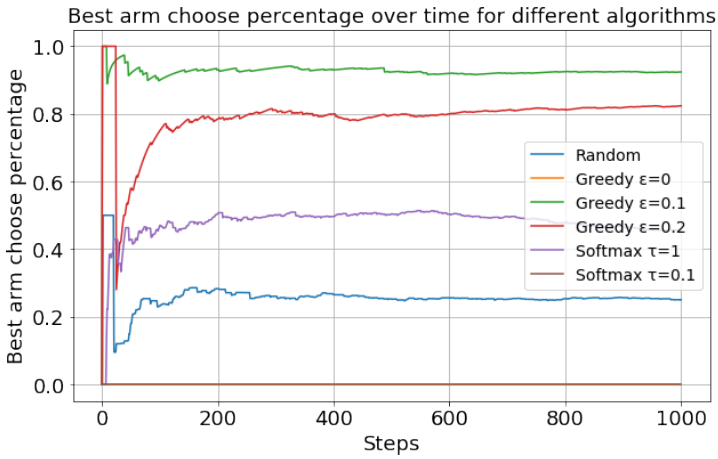

Figure 4: Best arm choose percentage over time, experiment 1

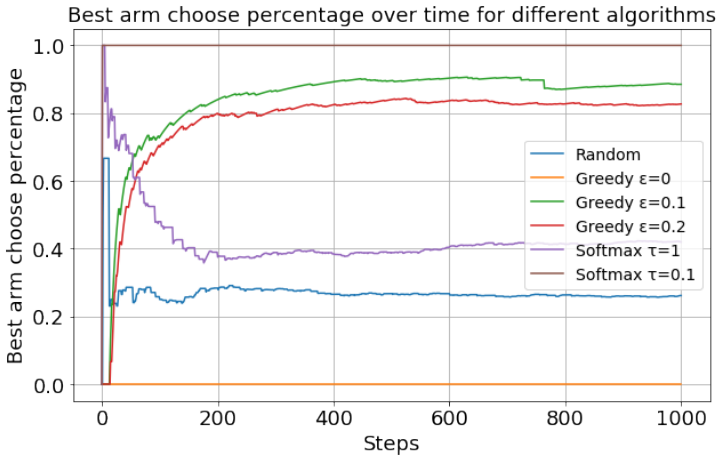

Figure 5: Best arm choose percentage over time, experiment 2

1. Algorithm Greedy $\epsilon = 0.1, 0.2$ and Softmax $\tau = 1$ always converge to best arm.
2. Random can not converge to best arm. Reason is evident.
3. According to Figure 3, arms selection counts distribution figure, Greedy $\epsilon = 0$ and Softmax $\tau = 0.1$ converge to best arm with an approximately probability $1/4$. Because they converge to one of four arms per experiment. In Figure 4, they all converge to arm 1, but in Figure 5, Greedy $\epsilon = 0$ converges to arm 1, Softmax $\tau = 0.1$ converges to arm 2.
4. Converge time only has meaning for Greedy $\epsilon = 0.1, 0.2$ and Softmax $\tau = 1$ algorithms. And their converge time is about 400 steps.

1.2 Doubling the standard deviations

1.2.1 Average reward for each algorithm

Result is in Figure 6

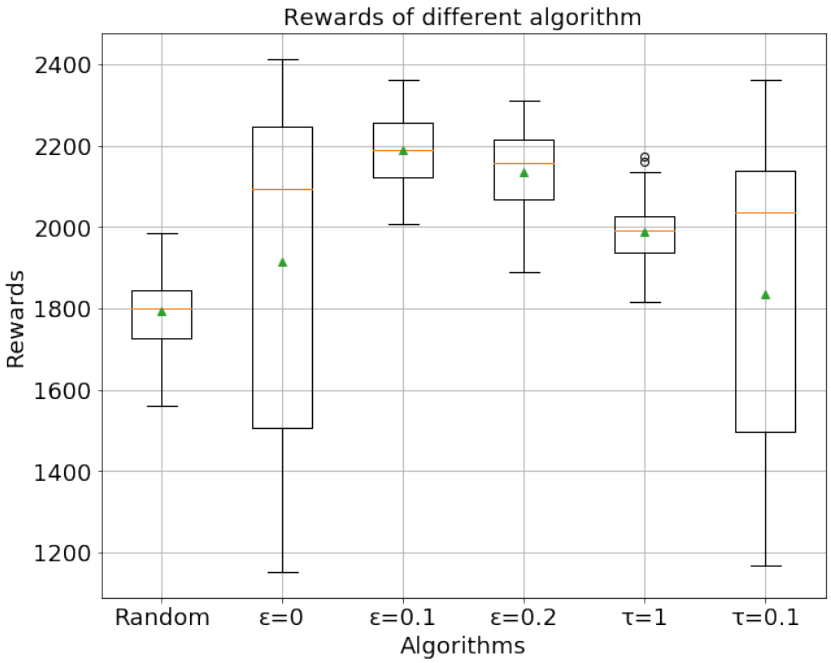

Figure 6: Box-plot of Average reward for each algorithm, Double std

Average rewards are: [1794.97, 1894.92, 2163.94, 2140.31, 1986.91, 1877.58]

Standard deviations are: [66.50, 379.86, 90.21, 86.01, 77.70, 369.76]

1. Comparing with original deviations, for doubled std result, average rewards are the same, but standard deviations augmented(apart from greedy $\epsilon = 0$ and Softmax $\tau = 0.1$). This is arms' rewards distribution deviations augment, which leads to the augmentation of total rewards.

1.2.2 Arms average reward estimation over time

After doubling deviations, arms rewards deviations become: [1.8, 1.2, 0.8, 4], I did 1500 steps for doubled deviation, as we can see in Figure 7, estimated means are farther from mean line, only mean of arm 3 can be well estimated.

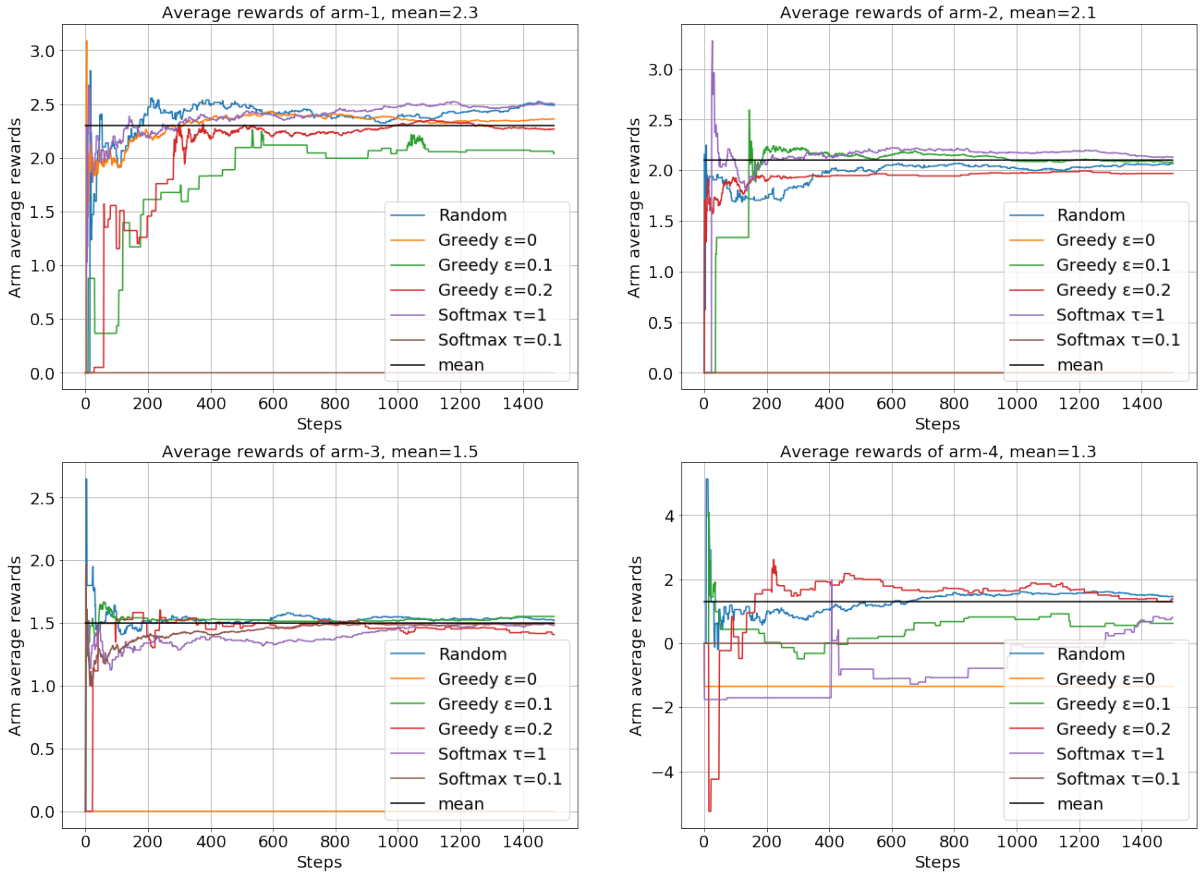

Figure 7: Arms average reward estimation over time, double std

1.2.3 Number of times each arm is selected

Each arm selected times histogram is shown in Figure 8. Times distribution of Random, Greedy $\epsilon = 0.1, 0.2$ and Softmax $\tau = 0.1, 1$ are the almost the same with original deviation result. But for Greedy $\epsilon = 0$, times of arm 4 decreases, this is because reward distribution of arm 4 becomes average 1.3, deviation=4, it has more possibility of getting negative reward. Therefore, greedy prefer not choose it.

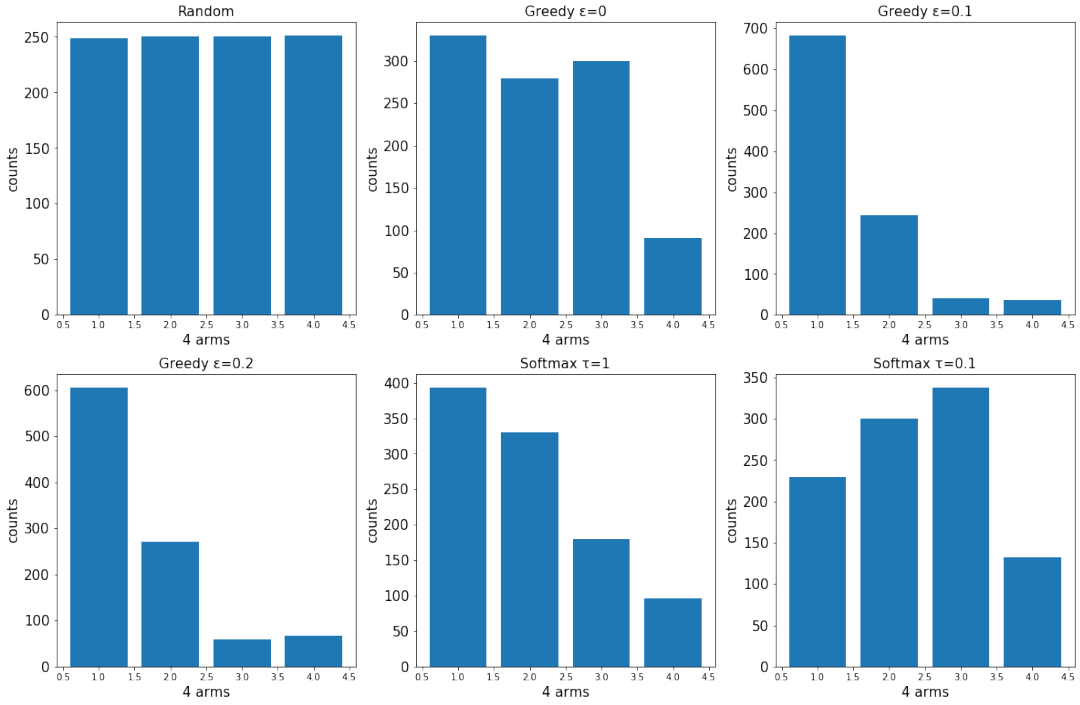

Figure 8: Number of times each arm is selected histogram, double std

1.2.4 Best arm

The same as before, I draw percentage of choosing arm 1 over time, but run 1500 steps. Result is in Figure 9.

1. About algorithms which converge to the best arm, result is the same as original deviation result. Algorithm Greedy $\epsilon = 0.1, 0.2$ and Softmax $\tau = 1$ always converge to best arm.
2. As for converge time, Greedy $\epsilon = 0.1, 0.2$ later than before, which is about after 800 steps. Because bigger deviation will change average rewards more greatly, just like shown in Figure 7 which will disturb greedy choose
3. Converge time of Softmax $\tau = 1$ doesn't change a lot. I think it is because Softmax will re-distribute each arm choosing probabilities using Softmax function, in this way, the choosing probabilities do not depend on original rewards, therefore, its convergence time doesn't change a lot after doubling deviations.

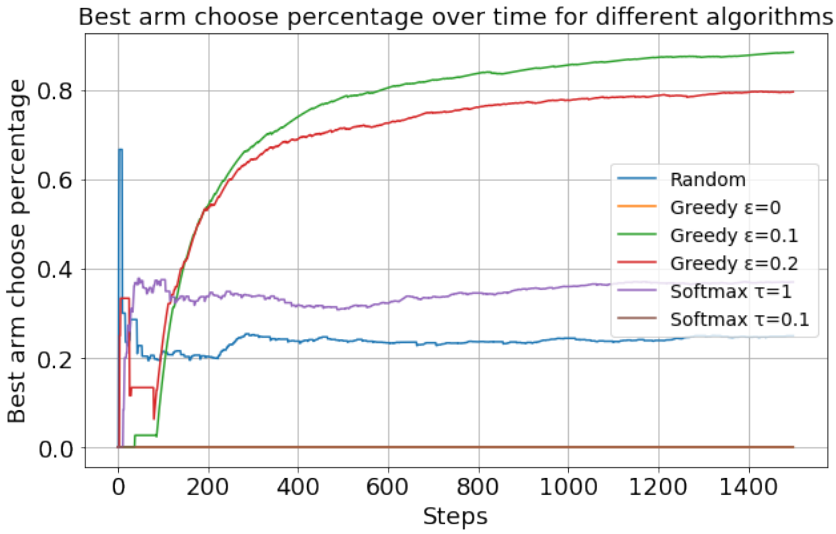

Figure 9: Best arm choose percentage over time, double std

1.3 Time-varying algorithms

For Greedy, parameter

$$\epsilon = \frac{1}{\sqrt{t}}$$

which means, at the beginning, ϵ is big, algorithm explores more, when t=1, $\epsilon = 1$, totally explore. But ϵ decreases along with time augments, then it exploits more. This time-varying greedy can combine exploit and explore better than fixed parameter greedy. Because at the beginning, it is reasonable to explore more, in order to obtain a better estimation of average rewards of arms.

For Softmax algorithm, parameter

$$\tau = 4 * \frac{1000 - t}{1000}$$

we can regard τ as a temperature, at the beginning, it is high, it explores more, and along with time, it cools down, exploit more. Its reasonability is the same as time-varying greedy.

1.3.1 Average reward for each algorithm

Average reward of each algorithm box plot is in Figure 10.

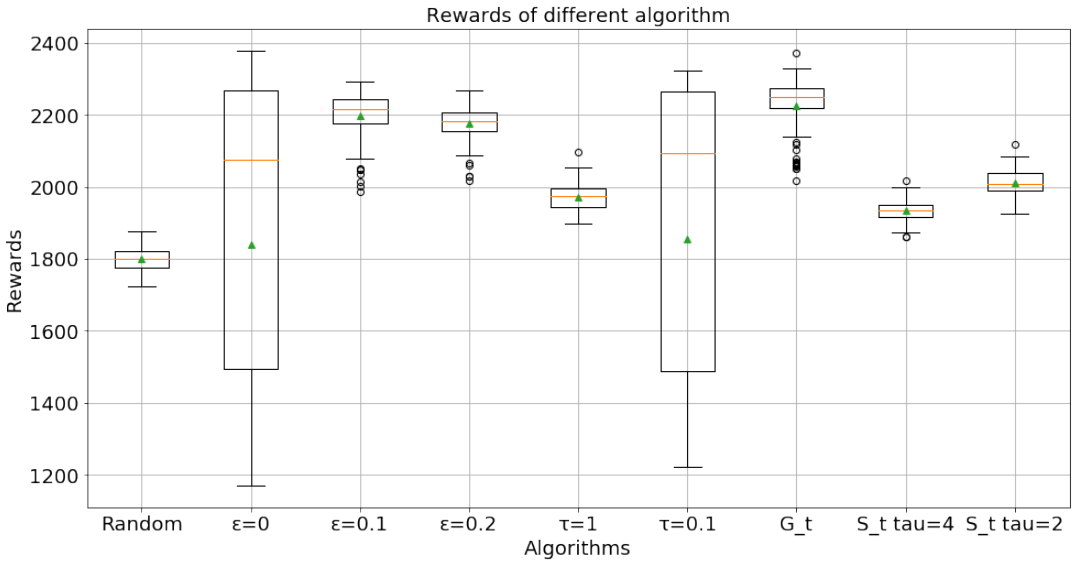

Figure 10: Box-plot of Average reward for each algorithm, with Time-varying algorithms

1. For Greedy algorithm, average reward and deviation of time-varying parameter is better than fixed parameters. This is because for fixed Greedy algorithms, its exploitation and exploration is always the same, but for time-varying Greedy, it explores more at the beginning, which benefits finding the best arm, and exploits more later, which makes player choosing best arm more with higher probability.
2. For Softmax time-varying algorithm, initial $\tau = 4$, its average reward is a little bit lower than fix $\tau = 1$, this is because at the beginning, τ is 4, higher exploration than $\tau = 1$, which makes it chose arm 2, 3, 4 more times at the beginning, this is also illustrated in Figure 11, subplot Softmax-time histogram tau=4. Because arm 2, 3, 4 have lower average rereads, thus the average reward of time-varying parameter Softmax tau=4 is a little bit lower than $\tau = 1$. But, if the initial τ is lower, I made an experiment with initial $\tau = 2$ result is shown in box-plot, most right bar, its better than fixed $\tau = 1$.

1.3.2 Number of times each action is selected

Result is in Figure 11, third row, Greedy with time-varying parameter is the best among these algorithms. Softmax time-varying parameter with initial $\tau = 2$ is better than $\tau = 4$.

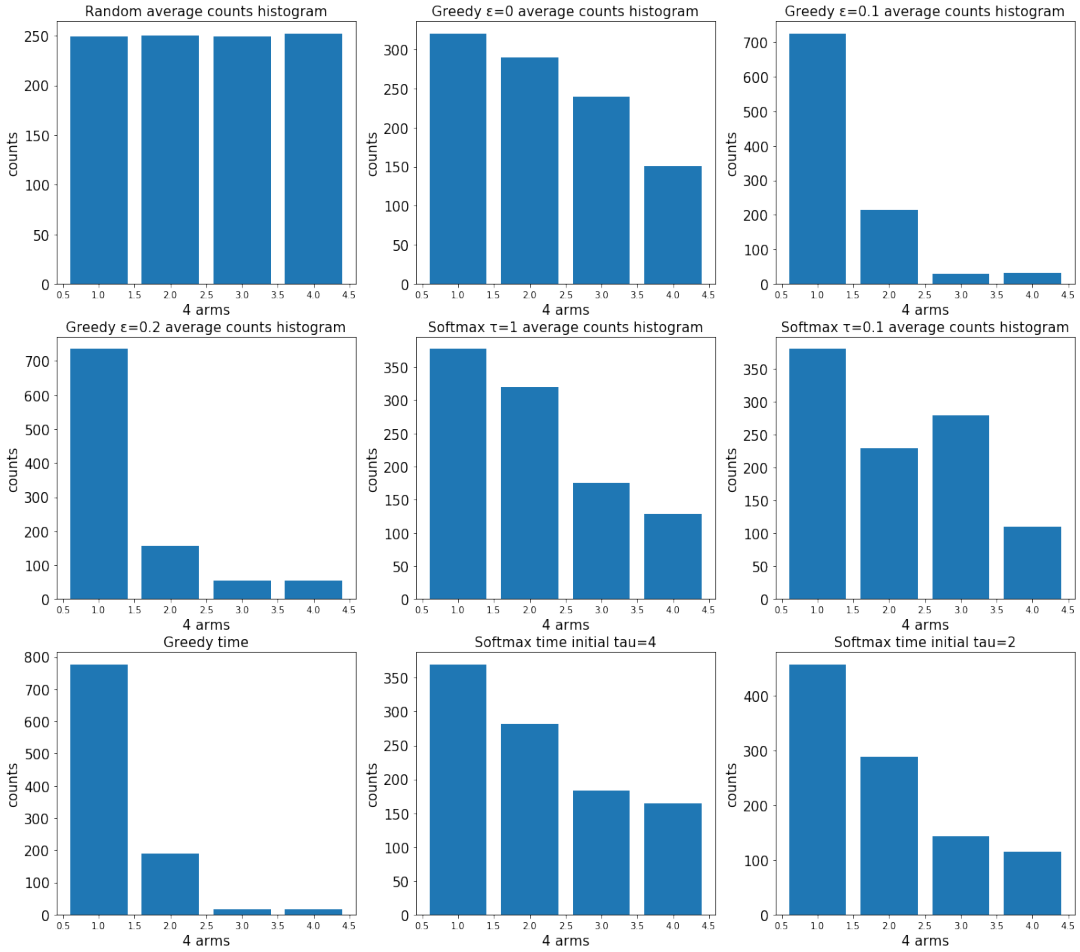

Figure 11: Number of times each arm is selected histogram, with Time-varying algorithms

1.3.3 Arms average reward estimation over time

Result is in Figure 12, time-varying Greedy can estimate average of arm 1 well, the others bad, because it chooses arm 1 mostly.

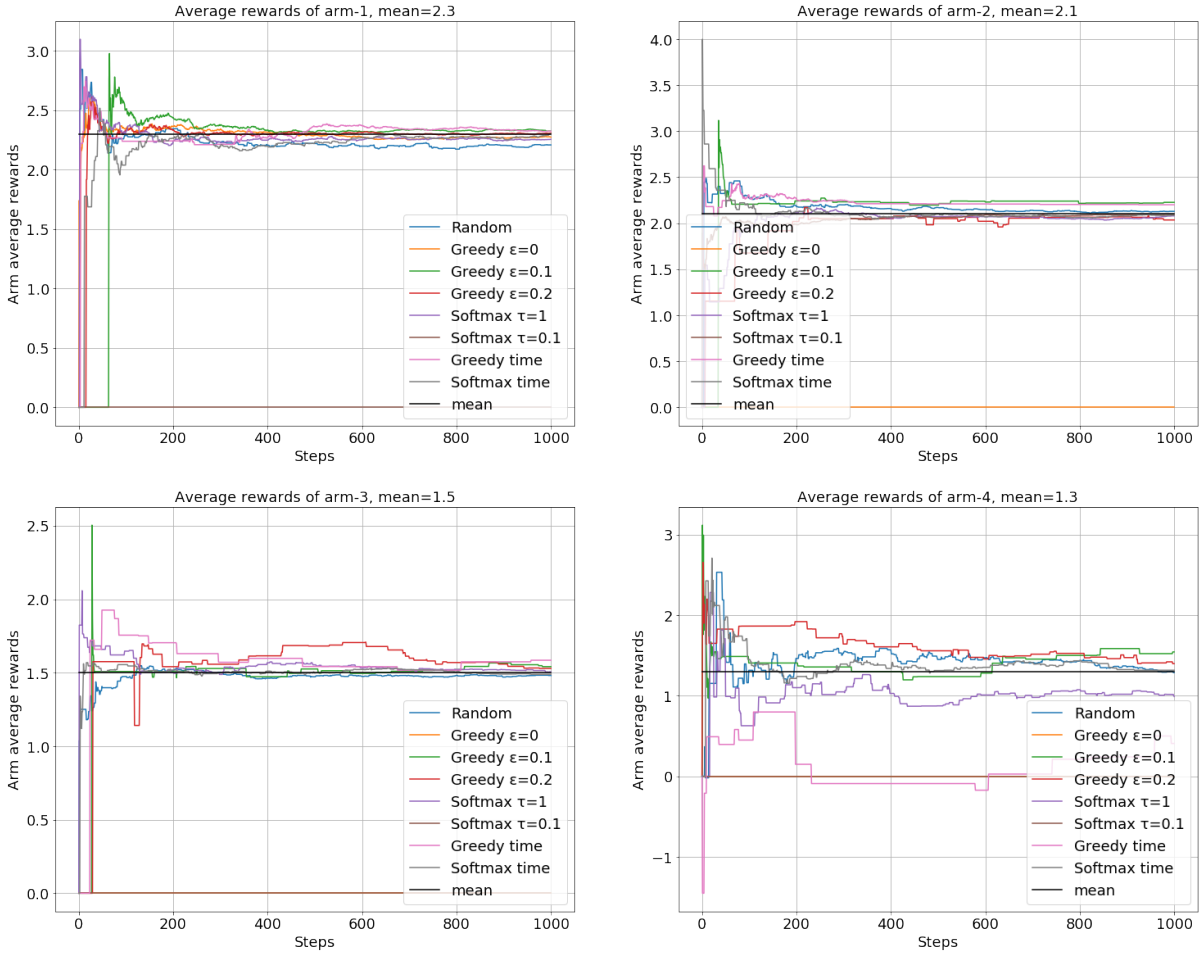

Figure 12: Arms average reward estimation over time, with Time-varying algorithms

1.3.4 Best arm

Best arm choose percentage over time is in Figure 13. Most of the time, for converge time, Softmax algorithm converges > Greedy $\epsilon = 0.1$ > Greedy $\epsilon = 0.2$ > time-varying Greedy.

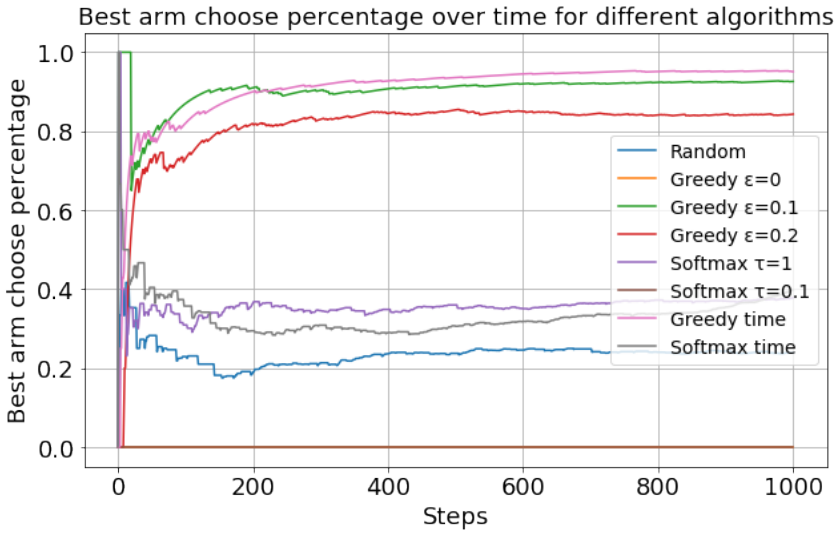

Figure 13: Best arm choose percentage over time, with Time-varying algorithms

2 Windy Gridworld

2.1 Introduction

Reinforcement learning algorithm is shown in Figure 14.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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., ϵ-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

Figure 14: Reinforcement learning algorithm

Reinforcement formula is:

$$Q(S,A) = (1-\alpha)Q(S,A) + \alpha(R + \gamma max_a(Q(S',a))) \tag{1}$$

Where, reward of old state is Q(S,A), reward of new state with action a is Q(S',a). $max_a(Q(S',a))$ is an estimate of optimal future reward.

Parameters of learning process are:

1. Policy is Greedy, parameter ϵ, which means for each step of one episode, action choice depending on ϵ, with probability ϵ choose one action randomly, else, choose the best action.
2. Reinforcement learning parameter α is the learning rate, which is to determine how many much reward is learned and how much old reward value is kept (old value is Q(S,A)). If $\alpha = 0$, means agent learn nothing (exclusively exploiting old knowledge), while $\alpha = 1$ makes the agent consider only the most recent information (ignoring prior knowledge to explore possibilities)
3. γ is the discount factor, determines the importance of future rewards. If $\gamma = 0$, which means Q(S',a) is never considered, it is 'short-sighted', only considering current rewards R.

2.2 Grid of the Windy gridworld

2.2.1 Gridworld plot

Figure 15 is the result, with $\epsilon = 0.2$, $\alpha = 0.1$, $\gamma = 0.9$. Yellow arrows are the grid of the Windy gridworld, purple arrows the form path from start to goal with the Greedy action selection (no exploration).

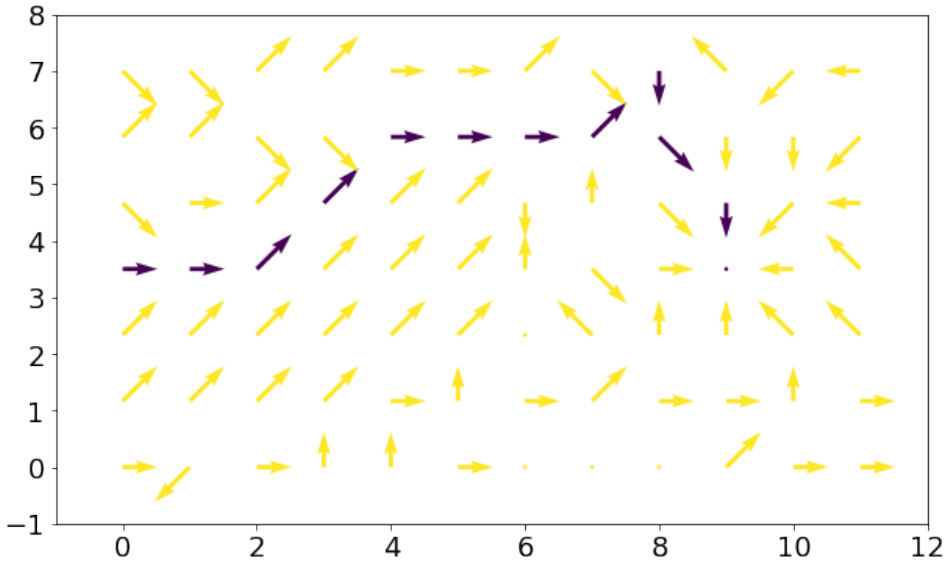

Figure 15: Path of gridworld

We can see, because of the wind in column in middle left area, the yellow arrows are blew upper wards. And there is no arrows on some positions, which means these positions are never be reached during learning process. Goal position can never be learned because each episode stops when reaches goal.

2.2.2 Reward and Steps per episode versus episode

Figure 16 is 'Total collected reward per episode versus the episode number' and 'Number of steps to reach the goal per episode versus the episode number', with original parameters.

1. Value of reward and steps is opposite, reason is evident, for each step, if next state is not goal, total reward minus 1, and steps number plus 1.
2. The converge time of $\epsilon = 0.2$, $\alpha = 0.1$, $\gamma = 0.9$ is about 500 episodes. Which means, After grid Gridworld is stable, the action choice of each state is fixed, the number of steps of each episode still slightly changes. This is because epsilon is not zero, at each state, even though the best action is fixed, it still has 0.2 possibility of choosing another action randomly.
3. Moreover, In order to observe the influence of learning rate and discount factor to converge time, I also tried two other parameters, which are: learning rate $\alpha = 0.4$ rather 0.1; discount factor $\gamma = 0.4$ rather 0.9. Results are shown in Figure 17, and 18.
4. For $\alpha = 0.4$, learning rate is higher than 0.1, therefore, it converges faster than 0.1. For $\gamma = 0.4$, it learns less from optimal future reward. Therefore, it converges slower than 0.9.

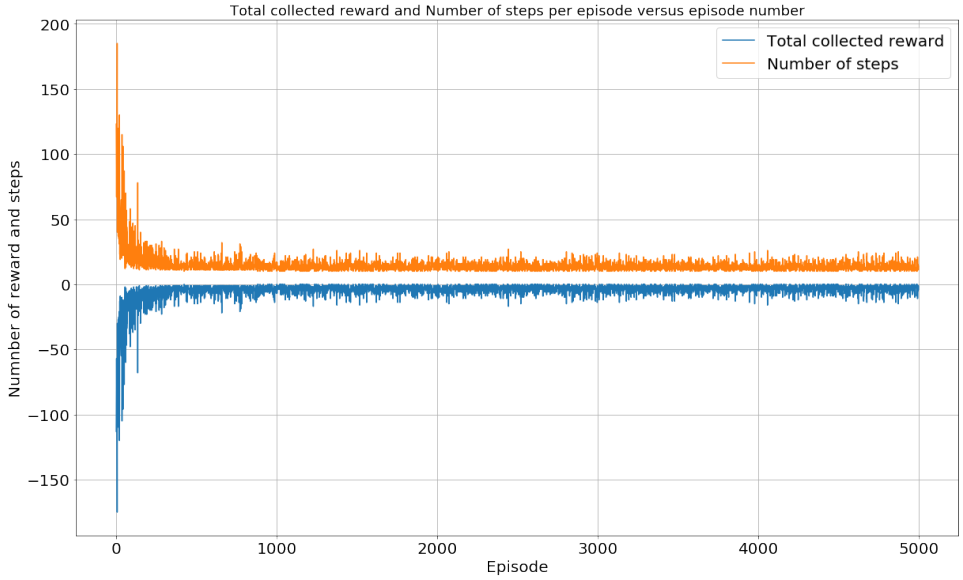

Figure 16: Reward and Steps over episode

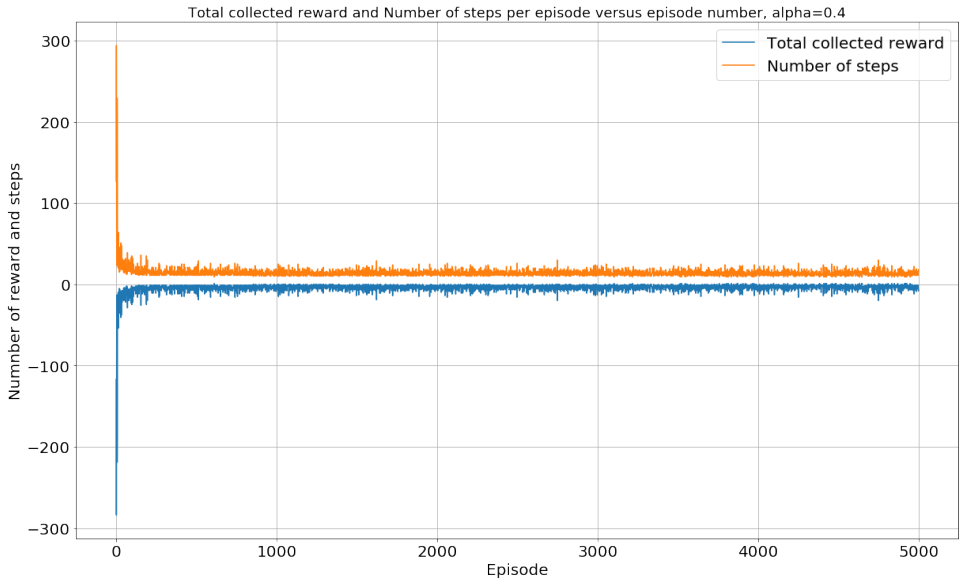

Figure 17: Reward and Steps over episode, alpha=0.4

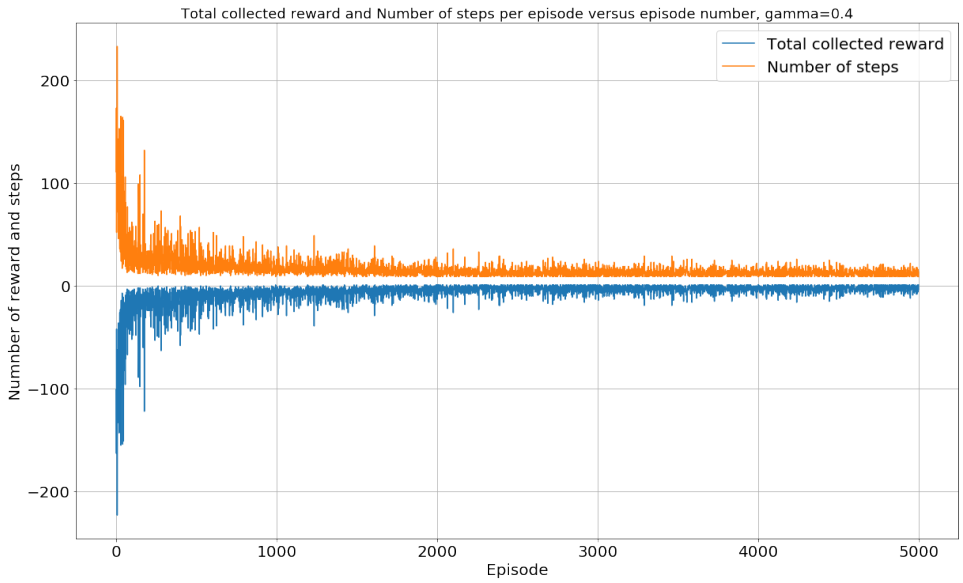

Figure 18: Reward and Steps over episode, gamma=0.4

2.3 Discuss $\epsilon = 0$ and $\gamma = 1$ situations

Figure 19 is the result with parameters $\epsilon = 0$, $\alpha = 0.1$, $\gamma = 0.9$. We can see, after gridworld is stable, approximately after 450 episodes, the steps of each episode is the same, because with $\epsilon = 0$, it only chooses the best action at each position state, therefore, its path is the same. Path is shown in Figure 20.

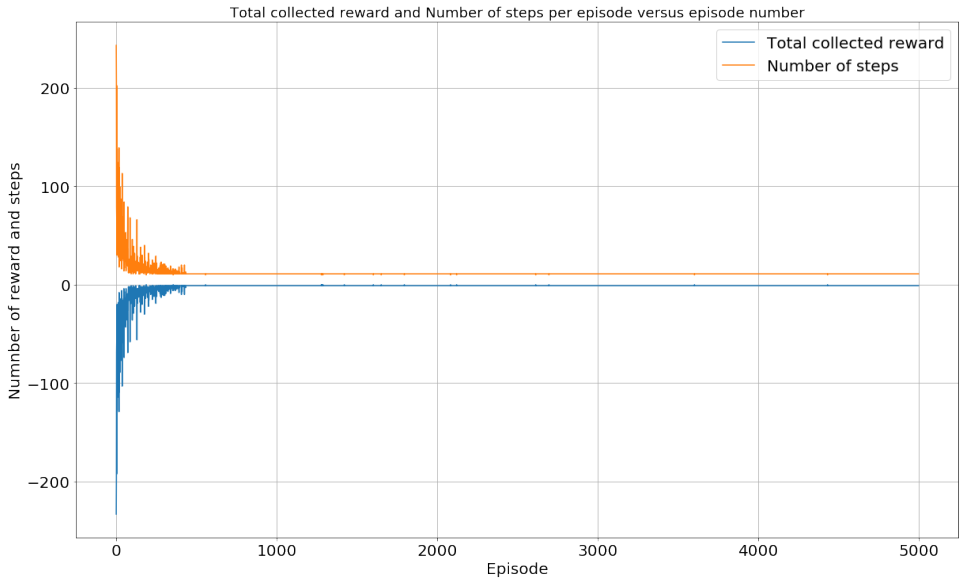

Figure 19: Reward and Steps over episode, epsilon=0

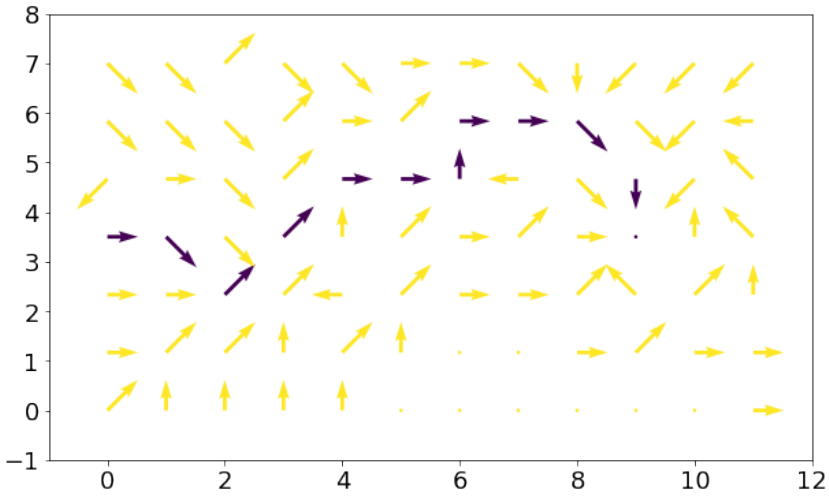

Figure 20: Path, epsilon=0

Figure 21 is the result with parameters $\epsilon = 0.2$, $\alpha = 0.1$, $\gamma = 1$.

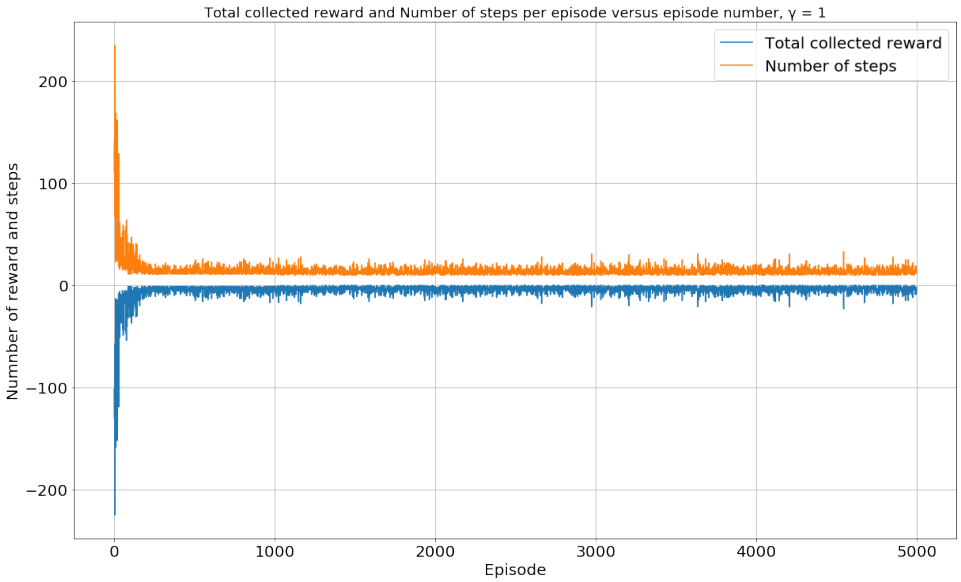

Figure 21: Reward and Steps over episode, gamma=1

When $\gamma = 1$, according to equation 1, which means the values of future reward are JUST AS MUCH as current reward. This means, for example, in ten actions, if an agent choose a good action, this is JUST AS VALUABLE as doing this action directly. The reward estimation becomes long and instable.

In order to check its instability, I change $\epsilon = 0$. If the gridworld is stable, the steps and reward of each episode should be the same after one time. But as shown in Figure 22, steps and rewards always changes, Therefore, $\gamma = 1$ is not a good choice.

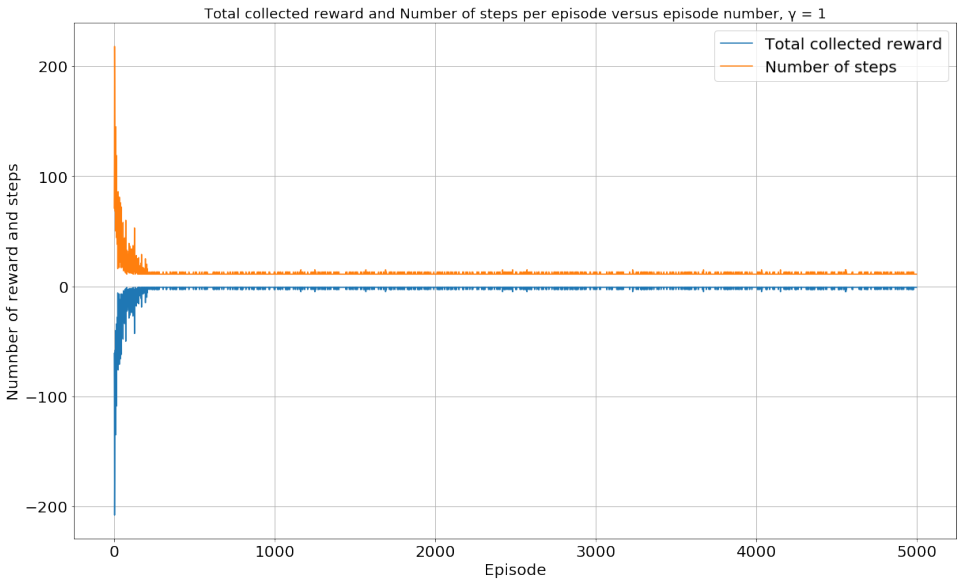

Figure 22: Reward and Steps over episode, gamma=1, epsilon=0

3 Graphical Coordination Game

3.1 Algorithm

Without loss of generality, I use a list of 26 number as my 26 agents, number is 0, 1, 2, ..., 25, the left neighbor of 0 is 25, the right neighbor of number 25 is 0.

Payoff matrices: Payoff matrix is shown in Figure 23, if actions chosen of two agents are the same, their rewards are both 1, else reward is 0.

	Action 0	Action 1
Action 0	1	0
Action 1	0	1

	Action 0	Action 1	Action 2
Action 0	1	0	0
Action 1	0	1	0
Action 2	0	0	1

Figure 23: payoff matrices for the coordination games with 2 agents and 3 agents

Algorithm 1 Win-Stay-Lose-probabilistic-Shift algorithm

```
initial 26 agents of a ring structure, alpha;
initial converge_num=0, converge_time=0;
while converge_time < 10 do
    converge_num += 1
    randomly select player1;
    randomly choose one of its neighbor, player2;
    if player1.action != player2.action then
        r1=random;
        if r1<alpha then
            player1 chooses its other actions randomly;
        end if
    end if
    r2=random;
    if r2<alpha then
        player2 chooses its other actions randomly;
    end if
    if actions of all agents are same then
        converge_time += 1
        else: converge_time=0
    end if
end while
return converge_num;
```

3.2 Experiments for different alpha

Question: What is the role of α for the Win-Stay-Lose-probabilistic-Shift algorithm? **Answer:** α is like a learning rate of this algorithm. When $\alpha = 0$, agents never learn, which means they always keep their own actions, in this situation, never converge. When $\alpha = 1$, agents always change their actions, because the action choice initial is random, it is statistically impossible for all agents chose the same action. So when $\alpha = 1$, agents also not converge.

Because for $\alpha = 0$ and 1 agents actions never converge, I only made experiments on $\alpha = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$, each α, each actions number, do 200 trails. Figure 24 is the convergence time result, and its corresponding deviation. I use log scale on y axis.

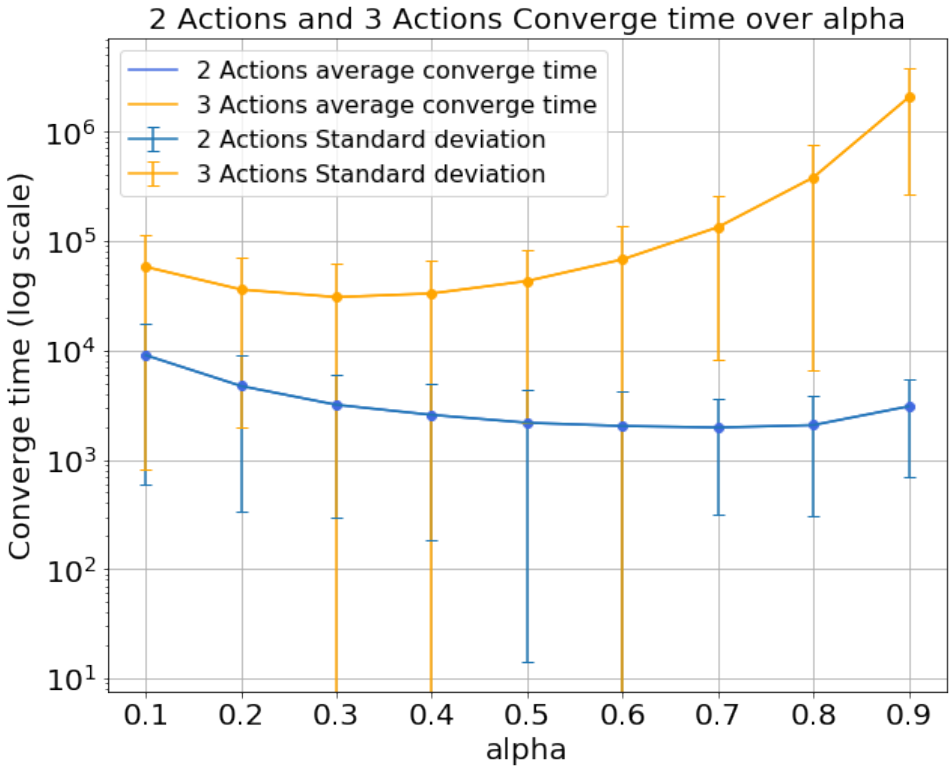

Figure 24: convergence time over different alphas

α affections on convergence time:

1. Convergence time of 3 actions game is much bigger than 2 actions convergence time. The more actions number, the more each agent's choice. So it needs more time to converge.
2. As shown in Figure 24, the convergence time line is convex. When α is very big and very small, the convergence time is very big. Because change actions too much or rarely change are both not good.
3. When $\alpha = 0$ and 1, it never converges. Explained before.
4. For 3 actions game, the fastest convergence time is reached when $\alpha = 0.3$, For 2

actions game, the fastest convergence time is reached when $\alpha = 0.7$.

5. For 3 actions game, convergence time of $\alpha = 0.9$ is much bigger than $\alpha = 0.1$. Because, for more actions game, changing actions will lead to more chaos than keeping actions. But for 2 actions game, $\alpha = 0.9$ converges faster than $\alpha = 0.1$, because each agent only has two choices, changing more could make agents more active and go to the converge state.