Reinforcement Learning

Maryam Rahbaralam

June 20, 2019

1 Question 1:

For simplicity, the expression for $\pi_\theta(a|x)$:

$$\pi_\theta(a|x) = \frac{\exp\left(\theta^\top \phi(x,a)\right)}{\sum_b \exp\left(\theta^\top \phi\left(x,b\right)\right)} = \frac{A}{B}$$

Also, to compute $\nabla_\theta \log \pi_\theta(a|x)$, so I'll go ahead and do that. We differentiate $\log \pi_\theta(a|x)$ with respect to the parameters, θ:

$$\nabla_\theta \log \pi_\theta(a|x) = \nabla_\theta \log \frac{A}{B} = \nabla_\theta \log A - \nabla_\theta \log B$$

$$= \nabla_\theta \left[\theta^\top \phi(x,a)\right] - \nabla_\theta \left[\log \sum_b \exp\left(\theta^\top \phi\left(x,b\right)\right)\right]$$

$$= \phi(x,a) - \frac{1}{\sum_b} \exp\left(\theta^\top \phi\left(x,b\right)\right) \sum_b \exp\left(\theta^\top \phi(x,b)\right) \phi(x,b)$$

$$= \phi(x,a) - \sum_b \frac{\exp\left(\theta^\top \phi(x,b)\right)}{\sum_{a'} \exp\left(\theta^\top \phi\left(x,a'\right)\right)} \phi(x,b)$$

$$= \phi(x,a) - \sum_b \pi(x,b,\theta)\phi(x,b)$$

herefore, we can write:

$$\nabla_\theta \log \pi_\theta(a|x) = \phi(x,a) - \sum_b \pi(x,b,\theta)\phi(x,b)$$

2 Question 2:

Question 2.1: How does the choice of γ influence the behavior of the algorithm?

The discount factor γ affects how much weight it gives to future rewards in the value function. A discount factor $\gamma = 0$ will result in state/action values representing the immediate reward, while a higher discount factor $\gamma = 0.9$ will result in the values representing the cumulative discounted future reward an agent expects to receive (behaving under a given policy). The convergence is influenced by the discount factor depending on whether it's a continual task or an episodic one. In a continual one, γ must be between $[0, 1)$, whereas an episodic one it can be between $[0, 1]$.

Question 2.2:

The idea behind the Multi-armed Bandits is that we have a "bandit" which can be thought of like a slot machine with different arms (in this case, we have k=2) to pull. We call our selection of arm an action and we get a different payout or reward depending on which action we take. The goal is to maximize our reward over T many pulls (in this case, we'll try 100 times). The challenge is to determine what the best action to take is, and to figure it out quickly. The sooner we can find the best, the more we can exploit that action and rake in the rewards. This is represented by a Q -value, which denotes the expected reward we get from each action. We learn about our bandits (environment) through interacting with it and gaining experience, and with each interaction we can update our Q -values. Often times these Q -values are represented as a function, Qn(a) where Q is the expected value given by action a for time step or iteration n . The solutions that we enumerate below are just different ways to estimate Qn(a).
For this task, I want to employ Gradient algorithm. This algorithm learns a preference, which causes it to select the higher preferenced actions more frequently. The preferences are calculated using softmax as follows: let π_θ be a parametrized stochastic policy with $\pi_\theta(a|x)$ being the probability of taking action a in state x. In particular, consider the policy

$$\pi_\theta(a|x) = \frac{e^{\theta^\top \phi(x,a)}}{\sum_{b \in \mathcal{A}} e^{\theta^\top \phi(x,b)}}$$

An MDP algorithm is considered with a single state x and two actions a_1 and a_2. The two actions lead to 0/1 valued random rewards such that each action a_i yields a reward of 1 with probability p_i, with $p_1 = 1/2 - \Delta$ and $p_2 = 1/2 + \Delta$ as the sum of the probabilities is always one.
To be more precise, the policy-gradient-based learning algorithm is implemented that repeats the following steps in each round k= 0,1,...:
1) The policy π_{θ_k} is computed and action is obtained
2) For each action reward is observed
3) Estimation of g_k of the policy gradient is computed.
4) Update the parameter vector as

$$\theta_{k+1} = \theta_k + \alpha_k g_k$$

To solve the **Question 2.2:** I fix $\gamma = 0.99$ and $\Delta = 0.05$, and consider different step sizes (α is the learning rate $0 < \alpha \leq 1$):

- $\alpha_1 = c/\sqrt{k}$
- $\alpha_2 = c/k$
- $\alpha_3 = c/k^2$

and for various choices of $c = [5, 10, 20]$.

Then I plot $\pi_{\theta_k}(a_2|x)$ as a function of k with considering different step sizes. We see that in Figure 1, with $\alpha_1 = c/\sqrt{k}$ the algorithm performs better as obtaining the higher policy. It means that by choosing this α_1 the agent uses a better policy π to select actions. Moreover, the higher c we have, the higher performance we get.

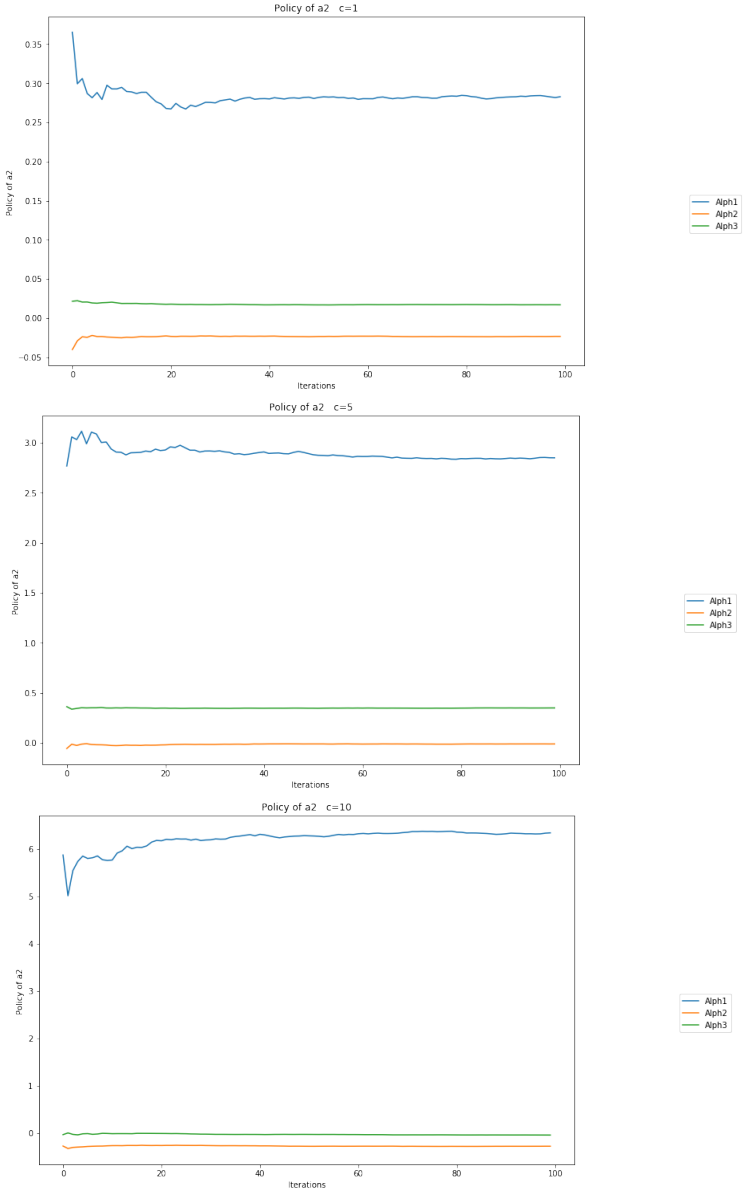

Figure 1:

Question 2.3:
Pick the best of these step sizes and plot $\pi_{\theta_k}(a_2|x)$ as a function of k for $\Delta \in \{0.01, 0.05, 0.1, 0.5\}$.

For this purpose, $\alpha_1 = c/\sqrt{k}$ is chosen and then for various choices of Δ, $\pi_{\theta_k}(a_2|x)$ as a function of k is plotted. As we can see in the figure 2 the lower Δ the better policy we have. It means that if the probabilities of the two actions have lower difference, the better policy is obtained.

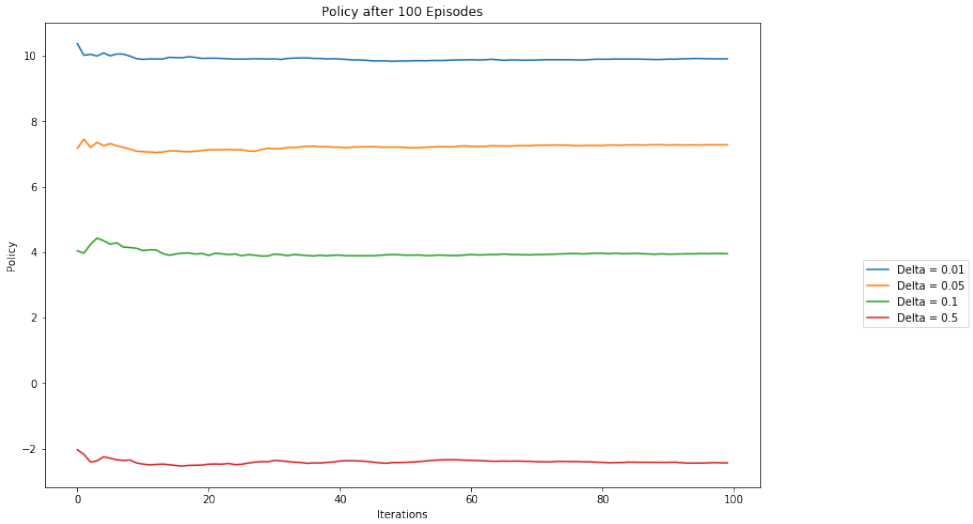

Figure 2:

3 Question 3:

Question 3.1: Upper Confidence Bound Bandit (UCB)

Implement the UCB algorithm and plot its total reward after T = 10000 rounds as a function of Δ.

For this purpose, the UCB algorithm is implemented, the idea of this UCB action selection is that the square root term is a measure of the uncertainty in the estimate of actions value. Firstly, the average UCB rewards as a function of k for different choices of $\Delta \in \{0.01, 0.05, 0.1, 0.5\}$ is plotted (see figure 3).

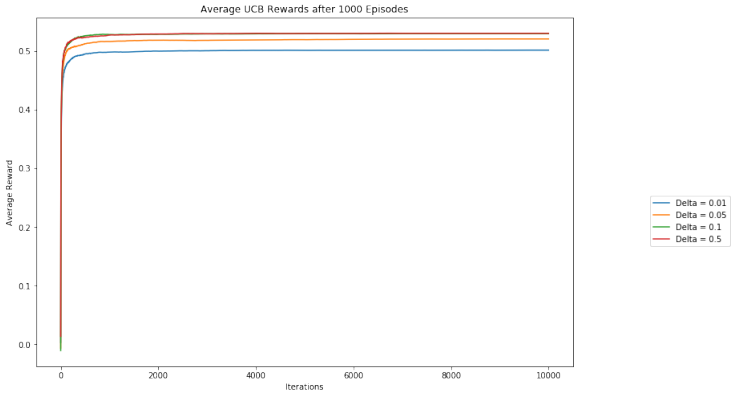

Figure 3:

Secondly, the total reward after T = 10000 rounds as a function of Δ is plotted (see figure 4). We can see that in the range of 0 to 0.1 of Δ we have upward trend as the higher Δ is,

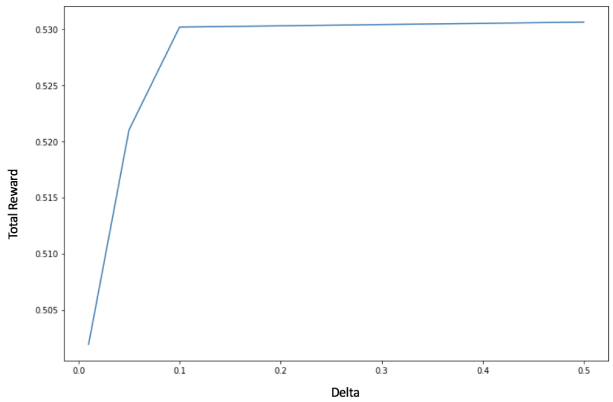

Figure 4:

the higher total reward we have. And for Δ is higher than 0.1 the total reward is remained unchanged.

Question 3.2: ε -greedy algorithm

Implement the ε -greedy algorithm with $\varepsilon_t = c/t$ for various choices of t. Plot its total reward after T = 10000 rounds as a function of Δ.

For this purpose, the ε -greedy algorithm is implemented, the idea of this ε -greedy algorithm selection is that with small probability ε the agent will pick a random action (explore) or with probability (1-ε) the agent will select an action according to the current estimate of Q-values. ε value can be decreased overtime as the agent becomes more confident with its estimate of Q-values.

Firstly, in order to see the behaviour of the ε-greedy algorithm, I set up some comparisons for different choices of $t \in \{10, 100, 1000\}$ (different values of ε). For each of these, we'll set $k = 2$, run 10,000 steps for each episode and run 1000 episodes. After each episode, we will reset the bandits and copy the averages across the different bandits to keep things consistent. Then, I plot the average ε-greedy rewards after T = 10000 rounds as a function of k for different choices of $t \in \{10, 100, 1000\}$, $c = 1$ and $\Delta = 0.5$. (see figure 5). Looking at the results, the greedy function under performs the other two consistently, with $\varepsilon = 0.01$ coming in between the two and $\varepsilon = 0.1$.

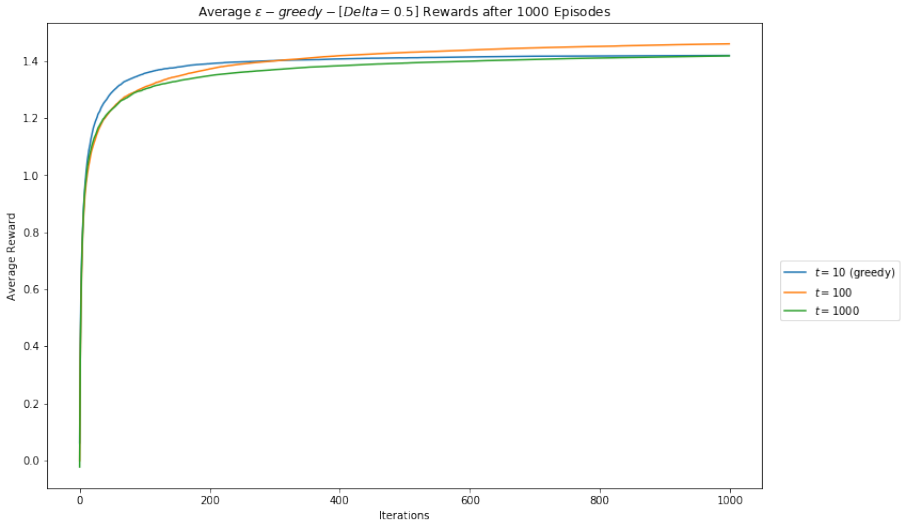

Figure 5:

Secondly, the average ε-greedy rewards after T = 10000 rounds for different $\Delta \in \{0.01, 0.05, 0.1, 0.5\}$ and $t \in \{10, 100, 1000\}$ is implemented. Then, I plot the average ε-greedy rewards after $T = 10000$ rounds as a function of Δ for different choices of $t \in \{10, 100, 1000\}$ (see figure 6). Looking at the results, the greedy function performs better with lower Δ and also have a higher total average reward with lower ε.

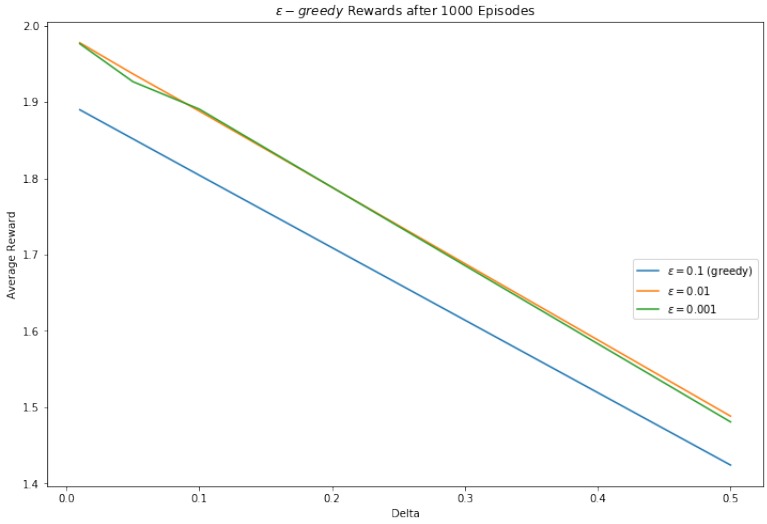

Figure 6:

Question 3.3:
Contrast the newly obtained results with those obtained for policy gradient methods. Which algorithm do you think is the best and why?

For this purpose, I implemented the two UCB and Gradient bandit algorithms after $T = 1000$ rounds and run experiments to obtain the total rewards for these two algorithms. Then, the optimal actions are counted.

We can see in the table 7 that the percentage of optimal selections for UCB algorithm is around %88, on the other hand for Gradient bandit is around %50.

```
Percentage of Optimal Selections for all episodes
UCB: 87.91
Gradient: 49.51
```

	Percentage of Optimal Selections
UCB	87.9115
Gradient	49.5104

Figure 7:

To sum up, I would say that the UCB bandit performed comparably to the gradient

bandit, as gradient bandit doesn't perform so well over the entire 1000 episode experiment in comparison to the UCB. The reasons why the UCB bandit performs well are that the bandits with a small number of arms and also the reward variance. However, it remains important to understand because it relates closely to one of the key concepts in machine learning: stochastic gradient ascent/descent. This makes up the backbone of numerous optimization strategies as the algorithm adjusts weights in the direction of minimum or maximum gradient (depending on what is being optimized for).