Multi-armed Bandits Homework ¶

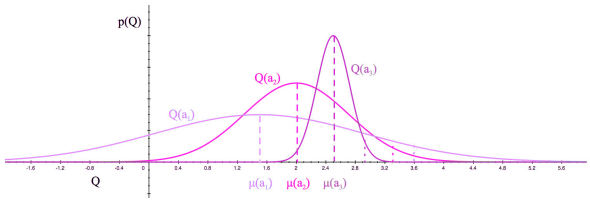
Sarmad Zandi Goharrizi - 810199181

Question 1

Using AMAlearn package and inheriting from AgentBase, we made our own 10-Armed Bandit agent.
To implement Thompson Sampeling policy, we needed different attributs and methods.The 4 most important attributs are $Pvalue$, $Qvalue$, α (a.k.a learning rate) and π_R.
Thompson Sampling uses a very elegant principle: to choose an action according to the probability of it being optimal.
In practice, this makes for a very simple algorithm. We take one sample from each posterior, and choose the one with the highest value.
Despite its simplicity, Thompson Sampling achieves state-of-the-art results, greatly outperforming the other algorithms. The logic is that it promotes efficient exploration: it explores more bandits that are promising, and quickly discards bad actions.

Note: In Thompson sampling, it is natural to assume that $Q(a)$ follows a Beta distribution.The value of $Beta(\alpha, \beta)$ is within the interval $[0, 1]$; α and β correspond to the counts when we succeeded or failed to get a reward respectively. But in this question, we have no *Prior Knowledge*, so, for the sake of generality we used a Normal distribution for $Q(a)$.

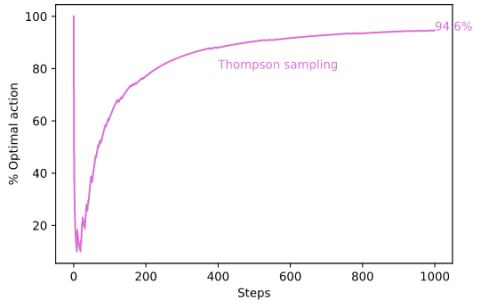
In the first trial, arm distributions have large variances; Each time we select an arm, our observations set our initial beliefs straight resulting in reaching selected arms's true variance. The more each arm is selected, the smaller the variance.
Although the agent I designed has no variance but has π, which as we know, is equal to $\frac{1}{\sigma^2}$.

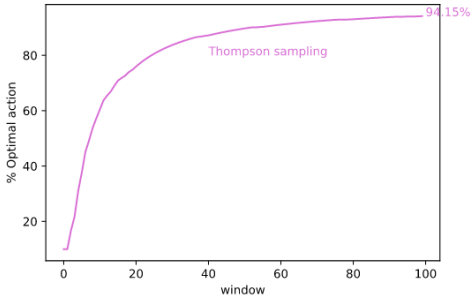


The darker distribution have been sampled more and its variance is converging to its true value.
The rules for updating are as follows:

$$Q(a) \leftarrow Q(a) + \alpha(r - Q(a))$$
$$\pi_R \leftarrow \pi_R + \pi_\varepsilon$$

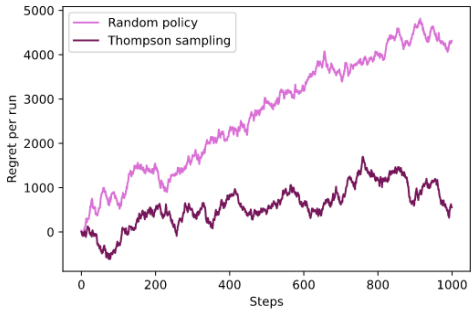
Rewards for actions are set as mentioned in *Sutton- Barto*: $N(0, 1)$
Below you can see Optimal action selection rate (in percentage) for *Thompson Sampeling* method(sliding window is applied to the figure in the right side):





As we can see, we choose the optimum action at about 94% of the times which is higher than ϵ-greedy method with $\varepsilon = 0.1$, $\varepsilon = 0.01$ or $\varepsilon = 0.1$ (greedy policy) implemented in *Sutton-Barto* **Figure 2-2**.
In **Figure 2-3** we can see an attempt to improve ε-greedy method using optimistic initial action-value estimate, but still, Thompson Sampling method is better , but not by far.
Thompson Sampling perfoms even better than Gradient method with $\alpha = 0.1$ with a baseline and optimistic initial action-value estimate as can be seen in **Figure 2-5**
So we can say that Thompson Sampling method has a better perfomance than ε-greedy method.

In this picture we can see the regret of our method. It is clear that we are performing pretty well comparing to *random policy* which is our goal .



Question 2

Part1. Developing idea

Delay is defined as the arrival time at the Enghelab sq. (6:55) until the time getting on a taxi or bus. I have to be at the university at exactly 7:30 to be right on time for my class. It takes 25 minutes to go from Enghelab sq. to university, meaning that the accepted delay is only 10 mintes.
I prefer taking the bus because it is cheaper. Also, it's good to mention that I prefer spending money than being late.

In order to pick the policy, several matters come to mind. ε-greedy can be inefficient because it selects the arms randomly. The second policy that comes to mind is UCB, a policy which achieves expected logarithmic regret uniformly over time,for all reward distributions, with no prior knowledge of the reward distribution required. In this problem, we are going to use UCB2 policy. UCB2 constructs an optimistic estimate in the form of an Upper Confidence Bound to create an estimate of the expected payoff of each action, and picks the action with the highest estimate. If the guess is wrong, the optimistic guess quickly decreases, untill another action has the higher estimate. UCB2 is an attempt to improve UCB1. This improvement is actually about reducing regret with a trade-off of a more complicated algorithm. Same as UCB1 this algorithm uses a new definition for utility function. Utility function is made up of 2 parts. One of them is the expected value of rewards, which is shown by Q and the other one is called bonus.

- Policy: (UCB2)
  - rewards:

    if delay time < 10 + no money spent $\rightarrow$ +100$\times$delay time
    if delay time< 10 + money spent$\rightarrow$ $\frac{100}{\text{delaytime}+1}$
    if delay time = 10 + money spent $\rightarrow$ 100 × (delay time+1)
    if delay time = 10 + no money spent $\rightarrow$ -$\frac{100}{\text{delaytime}+1}$
    if delay time> 10 + money spent$\rightarrow$ $\frac{100}{\text{delaytime}}$
    if delay time> 10 + no money spent$\rightarrow$ -100$\times$delay time

    Several factors come to mind when choosing the reward/punishment amount. As it was mentioned above, being late is an unwanted outcome, so when delay gets bigger than 10 minutes, there is definetly a punishment for the agent. The agent must learn not to spend money for no reason, so when the delay is less than 10 minutes, spending money is a punishable act. Also the agent must learn that 10 minutes delay is the best time to spend money in order to prevent being late for class. Rewards are calculated acording to the *delay time* and *spending money or not* (transport mode)

  - updating rule:

    $$Q(a_i) \leftarrow Q(a_i) + \alpha(r - Q(a_i))\,,\, 0 < \alpha < 1$$

  - utility function:

    $$u_t(a_i) = \underbrace{Q(a_i)}_{\mathbb{E}\{\text{reward}\}} + \underbrace{\sqrt{\frac{(1+\alpha)ln(\frac{en}{\tau(r_i)})}{2\tau(r_i)}}}_{\text{bonus}}\,,\, \tau(r) = \lceil (1+\alpha)^r \rceil$$

    UCB2 begins by playing each arm once to create an initial estimate. Then, for each iterate t, arm i is selected to achieve the maximum value for the utility function. After selecting the maximizing arm, it's played exactly $\tau(r_i+1) - \tau(r_i) = \lceil (1+\alpha)^{r_i+1} - (1+\alpha)^{r_i} \rceil$ times before ending the epoch and starting a new iteration with selecting a new arm.

    As it was mentioned in the question, we have the distribution for bus arrival which is $N(6, 16)$.
    With 95% confidence we can say that the bus can arrive upto 16 minute late.($\mu - 2.5\sigma < arrival < \mu + 2.5\sigma$, negetive arrival means nothing to us so: $0 < arrival < 16$)

To summerize, we have a 17-armed Bandit problem with UCB2 policy.

Part2. Implementing idea

To implement our *n-armed bandit* model, some key attributs and methods are needed. $Qvalue$, α (a.k.a learning rate), *number of times each arms is selected*, *transport mode* and *current trial number* are key features in making our agent. We have to teach our agent when to change from taxi to bus or vise versa, our agent must know that if the *delay time* is bigger than 10, it is gonna be punished for using the bus, rewarded for taking the bus after waiting for **exactly** 10 minutes and so on.

```
print('Optimum waiting time is:' ,optimum_waiting_time_UCB()[-1])
```

```
Optimum waiting time is: (10, 1100)
```

we can see that our agent is perfectly trained and waites 10 minues and then takes a taxi.

Part3. Implementing ε-greedy

In a pure-greedy method, you take one action, $A_n = a_1$, at $n = 1$ and get a reward.this becomes the highest reward (assuming it is positive) and the agent simply repeats it for all steps. To encourage a bit of exploration, we can use ε-greedy which means that the agent explores another option with a probability of ε. This provides a bit of noise into the algorithm to ensure that the agent keeps trying other values, otherwise, it keeps on exploiting the maximum reward.
This strategies have an obvious weakness, they continue to include random noise no matter how many examples they see. It would be better for these to settle on an optimal solution and continue to exploit it. To this end, ε-decay is used which reduces the probability of exploration with every step. This works by defining ε as a function of the number of steps:

$$\varepsilon = \frac{1}{1 + \beta n}$$

In this part we are going to implement ε-greedy method, we know that this method can get stuck on local optimum because it doesn't explore enough to find the best $Qvalue$.To prevent this issue from happening we use optimistic initial action-value estimate for $Qvalues$, decaying ε with $\beta = 1$.

```
print('Optimum waiting time is:' ,optimum_waiting_time_greedy())
```

```
Optimum waiting time is: (10, 1100)
```

As it is shown, our agent learnt to wait for 10 minues before switching to taxi.

Part3. Comparing

Other than issues mentioned in the above text, the ε-greedy strategy using a hyperparameter(ε) to balance exploration and exploitation is not ideal, as it may be hard to tune. Also, the exploration is not efficient, as we explore bandits equally (in average), not considering how promising they may be.

The UCB strategy, unlike the ε-greedy, uses the uncertainty of the posterior distribution to select the appropriate bandit at each round. It supposes that a bandit can be as good as it's posterior distribution upper confidence bound. So we *favor exploration* by sampling from the distributions with high uncertainty and high tails, and *exploit* by sampling from the distribution with highest mean after we ruled out all the other upper confidence bounds.

Part4. Bonus

In Reinforment learning, we cannot have a train and test data to performe the actual k-fold cross validation, but we can set a goal, "*Maximizing the reward*" and change different parameters to reach our goal. For example in part1, UCB2 policy, has a constant that is the coefficient of the $bonus$ part of the utili, we can find it using cross validation to reach the highest possible rewrd in the shortest possible time.

Question 3

Part1. Developing idea

We can see this shortest path (least delay) problem as an *45-armed bandit* with below arms:
[[0, 1, 4, 9, 12], [0, 1, 4, 10, 12], [0, 1, 4, 11, 12], [0, 1, 5, 9, 12], [0, 1, 5, 10, 12], [0, 1, 5, 11, 12], [0, 1, 6, 9, 12], [0, 1, 6, 10, 12],
[0, 1, 6, 11, 12], [0, 1, 7, 9, 12], [0, 1, 7, 10, 12], [0, 1, 7, 11, 12], [0, 1, 8, 9, 12], [0, 1, 8, 10, 12], [0, 1, 8, 11, 12], [0, 2, 4, 9, 12],
[0, 2, 4, 10, 12], [0, 2, 4, 11, 12], [0, 2, 5, 9, 12], [0, 2, 5, 10, 12], [0, 2, 5, 11, 12], [0, 2, 6, 9, 12], [0, 2, 6, 10, 12], [0, 2, 6, 11, 12],
[0, 2, 7, 9, 12], [0, 2, 7, 10, 12], [0, 2, 7, 11, 12], [0, 2, 8, 9, 12], [0, 2, 8, 10, 12], [0, 2, 8, 11, 12], [0, 3, 4, 9, 12], [0, 3, 4, 10, 12],
[0, 3, 4, 11, 12], [0, 3, 5, 9, 12], [0, 3, 5, 10, 12], [0, 3, 5, 11, 12], [0, 3, 6, 9, 12], [0, 3, 6, 10, 12], [0, 3, 6, 11, 12], [0, 3, 7, 9, 12],
[0, 3, 7, 10, 12], [0, 3, 7, 11, 12], [0, 3, 8, 9, 12], [0, 3, 8, 10, 12], [0, 3, 8, 11, 12]]
Each path consists of several nodes and links with different delays and congestions. Delay between each 2 nodes follows the following formula :

$$delay_{i,j} = link_{i,j} + congestion_j$$

Each link has a different delay based on its color :

$$Blue \sim N(6, 2.2) \qquad Green \sim N(2, 7) \qquad Orange \sim N(6, 6, 5)$$

and each node has a different congestion :(with different propapilities adds a 30 sec. delay to our total delay).

$$Node_i \sim binomial(1, p_i)$$

In the begining of solving this question reward was defined as *-delay time*, meaning that the agent gets punished no matter what! But then to be a little fair and gentle, and to satisfy the condition mentioned in the problem that the agent must find the best path in the shortest time possible* , reward is defined as a Bias(100 sec.), minus the delay time, so the agent seeks the least delay (i.e. maximum reward).

$$reward = Bias \text{ - } delay\ time$$

* it was observed that giving a bias reduced the convergence time.

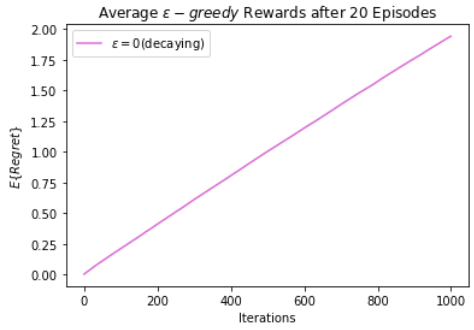
Part2. ε-greedy

To implement this method, everyting is as mentioned in question 2, part 3 except for the optimistic initial action-value estimate. It is clear that our average reward converges to an optimum value, after 100 episodes of 1000 trials, the arm selected was [0, 3, 7, 10, 12] with *55.33* reward(delay = 100 - 55.33 = 45.67) (i.e. delay time of)

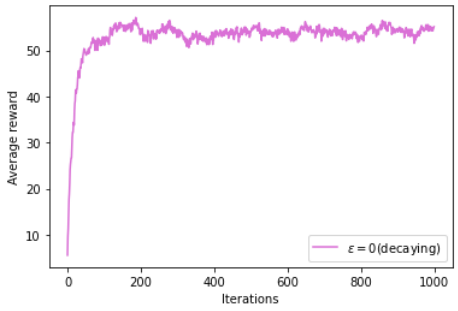
```
eps_arm, eps_reward
```

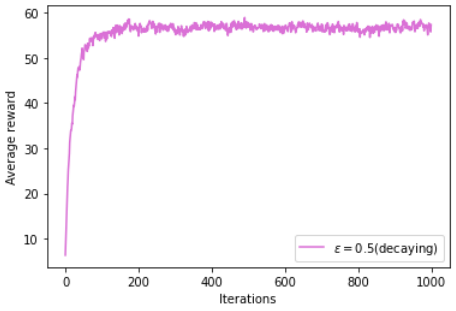
```
(10, 55.34713905212698)
```

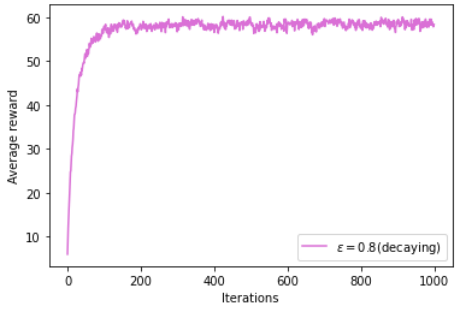
Below there are figures for different ε values. Regardless of the epsilon values, all algorithms learned the best option. ε-greedy method picks options randomly; therefore, it's not guaranteed to always pick the best option even if it found that option. That's the main reason why none of the εs achieved a probability = 1 of selecting the best option or average rewards = % rewards of the best option even after they learned the best option.
As $\uparrow \varepsilon \rightarrow \uparrow$exploration $\rightarrow \uparrow$chance of picking options randomly instead of the best option $\rightarrow \downarrow$accumulative reward $\rightarrow \downarrow$convergance rate.
Algorithms with higher epsilon learn quicker but don't use that knowledge in exploiting the best option.

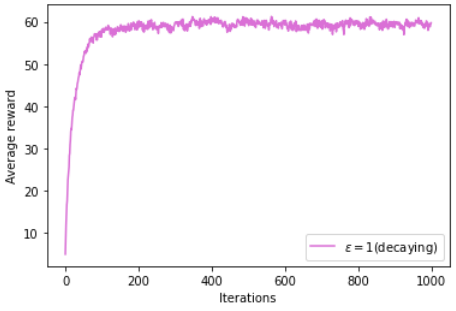


Regret for $\varepsilon = 0$ is compeletly linear beacuse we are always picking randomly. for $\varepsilon = 1$ (decaying) we eventually get a linear regret.

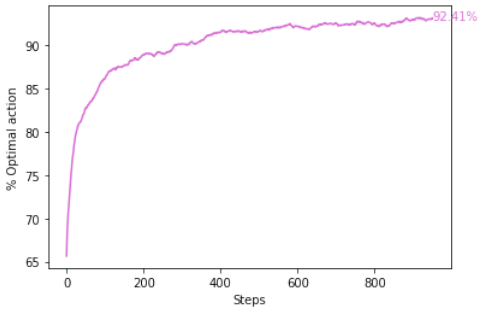








Below you can see Optimal action selection rate (in percentage) for ε-greedy method:



Note:when options are very similar (in terms of rewards), the probability of selecting the best option by all algorithms decreases.

Part3. Gradient method

Gradient algorithms take a different approach than the ones that we've discussed thus far. This is a measure of the relative value of a given action over and above the other actions that are available. The algorithm learns a preference, $H_t(a)$, which causes it to select the higher preferenced actions more frequently. The preferences are calculated using softmax.

$$Pr(A_t = a) = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(a)}} = \pi_t(a)$$

$\pi_t(a)$ is essentially the probability of taking action a at time t.
The algorithm is initialized with $H_0(a) = 0 \ \forall a$, so that initially, all actions have an equal probability of selection. In this case, the algorithm doesn't update the average of the rewards, but it updates the $H_t(a)$value for each action using stochastic gradient ascent. Each time an action is taken, a reward is returned which is weighted by the probability of the action and the learning rate. This becomes the new value for $H_t(A_t)$. Because the probabilities are all relative to one another, they are all updated in turn. The procedure can be expressed as follows:

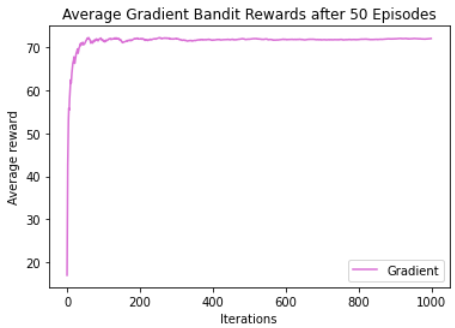
$$H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t))$$

$$H_{t+1}(a) = H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a) \quad \forall a \neq A_t$$

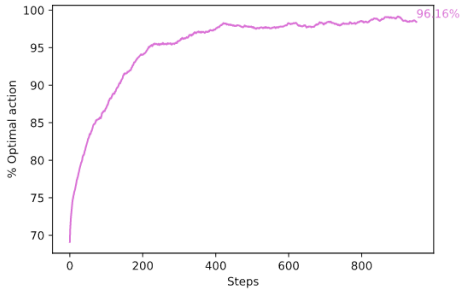
It is clear that our average reward converges to an optimum value, after 50 episodes of 1000 trials, the arm selected was [0, 2, 5, 10, 12] with reward *84.521* reward(delay = 100 - 84.521 = 15.470) (i.e. delay time of)

```
gradient_arm, gradient_reward
```

```
(19, 84.52160645361987)
```



Below you can see Optimal action selection rate (in percentage) for *Gradient bandit* method:



In conclusion *Gradient bandit* method is the superior choice; Less needed episodes for converging, higher reward, and above all better probability of choosing the best arm are it's benefits.

Part4. Compare

We need exploration because information is valuable. In terms of the exploration strategies, we can do no exploration at all, focusing on the short-term returns. Or we occasionally explore at random. Or even further, we explore and we are picky about which options to explore — actions with higher uncertainty are favored because they can provide higher information gain.

In ε-greedy method, we use random selection; The problem with random selection of actions is that after sufficient timesteps even if we know that some arm is bad, this algorithm will keep choosing that with probability ε, or $\frac{\varepsilon}{n}$ when using decayed ε-greedy. Essentially, we are exploring a bad action which does not sound very efficient. The approach to get around this could be to favour exploration of arms with a strong potential in order to get an optimal value.

The best approach to solving these kind problems (a.k.a shortest path problems) is Thompson Sampling method, although UCB2 can perform just as good.
For moderate sized graphs,trying each possible path would require a prohibitive number of samples,and algorithms that require searching through the set of all paths to reach a decision. An efficient approach therefore needs to leverage the statistical and computational structure of problem.
For these problems, we seek a general approach to exploration that accommodates flexible modeling. We know that Thompson sampling accommodates such flexible modeling, and offers an elegant and efficient approach to exploration, thereforis the best desicion.