Learning Dynamics: Assignment 3
Multi-Armed Bandits and Stochastic Reward Game

Hakim Boulahya

hboulahy@ulb.ac.be - 000391737

Université Libre de Bruxelles

December 19, 2017

Contents

1 N-Armed Bandit

Remark About the notation, the first arm/action starts at #0.

1.1 Exercice 1

1.1.1 Average rewards

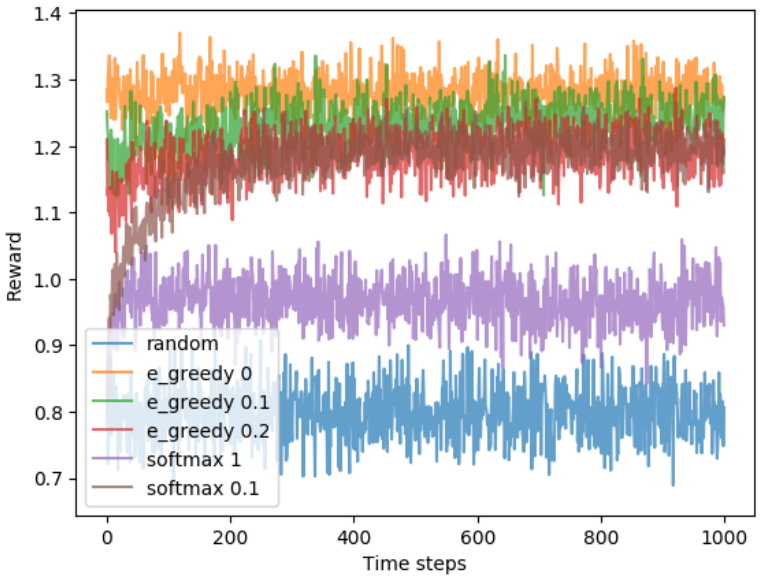


Figure 1: Average rewards for all algorithms

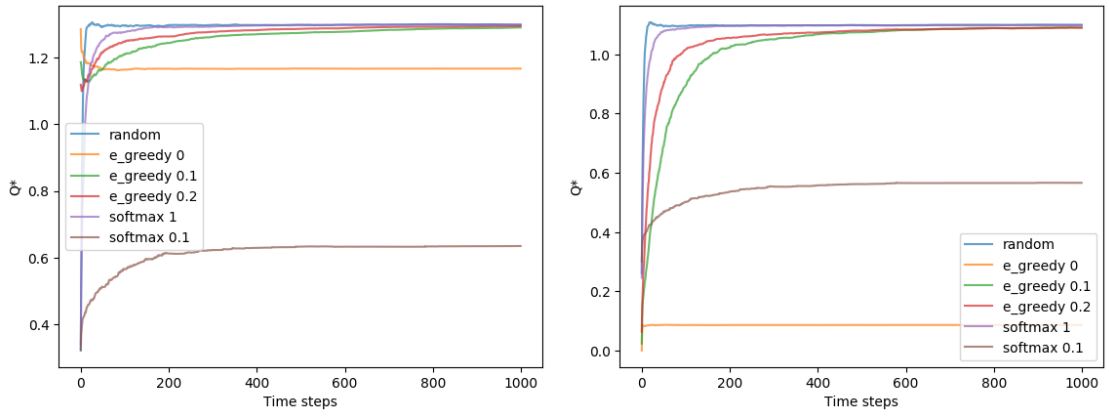
We can see that only exploring *i.e.* random and softmax with larger temperate $\tau = 1$ doesn't perform well, and the reward stays low. Using softmax with a temperature of 0.1 gives better results because of the fact that it gets more greedy.

Using ϵ-greedy with small value, i.e. a large probability to choose the optimal actions, tends to better rewards. egreedy with 0 seems to arrives *directly* to the best arm. We can see on the graph that when ϵ is bigger it usually takes more time to get to the best arm, because of the exploration. Softmax with a temperature of 0.1 does tends to the best arm, but takes more time.

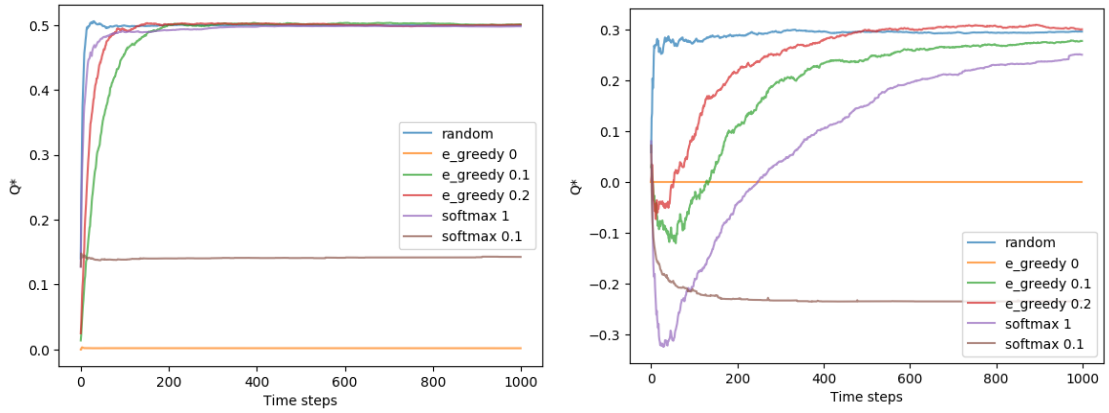
It is important to note that the greedy behaviour is better, because the standard deviation is small for this exercice. Because of that, using the best action based on the estimated Q-values is accurate.

1.1.2 Q-values estimation

Figure 2 shows the estimation of the Q-values for all algorithms.



(a) $Q_{a_0}(t)$ (b) $Q_{a_1}(t)$



(c) $Q_{a_2}(t)$ (d) $Q_{a_3}(t)$

Figure 2: Plot per arm showing the $Q^*_{a_i}$ of that action along with the actual Q_{a_i} a i estimate over time with μ = (1.3, 1.1, 0.5, 0.3), σ = (0.9, 0.6, 0.4, 2.0)

Random

Since random methods is full exploration, we can see that for all Q-values the estimation of the $Q^*_{a_i}$ are accurate.

ϵ-greedy

$\epsilon = 0$ Using a *only* greedy exploration, we can see that only the best actions have Q-values different from 0. Actually the best action #0 tends to be estimated fast enough. For action #1, the Q-values is bigger than 0 because at the beginning of the learning, the algorithm is still looking for the greedy action, therefore action #1 could be the best one, until the algorithm find that #action 0 is better. The two other actions, are never explored since the it is a full greedy method.

$\epsilon = \{0.1, 0.2\}$ The estimation is more accurate when $\epsilon > 0$ *i.e.* when the action are also selected randomly which allows exploration. We can see that by using a non-null ϵ, the estimation of

the Q-values overtime tends to be accurate. An intersting observation is that the estimation of the Q-values is faster (the accuration rate) when ϵ is larger. This is logical, since more ϵ is large, more the action are selected randomly.

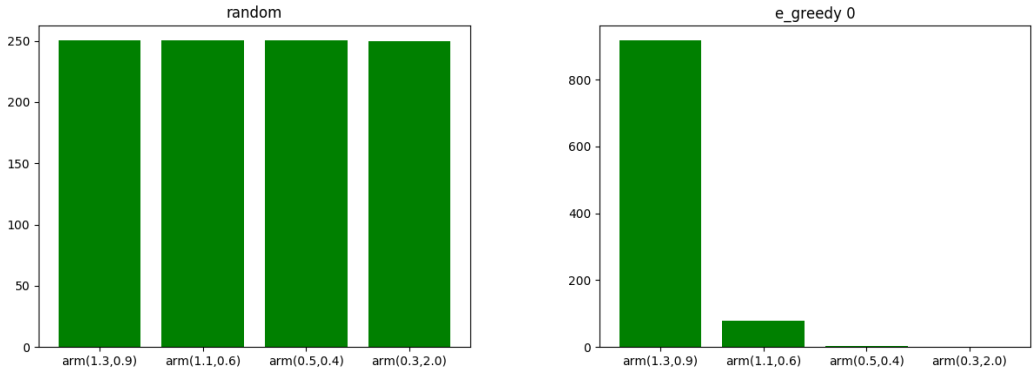
Softmax

$\tau = 1$ Using $\tau = 1$ (a large enough temperature) the Q-values estimation follows the same pattern as for any exploration methods like random or $\epsilon = 0$. We can see that except for the last action the curves are close to the random ones, which confirm the exploration nature of a large temperature. The fact that the curves of the last action #3 are more biased is because the standard deviation is larger that for the other actions, which make the Q-values estimation harder to make.

$\tau = 0.1$ Using a smaller temperature for the softmax selection method, we have a more greedy action selection. This action selection method is the worst one that estimate Q-values (excluding the full exploiting method 0-greedy). We can see that for the best arms #0 and #1, the estimations are larger than for there worst arms, but it is *far* from the actual Q^*. This can be explained by the nature of the softmax selection method with a small τ: it sticks to the best arm found and doesn't explore much to estimate the correct Q-values overtime.

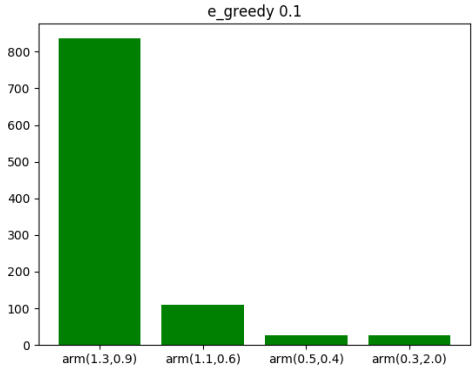
1.1.3 Histograms

Figure 3 shows the histograms for all algorithms.

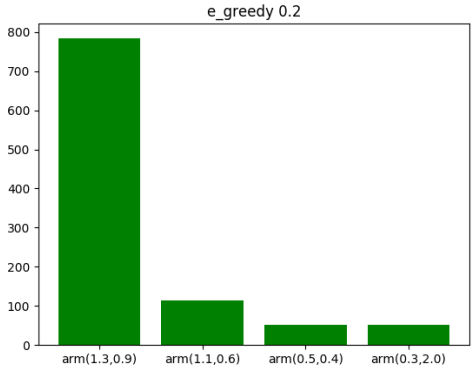


(a)

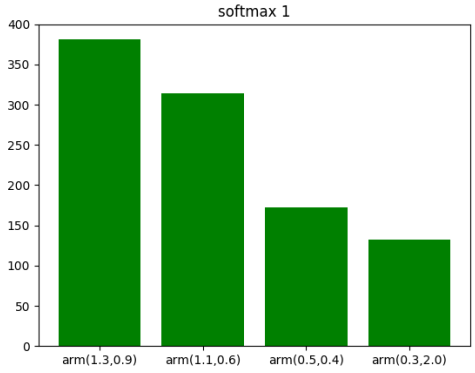
(b)



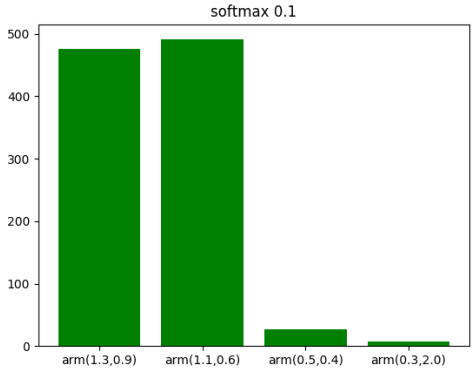
(c)



(d)



(e)



(f)

Figure 3: Histograms showing the number of times each action is selected per selection strategy with μ = (1.3, 1.1, 0.5, 0.3), σ = (0.9, 0.6, 0.4, 2.0)

Random

The action selections, shown in Figure 3a, for the random selection method is equidistributed. Since all actions have the same probablity it is logical that the selection is equidistributed when there $t \to \infty$.

ϵ-greedy

Figures 3b 3c 3d shows the histograms of action selections for ϵ-greedy selection methods.

$\epsilon = 0$ With a null ϵ, the action selection will be greedy *i.e.*, and since we have a pretty small standard deviation, the best action #0 is always chosen when found by the algorithm. This means that there is no exploration, the method chooses the best arm directly. The fact that the other actions are chosen, is that at the beginning it is necessary to explore a little bit to find the best action.

$\epsilon = \{0.1, 0.2\}$ It follows the same pattern that when ϵ is null, except that here it is possible for the methods to explore more *i.e.* choosing other actions than the best one. This is why the bar of other actions are higher when ϵ is bigger. If $\epsilon = 1$ the bars will be as shown for the random methods in Figure 3a.

Softmax

Figures 3e 3f shows the histograms of action selections for softmax selection methods.

When temperature $\tau = 1$, the action selections are more randomize *i.e.* the exploration is more noticable. An interesting observation in Figure 3f is that when $\tau = 0.1$, *i.e.* the action selection is more greedy, softmax algorithm tends to hesitate to choose the best actions. The number of times that the 2 best actions, action #0 et #1, are chosen is more or less equal.

1.1.4 Questions

Which algorithms arrive close to the best arm ? Based on the analysis that we made on the plots from this section, we observed that the algorithms that exploit are getting closer to the best arm. The *full* exploit algorithm 0-greedy is the fastest one followed by 0.1-greedy and 0.2-greedy. Using a low temperature $\tau = 0.1$ with the softmax algorithm also allow exploitation, but we can see that it is slower that the ϵ-greedy ones.

1.2 Exercice 2

1.2.1 Average rewards

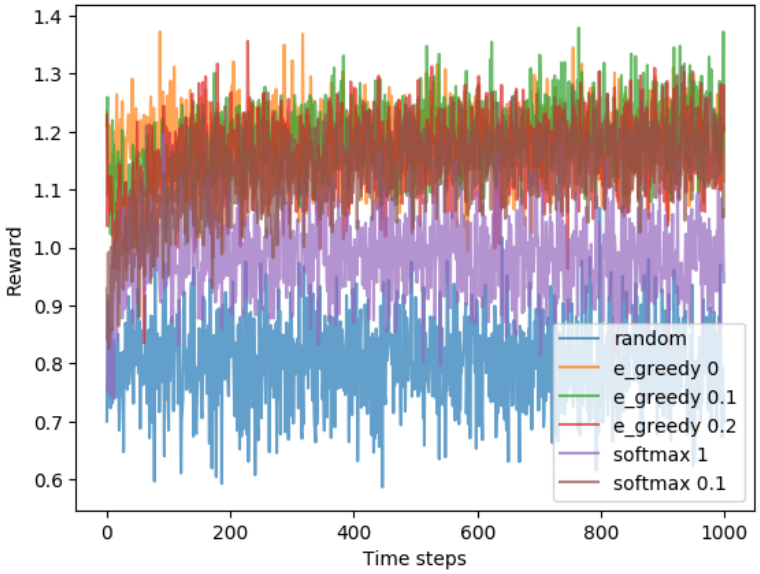


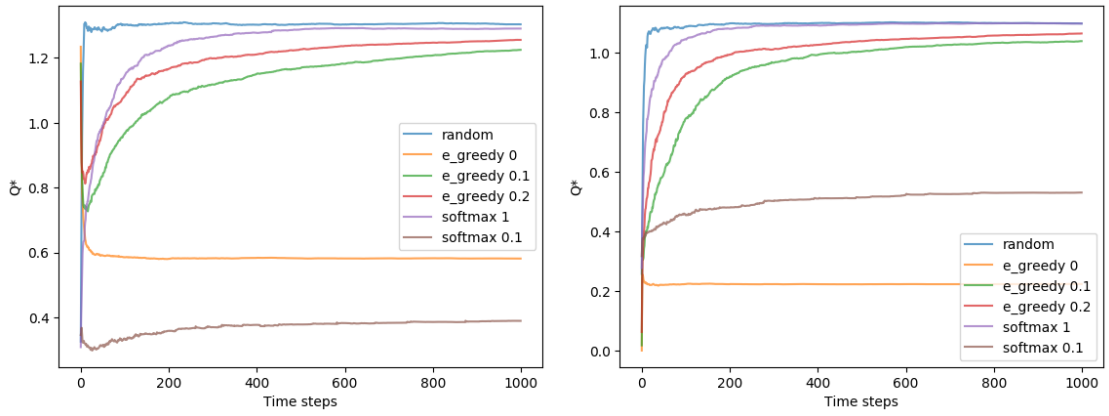
Figure 4: Average rewards for all algorithms

The results order of the algorithms that perform best doesn't really changes compared to exercice 1. By doubling the standard deviation, we only allowed the algorithms to provide a more random rewards *i.e.* a bigger range. We can see that the dynamics of the algorithms don't change.

It is harder to learn because of the standard deviation that provides a more randomize outcomes, therefore harder to find the best rewards.

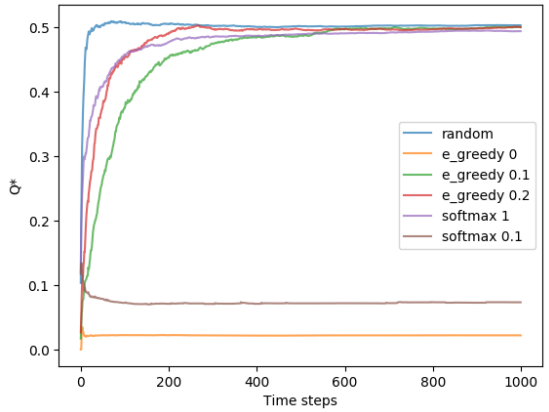
1.2.2 Q-values estimation

Figure 5 shows the estimation of the Q-values for all algorithms.

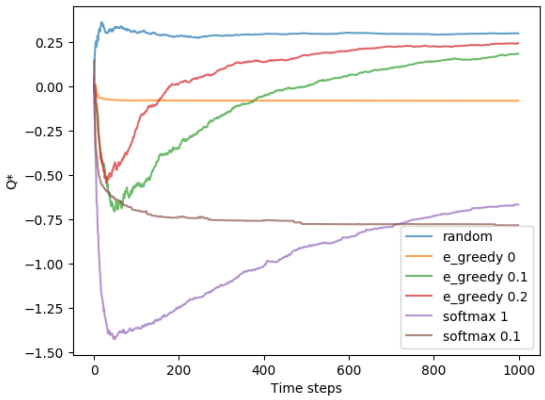


(a) $Q_{a_0}(t)$

(b) $Q_{a_1}(t)$



(c) $Q_{a_2}(t)$

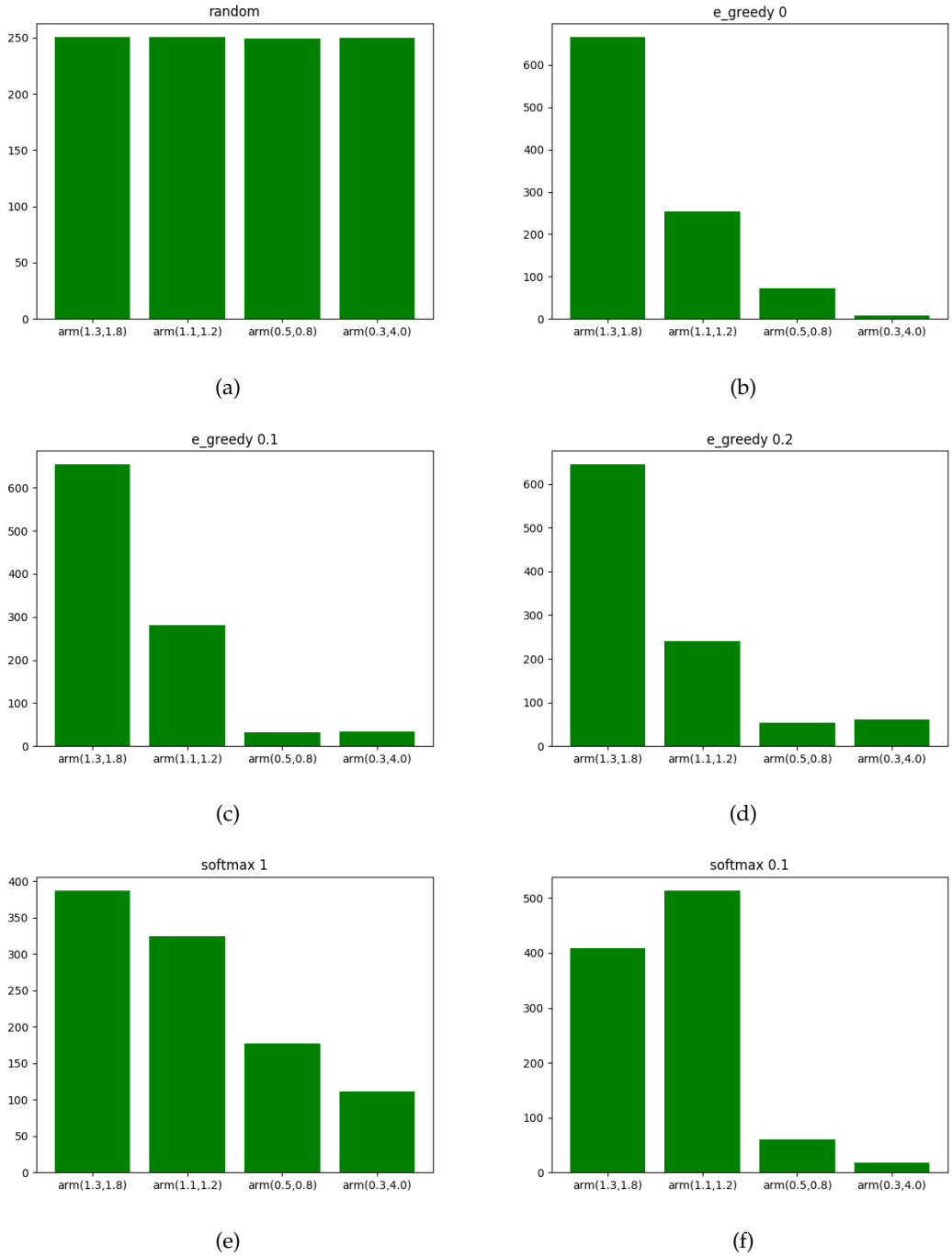


(d) $Q_{a_3}(t)$

Figure 5: Plot per arm showing the $Q^*_{a_i}$ of that action along with the actual Q_{a_i} a i estimate over time with μ = (1.3, 1.1, 0.5, 0.3), σ = (1.8, 1.2, 0.8, 4.0)

The behaviour of the estimations for all algorithms are the same that we explain in section 1.1.2. The major difference is that the Q-values estimations for all algorithms, when exploring takes more time, because the standard deviation is larger. Because of that it is harder for the learner to estimate the correct Q-values.

1.2.3 Histograms



(a) (b)

(c) (d)

(e) (f)

Figure 6: Histograms showing the number of times each action is selected per selection strategy with μ = (1.3, 1.1, 0.5, 0.3), σ = (1.8, 1.2, 0.8, 4.0)

The actions selection behaviour for all algorithms seems to follow the same pattern that we explain in section 1.1.3. The major difference is that for the one that exploit th most, for example 0-greedy, the selection of the *worst* action have a higher bars. This is because of the larger standard deviation for all actions. It is harder to determine which one is the optimal selection *i.e.* the exploration at the beginning is necessary, because the Q-values are estimated delayed compared to when the standard deviation is smaller.

1.2.4 Questions

Harder or easier that Exercice 1 ? It is harder. Due to the higher standard deviation, the Q-values are harder to estimate, therefore hard to tend to the best arm.

What about performance ? The performance changes are noticable. By that we mean that the Q-values estimation are more inaccurate because of the scale of the randomness that is larger. It takes more time to estimate correctly, and the rewards curves are less accurate *i.e.* they tends to the same range of values. As we can see in Figure 4, the reward lines interlace more and the exploit algorithms tends to the same range of reward values. The deviation is small enough to emphasize the fact that exploration algorithms (random and softmax 1) are less efficient.

1.3 Exercice 3

1.3.1 Average rewards

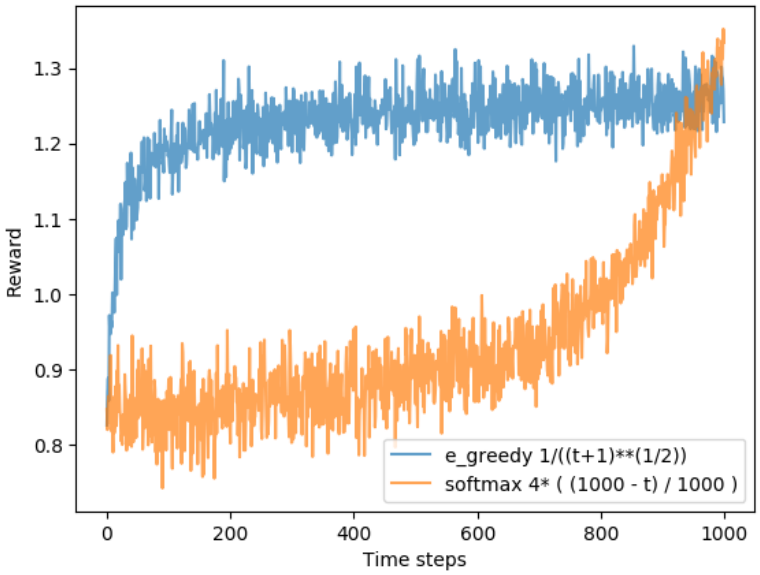


Figure 7: Average rewards for all algorithms

Dynamic ϵ-greedy The curve of the dynamic egreedy follows the same behaviour than the static ones. It grows fast to reach a steady state around the best arm. This is due to the fact that $1/\sqrt{t}$ will have a high probability to explore at the beginning, but will fall fast enough to be greedy overtime (at around epoch 100).

Dynamic softmax The curve of the dynamic softmax is different from the static ones. Using the temperature $4 * \frac{1000-t}{1000}$ makes the algorithm to choose the action randomly at the beginning and during a *long* moment. It's only at around epoch 800 that the action selection will be more greedy, therefore tends to the best arm. This is explained by the fact that the temperature at $t = 0$ (when t is small) the temperature is large e.g.:

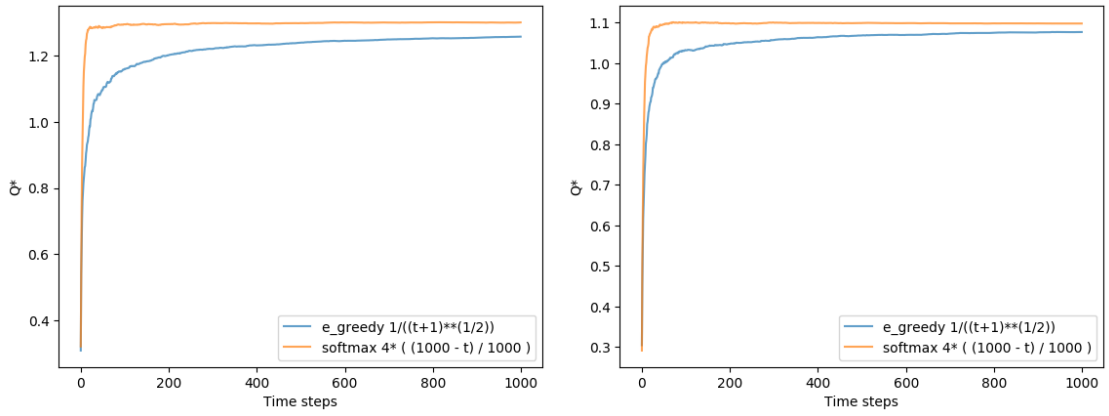
$$\tau = 4 * \frac{1000 - t}{1000} = 4$$

And when $t = 1000$ (when t is large) the temperature is low *i.e.* the action selection method is more greedy:

$$\tau = 4 * \frac{1000 - 1000}{1000} = 0$$

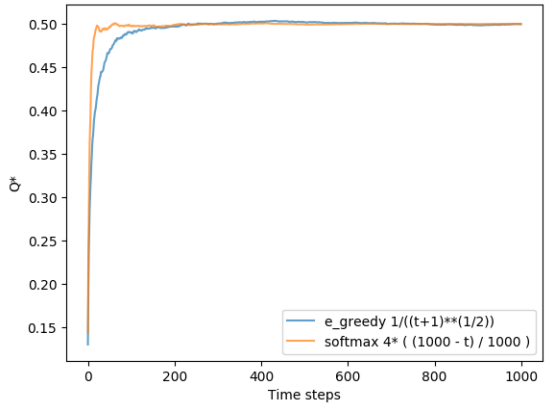
1.3.2 Q-values estimation

Figure 8 shows the Q-values estimation overtime of the dynamic selection methods ϵ-greedy and softmax.

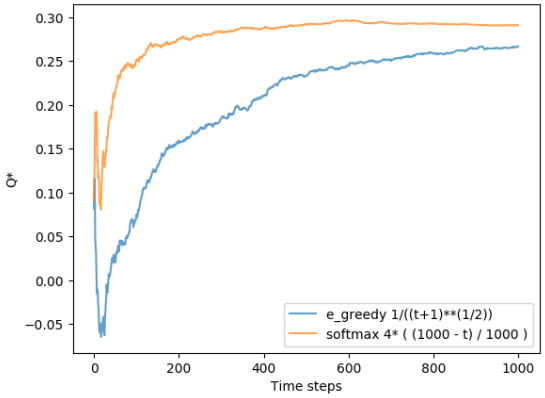


(a) $Q_{a_0}(t)$

(b) $Q_{a_1}(t)$



(c) $Q_{a_2}(t)$



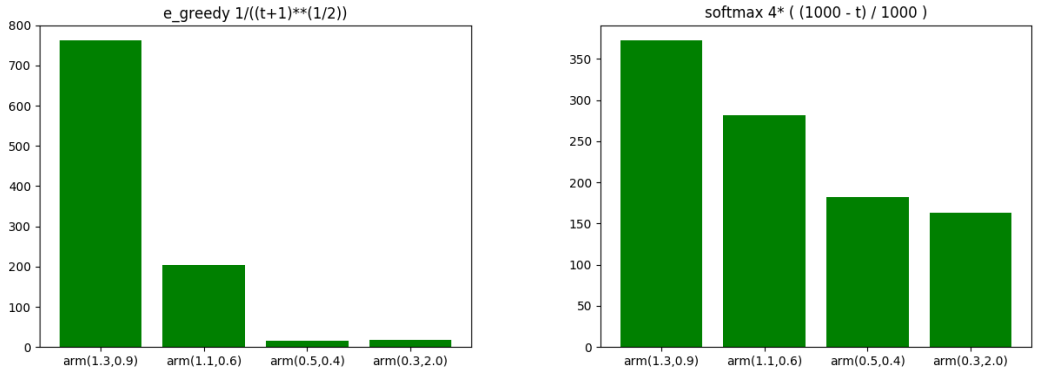
(d) $Q_{a_3}(t)$

Figure 8: Plot per arm showing the $Q^*_{a_i}$ of that action along with the actual Q_{a_i} a i estimate over time

Dynamic ϵ-greedy Since the dynamic e-greedy method is randomize at the beginning and more greedy overtime, the Q-values are correctly estimated and tends to the correct Q^* overtime.

Dynamic softmax We have a large temperature at the beginning and a smaller temperature overtime. This leads to a algorithm that explore a lot at the beginning, therefore estimate the Q-values very fast.

1.3.3 Histograms



(a) (b)

Figure 9: Histograms showing the number of times each action is selected per selection strategy

Dynamic ϵ-greedy The action selection tends to the optimal action #0 very fast. This is because that the probability ϵ drops very fast overtime, which lead to the greedy selection behaviour.

Dynamic softmax Since dynamic softmax has a large temperature when t is small, therefore explore a lot, we can see that all actions are selected. Overtime to selection will be more greedy, and since the Q-values are accurate because of the exploration, when t is high enough, the action #0 and #1 will be selected more often. That explains the *stairs* form of the histogram.

1.3.4 Questions

Fixed parameters vs dynamic parameters Dynamic ϵ-greedy doesn't seems to perform better that the static ones. The resulting rewards tends to be identical. Observing the curve in Figure 7 shows that it takes more time to stabilize to the best arm.

Dynamic softmax seems to be better, if you consider the long term reward. It takes to around epoch 1000 for the algorithm to find the best reward value in comparaison to the softmax using a a temperature of 0.1, where it is close to the best arm at around epoch 200. But not as close as the dynamic softmax, where it is closer to the best Q-value, when it is stabilized.

2 Stochastic Reward Game

2.1 Algorithms

I chose to implement the softmax action selection with a temperature $\tau = 0.1$. For the heuristic algorithm, the FMQ heuristic from [1]. The expected value function is set as follow:

$$EV(a) = Q(a) + c * maxR(a)$$

The only difference with the article is that we don't use use frequence of the max reward. This is due to the fact that the reward are stochastic, *i.e.* each reward is random and will occur only once. c is the weight of the FMQ (importance of the heuristic compared to the Q-values), and is set to 1.

The temperature of the boltzmann distribution is:

$$\tau(t) = e^{-t} * \tau_{max} + \tau_{min}$$

τ_{max} is the value that the temperature has at the first epoch, and τ_{min} is the lowest temperature that can be reach overtime.

And the variables are set as follow:

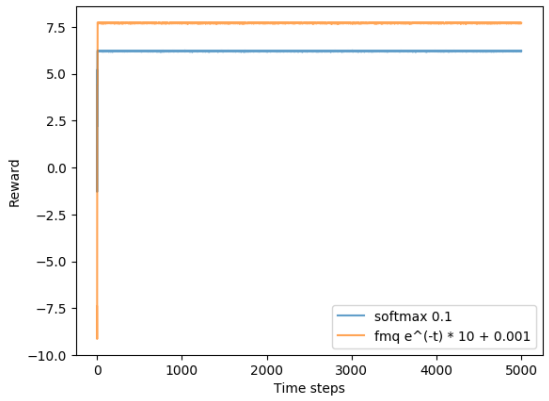
- $\tau_{max} = 10$
- $\tau_{min} = 0.001$

The FMQ heuristic consists of choosing the action based on a boltzmann distribution *i.e.* a softmax action selection method with a decreasing temperature $\tau(t)$ and a expected values that is not only the Q-values but the function $EV(a)$.

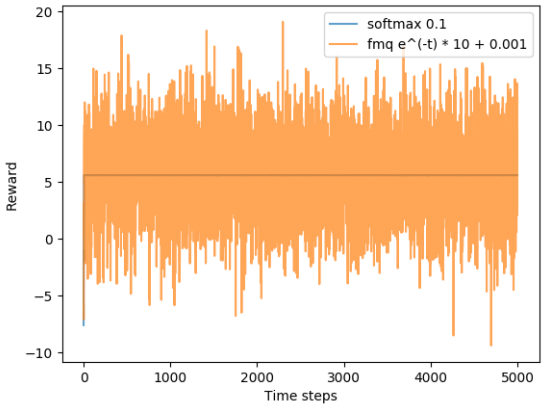
The Q-values are computed as in the N Bandits Problems, since the learners should choose their action independently. For this game the difference is with the rewards. With 3 actions per player and joint learning *i.e.* the reward is the same for both player and based on their actions, their is 9 outcomes possible in opposition to independent learning where the outcome is based only on the player actions, therefore 3 outcomes possible.

2.2 Results

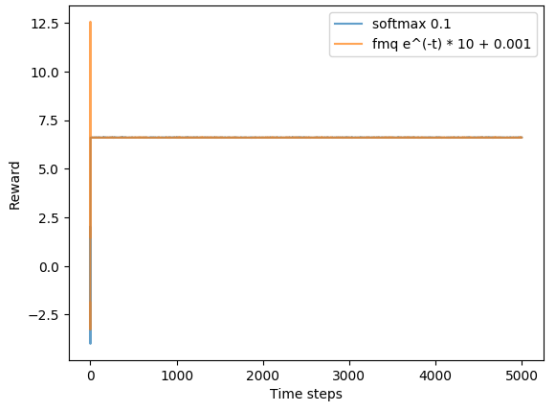
Figure 10 shows the three graphs of the total collected reward per episode for all three cases. 20 iterations of 5000 time steps per simulation were ran, and the resulting plots shows the average reward of all iterations combined.



(a) $\sigma_0 = \sigma_1 = \sigma = 0.2$



(b) $\sigma_0 = 4$ and $\sigma_1 = \sigma = 0.1$



(c) $\sigma_1 = 4$ and $\sigma_0 = \sigma = 0.1$

Figure 10: Total collected reward per episode versus the episode number for softmax 0.1 and FMQ heuristic

By settings all standard deviation to 0.2, the results plot is shown in Figure 10a. We can see that the FMQ heuristic provide a better rewards. Actually the rewards are an average of the two best joint actions $< a_1, b_1 >$ and $< a_2, b_2 >$. It seems that the algorithm stick to one best joint-actions.

By setting $\sigma_0 = 4$ *i.e.* the best joint-action $< a_1, b_1 >$ Q-values will be harder to estimate. We can see in Figure 10b that the softmax method will choose one equilibrium and stick with it. The FMQ heuristic method average rewards shows that the method explore much more during each epoch, and provide a more randomize outcomes.

By setting $\sigma_1 = 4$ *i.e.* the joint-action $< a_2, b_2 >$ Q-values wil be harder to estimate. Since the standard deviation is high for the second optimum, the algorithms sticks to the one they found.

2.3 Discussion

How will the learning process change if we make the agents independent learners? If the agents learn independently, the rewards will be based only on their own actions, and not on the joint-action. This yield to the process of using only 3 possible values instead of 9. Therefore it will be harder to estimate the Q-values and find the best action. the IL will be slower at determining the optimal equilbrium. Also if the standard deviation are larger, the probabilty of finding the best actions will be harder for IL than for JAL.

How will the learning process change if we make the agents always select the action that according to them will yield the highest reward (assuming the other agent plays the best response)? This is the action selection made by the 0-greedy method. Since we are choosing our best action when the other player chooses his best action, this provide a *full* greedy action selection. Because of that the optimum might be harder to find because no exploration is made by the agents, and this will lead to a big possiblity to miss the optimum equilibrium.

3 Project source files

The project wa implemented in Python3. It uses a combination of scripts and config files to implement the exercices.

There is two scripts that are used, with config files, to run the simulation:

- **nbandits.py**: scripts used for the first part
- **climbing.py**: scripts used for the second part

Config files for exercices of the first part are in the directory **bandits_exercices/**.
Config files for exercices of the second part are in the directory **climbing_exercices/**.
The config files are self-explanatory.

3.1 How to run

Install the dependencies (numpy and matplotlib):

```
pip install -r requirements.txt
```

Run the scripts:

```
python3 {nbandits.py, climbing.py} config.py
```

Example:

```
python3 nbandits.py bandits_exercices/ex1_table3.py
```

References

[1] Kapetanakis, S. and Kudenko, D. (2005). *Reinforcement learning of coordination in heterogeneous cooperative multi-agent systems.* Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 3394 LNAI:119–131.