Pierard
Florian
Bioinformatics Master

03/12/2015

LEARNING DYNAMICS

Assignment 3

ULB student

Learning dynamics (VUB)

Academic year: 2015-2016

1. N-armed Bandit

1.1) Exercice 1

In these plots I have run my algorithm also with the epsilon and the Tau depending on the time (cfr. Ex3). I will analyze the plots depending on the time in this exercise for an easier understanding.

For my algorithm if the first choice gives a negative reward because of the standard deviation and if the next move is exploitation, he will chose one of the other actions randomly because the other Qa will be equal to 0. So all Qa's will be higher than the negative one.

I obtain this type of graph:

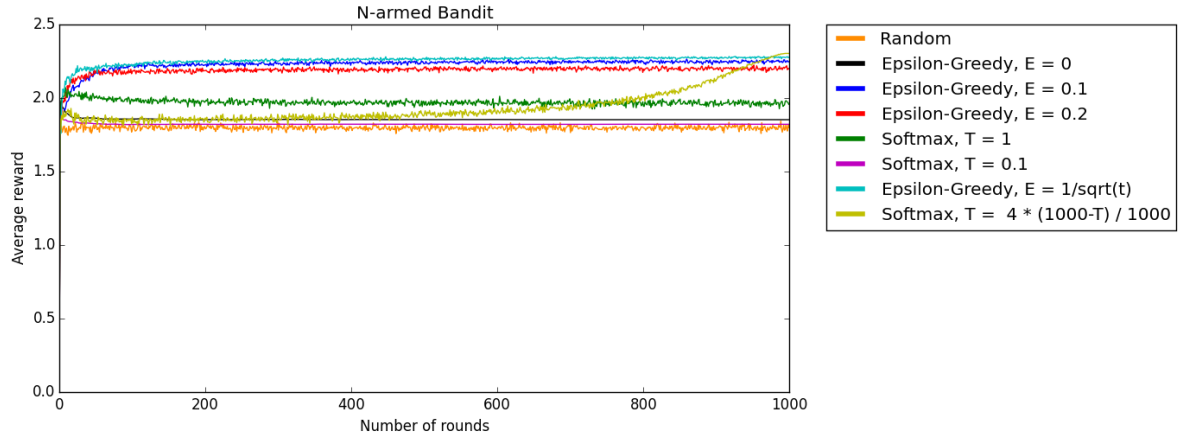


Figure (1) : Average reward for each algorithm

We can notice the quick increase at the first reward no matter the strategy used (Figure 1). Let's zoom in the range 1.7 – 2.3 to have a better idea of the curves.

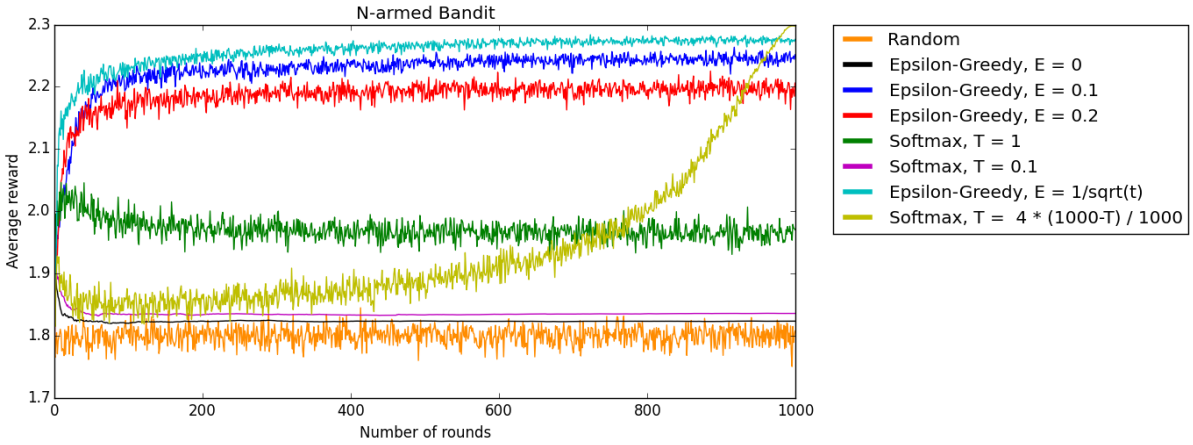


Figure (2) : Average reward for each algorithm (zoomed on the interesting value)

In this plot (Figure 2), it's easier to notice the differences between algorithms.

For random strategy:

No matter what it knows, it will always choose randomly an action the next round. So the mean reward will vary around the mean of every Qa (= Qa*): $\frac{2.3+2.1+1.5+1.3}{4} = 1.8$ for the average reward.

For ε-greedy strategies:

- When ε = 0: In this case, it will always choose exploitation, so always the best possible action in its knowledge. For the first choice, it will choose randomly between every action. When he chooses an action, he will always continue in this action except if the reward is negative which is more possible in the third action. It's why the average reward is a little bit higher than 1.8, the mean of all action (Qa*).

- When ε = 0.1 or ε = 0.2: In this case, we have a probability of ε to choose exploration. If it explores, it will choose randomly a neighbor to explore the different possibilities and try to find a better action. So if the ε = 0.1, it will explore less than ε = 0.2. If it explores less (ε = 0.1), it will increase slowly but it will end up with a better average reward because it will choose less often a "non-optimal action" at the end. In this plot, we can notice that the red curve (ε = 0.2) increases quicker and then it cross the blue curve (ε = 0.1) at approximately 50 rounds. The ε = 0.1 ends up the 1000 rounds with a better average reward than ε = 0.2.

- When ε = $\frac{1}{\sqrt{t}}$ (= ε-greedy(t)): In this case, ε is high at the start and it decreases rounds after rounds. So at the start of the algorithm, it will explore a lot to establish the mean of each action. And after it will exploit more and more to choose more often the best action. It's why it's the quickest algorithm to reach its max average reward.

For Softmax strategies:

Tau represents the computational temperature. If Tau is high, the temperature is high and so he will explore more and the all actions will be equiprobable. If Tau is too low, it will almost never explore. So it won't change its strategy often enough. The higher Tau is, the most equiprobable are all the actions.

- When Tau = 1: In this case, it will increase quickly and then decrease a little to reach a state where it will vary because it will often choose a non-optimal action.

  Imagine the first action it chooses is 1 and he receives a reward of 2, next round:

  Probability to choose action 1: $proba1 = \frac{e^{2/1}}{e^{2/1}+e^0+e^0+e^0} = \frac{7.4}{7.4+3} = 0.71$ and then 10% for every other action. So even with such a difference (= 2) between the actions, it still has 30% of exploration.

  If the first choice (e.g. action3) gives a negative reward of -1, next round: $proba3 = \frac{e^{-1/1}}{e^{-1/1}+e^0+e^0+e^0} = \frac{1.7}{1.7+3} = 0.36$

  So even with a negative reward, it has a probability of 0.36 to choose it again next round.

- When Tau = 0.1: Here it won't explore enough because the temperature is too low. So it will almost always choose the best action and so it will not explore if there is better actions than what it already knows. It still a little bit higher than the ε-greedy(ε = 0) because with Softmax(T = 0.1), there is still a small probability to change its choice.

  Imagine the first action it chooses is 1 and he receives a reward of 2, next round:

  Probability to choose action 1: $proba1 = \frac{e^{2/0.1}}{e^{2/0.1}+e^{0}+e^{0}+e^{0}} = \frac{485\,165\,195}{485\,165\,195+3} = 0.9999$ So he will almost always choose the best action with what he knows. So he will almost choose always the first action that gives him a positive reward.

  If the first choice (e.g. action3) gives a negative reward of -1, next round:

  $proba3 = \frac{e^{-1/0.1}}{e^{-1/0.1}+e^{0}+e^{0}+e^{0}} = \frac{0.000045}{0.000045\ +\ 3} = 0.0001\%$

  So it becomes almost impossible it chooses again this action.

- When Tau = $4 * \frac{1000-t}{1000}$ (= Softmax(t)) : In this case, it's the same idea than the ε-greedy, ε = $\frac{1}{\sqrt{t}}$. But in Softmax(t), it takes more time to find the right average reward. But when it finds the best action, it takes almost only this one. So he takes time to affine the average reward but after, it increases slowly it's average reward and after 1000 rounds it gives the best average reward of all algorithm tested.

Qai vs Qai*

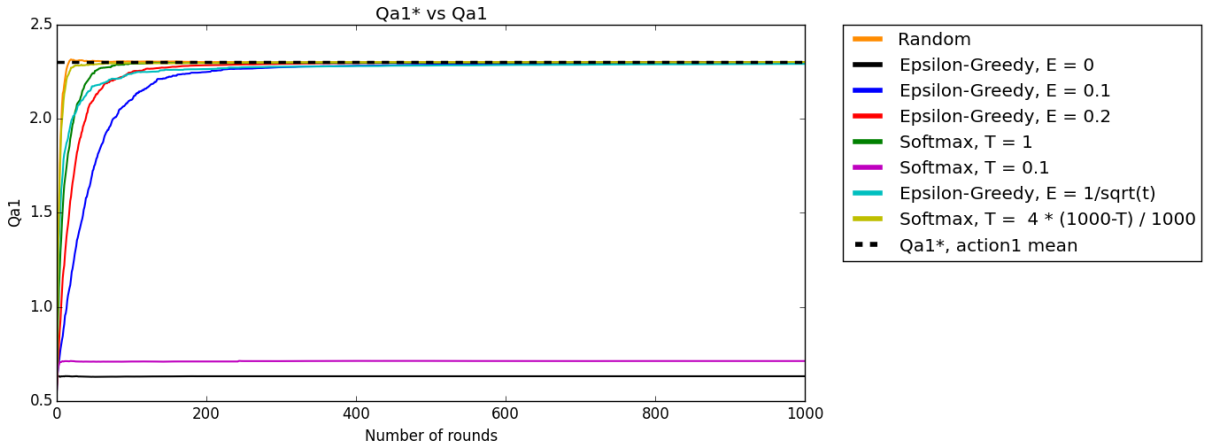


Figure (3) : Qa1 along the actual Qa1*

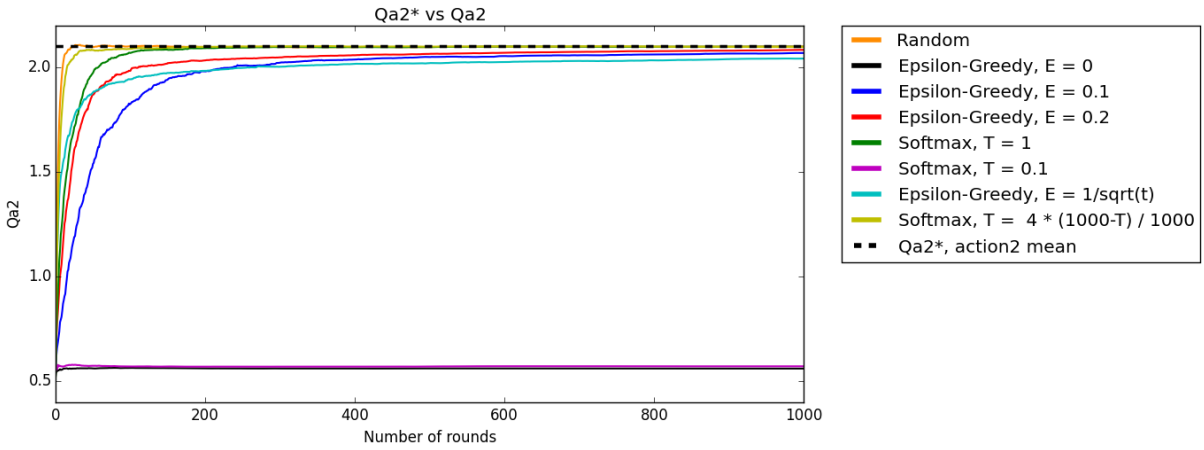


Figure (4): Qa2 along the actual Qa2*

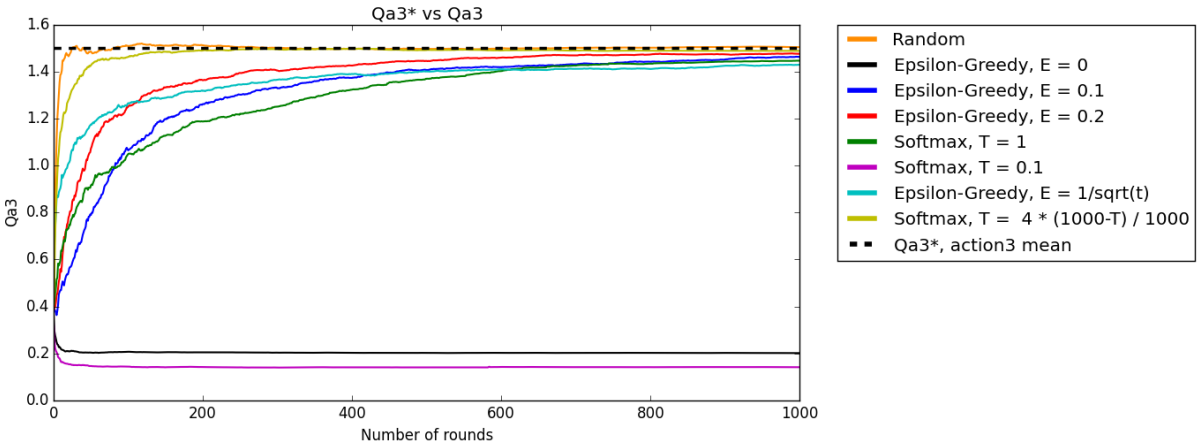


Figure (5): Qa3 along the actual Qa3*

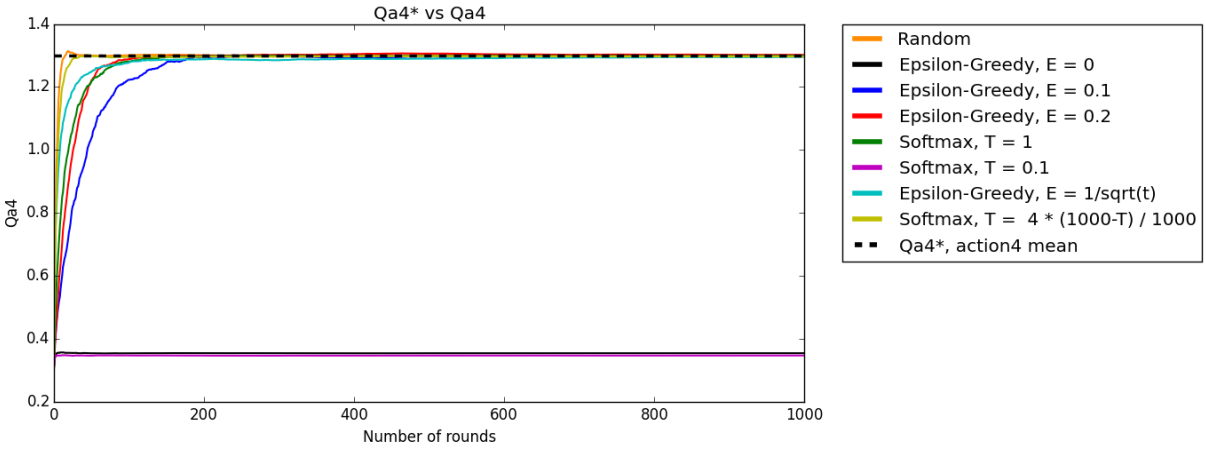


Figure (6) : Qa4 along the actual Qa4*

Qa* = mean of all the Q of that action

In every plot, random algorithm and Softmax(t) reach quicker the mean. For random, it's because he will choose a same action approximately every 4 rounds. So it will explore faster all the actions and so it will find faster the mean. For Softmax(t), it's because the T is high at the beginning so it will explore a lot and it's like a random algorithm.

For Softmax with Tau = 1, it takes more time to reach the mean than a random but it' still faster because the exploration is high.

With ε-greedy = 0 and with Softmax with Tau = 0.1, there is not enough exploration to reach the mean. They can't collect enough information about the different actions. So there are some runs where the action is not taken at all. Each action is taken with probability of 0.25 at the first round and then these algorithms will always keep their first choice except if it is negative (even if the Softmax(T=0.1) still has a very small probability to try another action). So these 2 algorithms will tend to reach the mean of the action multiplied by 0.25:

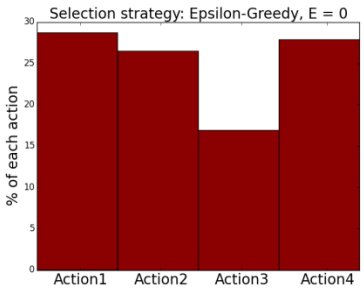
action1→ 2.3/4 = 0.575, action 2 → 2.1/4 = 0.525, action 3 → 0.375, action 4 → 0.325

With ε-greedy = 0.1 and ε-greedy = 0.2, the exploration is higher than ε-greedy = 0 but lower than random. We can notice that ε-greedy = 0.2 reach quicker the mean than ε-greedy = 0.1 because ε = 0.2 allows more exploration. With ε-greedy (t), it will increase higher than ε = 0.2 and ε = 0.1 at the beginning. And after it will cross the curve of ε = 0.1 where $\frac{1}{\sqrt{t}} = 0.1$ so where t = 100 rounds and it will cross the curve of ε = 0.2 where $\frac{1}{\sqrt{t}} = 0.2$ so where t = 400 rounds.

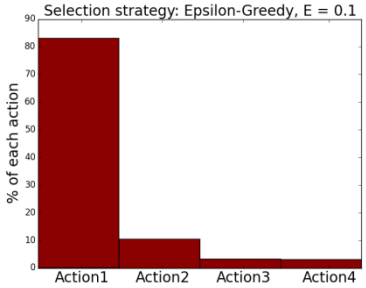
It takes a little bit more time for Qa3 to reach the mean because the standard deviation is high. And it's not the best action. So if the algorithms can learn they will choose less and less this action over the time.

The higher the rate of exploration in the algorithm is, the quicker the algorithm reaches the mean.

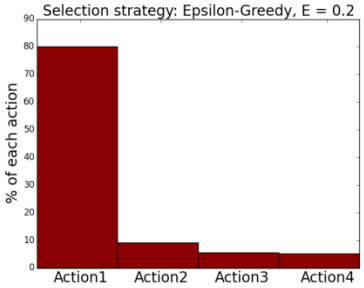
1. RANDOM



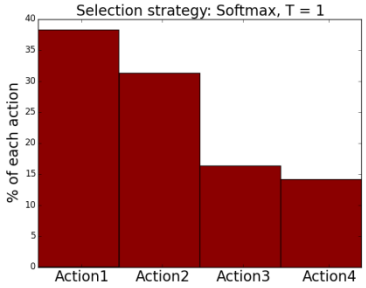
2. Epsilon-Greedy, E = 0



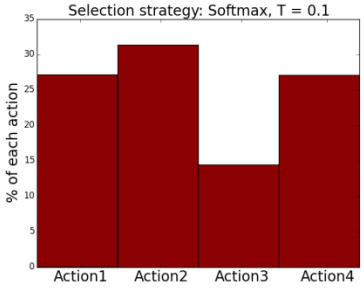
3. Epsilon-Greedy, E = 0.1



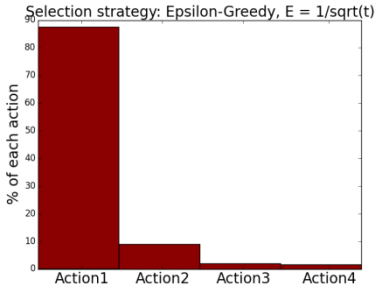
4. Epsilon-Greedy, E = 0.2



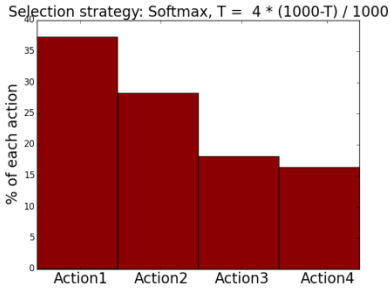
5. Softmax, T = 1



6. Softmax, T = 0.1



7. Epsilon-Greedy, E = $1/\sqrt{t}$



8. Softmax, T = $4 * \frac{1000-t}{1000}$

Figure (7) : Histogram showing the number of times each action is selected depending on the selecting strategy

1) The random algorithm gives equal probability to choose each action.

2) The ε-greedy(ε = 0) algorithm gives almost equal probability of action1, 2 and 4 (a little bit higher for action1). But the number of actions 3 is lower because it can give more often negative reward and in this case, the algorithm will chose another action.

3) The ε-greedy, ε = 0.1 algorithm gives a high probability to choose action 1 which is the best one. The number of an action is lower when the average reward is lower.
Action 1 > Action 2 > Action 3 > Action 4

4) The ε-greedy, ε = 0.2 algorithm gives almost the same histogram as the ε-greedy(ε = 0.1) algorithm.

5) The Softmax(T = 1) algorithm gives a higher number of action 1 but not as much as the ε-greedy, ε = 0.2 or 0.1 because it explores to much so it chooses too often a non-optimal action.

6) The Softmax(T = 0.1) algorithm gives pretty much the same histogram than the ε-greedy(ε = 0) because they have almost the same rate of exploration. They both choose almost always the same action during all the runs.

7) The ε-greedy, ε = $\frac{1}{\sqrt{t}}$ algorithm gives the best percentage of best action (action 1). Because it will explore at the beginning and exploit much rounds after rounds. So at the end, it will choose almost always the best action.

8) The Softmax, T = $4 * \frac{1000-t}{1000}$ algorithm gives a higher number of action when the mean reward of this action is higher. We don't notice such a high number of action 1 than ε-greedy, ε = $\frac{1}{\sqrt{t}}$. In fact, it explores quicker at the beginning so randomly choose each action. But rounds after rounds, the Tau is decreasing so at the end, it will only choose the best action. In the histogram, we only see the number total of each action so we can't see that beginning gives random action and that the end gives only best action.

Conclusion of exercise 1

After 1000 iterations, we have from the best average reward to the worst:

Softmax(t)
ε-greedy(t)
ε-greedy(0.1)
ε-greedy (0.2)
Softmax(T=1)
Softmax(T=0.1) ⎫
ε-greedy(0) ⎬ Almost the same
Random ⎭

The faster to reach its maximum average reward:

ε-greedy(t)
ε-greedy(0.1)
ε-greedy (0.2)
Softmax(T=1)
Softmax(t)
Softmax(T=0.1) ┐
ε-greedy(0) ├ Almost the same
Random ┘

Softmax(T=0.1) and ε-greedy(0) don't explore enough to find another best action so they will tend to the mean reward of all action (Qa* = 1.8). Random will also tend to the mean because it explores always so it doesn't even take into account the best value.

Softmax(T=1) explores too much. So even if it finds the best action, it will continue to take a lot of non-optimal actions.

E-greedy(0.2) explores more than ε-greedy(0.1) so ε-greedy(0.2) increases faster but its final average reward is lower. Indeed, after 1000 rounds ε-greedy(0.2) will still choose to many times non-optimal actions.

E-greedy(t) learns quicker than the other because the exploration is high at the beginning but decrease over the time. So rounds after rounds, it will chose more often its best action to increase its reward.

Softmax(t) gives the best final average reward. It takes time to learn which one is the best action. But when it's done, it will increase slowly to the maximum average reward and will tend to choose always the best action (in our case: action 1).

So for a short term reward, I will choose the ε-greedy over the time because he increases faster than the others. But for a long term rewards I will choose the Softmax over the time because at the end, it always takes the best action.

1.2) Double standard deviation

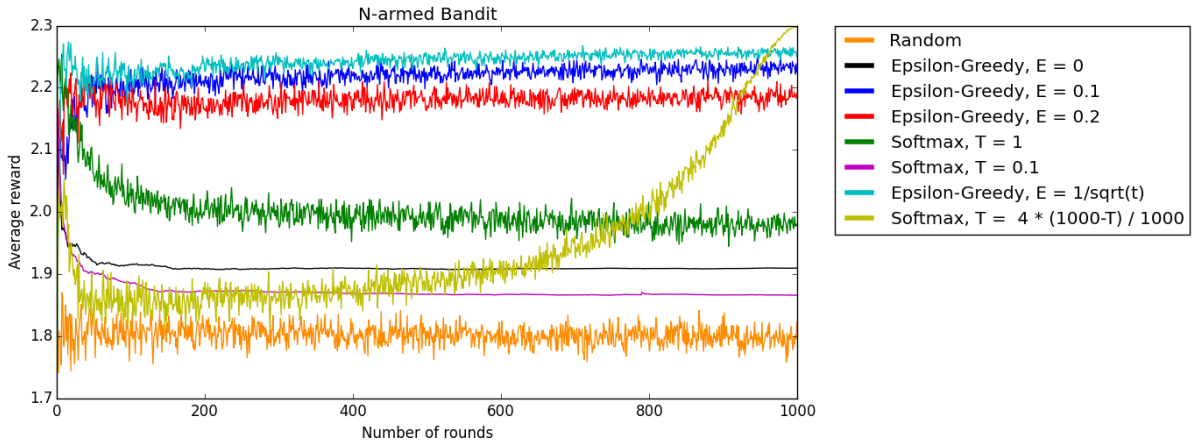


Figure (8) : Average reward for each algorithm with double standard deviation (zoomed on the interesting value)

We can't notice any significant difference between average reward with normal and double standard deviation except some higher variations.

Softmax(T = 0.1) and ε-greedy(0) give a little bit higher average reward. Indeed, there is a better probability to reach a negative reward (mainly with action 3). And in this case, it will choose another strategy giving a better mean reward.

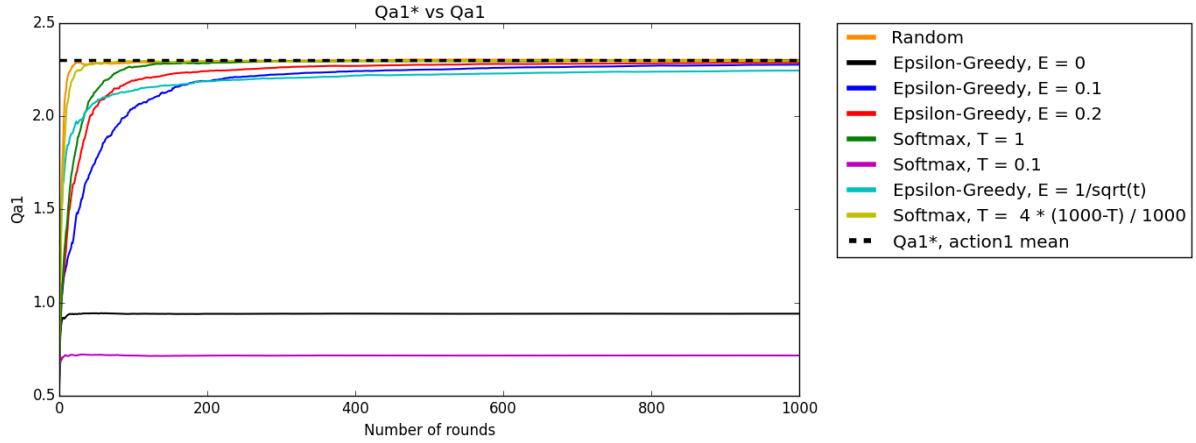


Figure (9) : Qa1 along the actual Qa1 with double standard deviation*

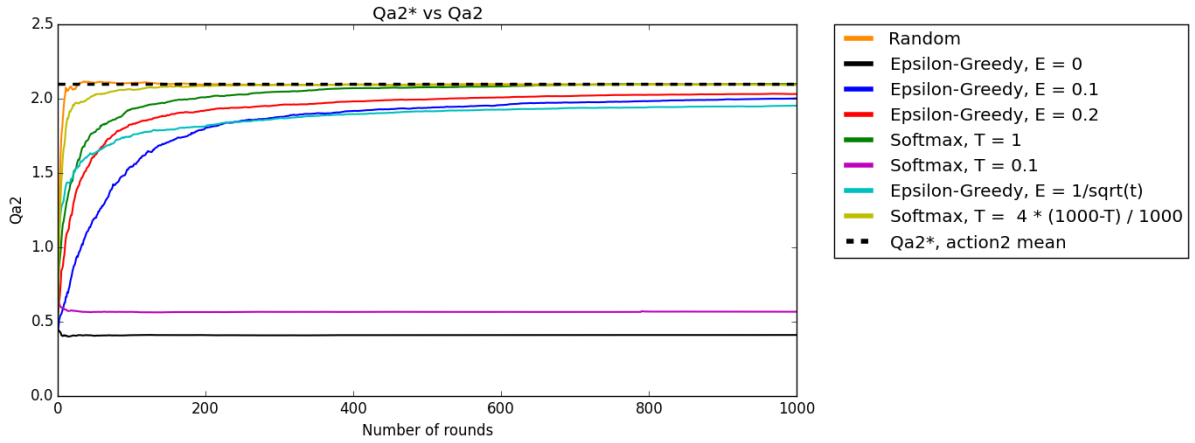


Figure (10) : Qa2 along the actual Qa2 with double standard deviation*

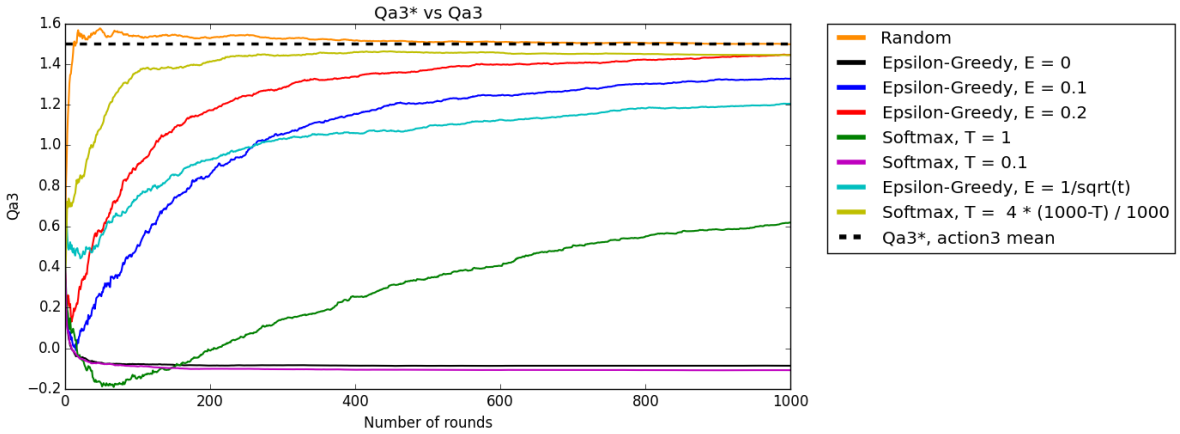


Figure (11) : Qa3 along the actual Qa3 with double standard deviation*

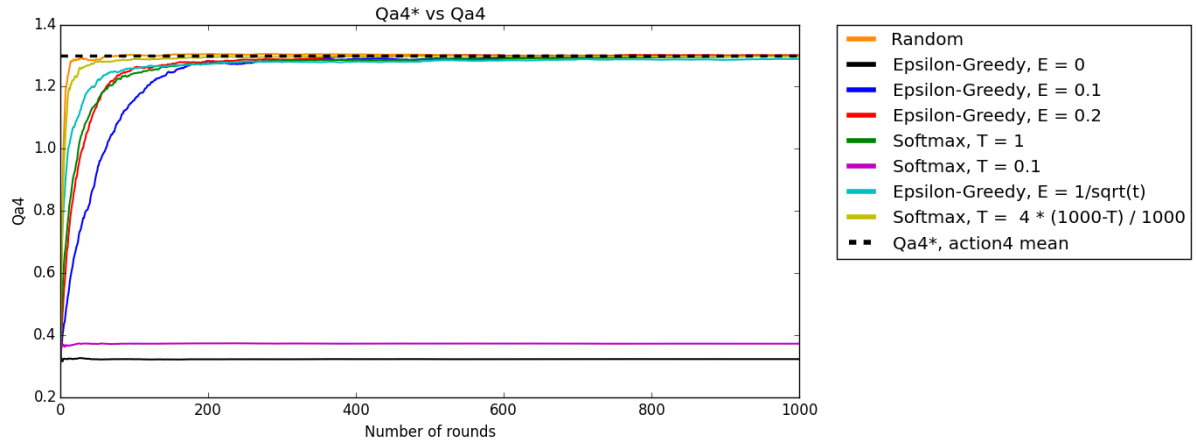
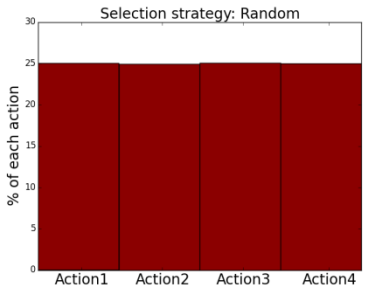


Figure (12) : Qa4 along the actual Qa4 with double standard deviation*

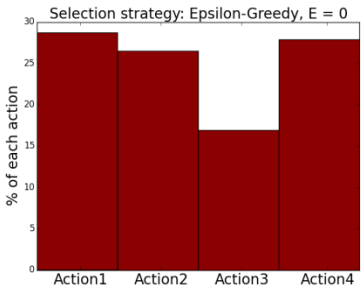
When the standard deviation is doubled, each algorithm will take more rounds to find the mean of each action except the random which takes as few rounds as with normal standard deviation.

For Qa3, it doesn't reach the Qa3* very often. For Softmax(T=0.1) and for ε-greedy(0) the Qa3 will even be just under the 0. Indeed when the reward of action 3 is negative these algorithms will never choose this action again and so the average reward in this run will stay negative. And if these algorithms take another action as first choice, they will keep it during all the rounds. So the mean of all Qa3 will be a little bit under 0.

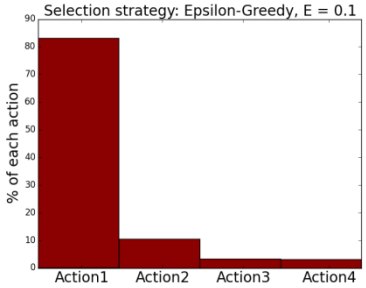
The Softmax(T=1) goes under 0 for the first few round and then increases rounds after rounds showing it collects more and more Qa3 over the time because in this algorithm, the exploration is high. It goes under 0 because action 3 can have negative reward because of its high standard deviation.



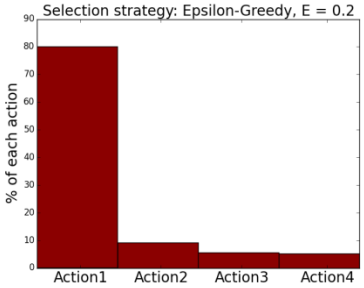
1. RANDOM (σ * 2)



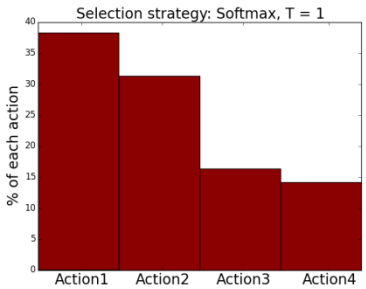
2. Epsilon-Greedy, E = 0 (σ * 2)



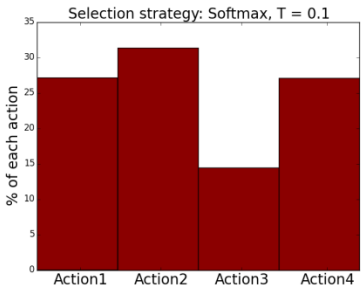
3. Epsilon-Greedy, E = 0.1 (σ * 2)



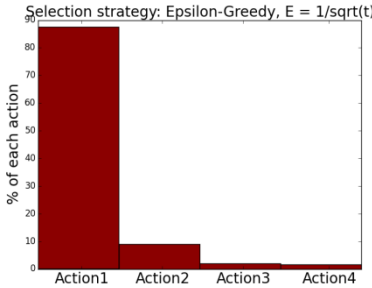
4. Epsilon-Greedy, E = 0.2 (σ * 2)



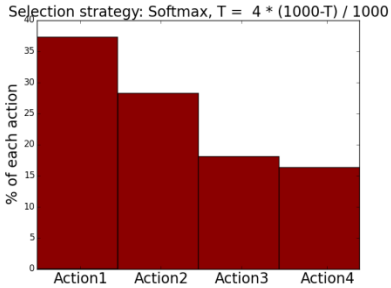
5. Softmax, T = 1 (σ * 2)



6. Softmax, T = 0.1 (σ * 2)



7. Epsilon-Greedy, E = $1/\sqrt{t}$ (σ * 2)



8. Softmax, T = $4 * \frac{1000-t}{1000}$ (σ * 2)

Figure (13) : Histogram showing the number of times each action is selected depending on the selecting trategy

All histograms are almost the same than the ones with the original standard deviation.

1.3) Algorithm depending of the time

These algorithms were analyzed in the first part of this report.

2. Windy Gridworld

2.1) Plotting

SE	SE	SE	SE	SE	E	NE	SE	S	SW	S	SW
NE	SE	E	NE	E	NE	E	NE	SE	S	SW	W
NE	E	NE	NE	NE	NW	E	NE	SE	S	SW	NW
E	NE	NE	NE	N	NW	NE	NE	E	GOAL	W	W
E	NE	NE	N	NE	N	NE	N	NE	N	N	NE
NW	NE	S	NE	S	N	NE	NE	NE	S	N	NW
SW	S	SE	NE	NE	SE	E	N	W	SE	SW	S
0	**0**	**1**	**1**	**1**	**2**	**2**	**1**	**0**	**0**	**0**	**0**

Figure (14) : Windy gridworld with the best direction given by a geographic direction

Here is the plot with an epsilon = 0.2. In this grid, I write the best direction he will choose by running my algorithm with only exploitation to find the best direction.
N = North, NE = North-East, E = East, SE = South-East, S = South, SW = South-West, W = West and NW = North-West.

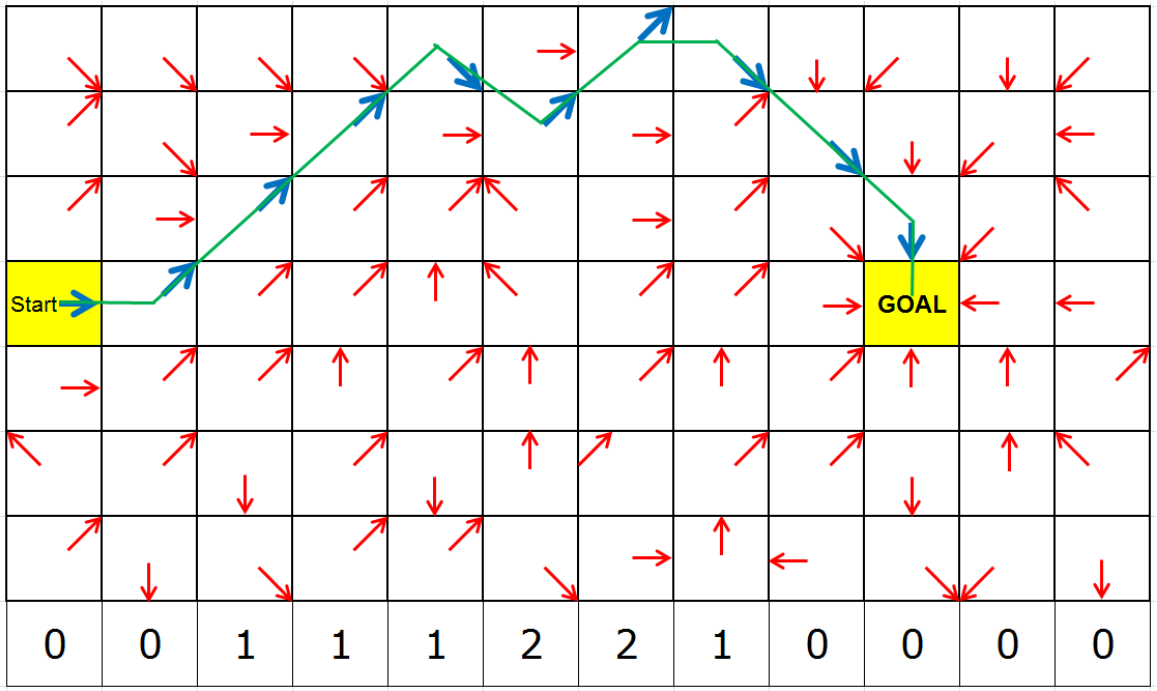


Figure (15) : Windy gridworld with all best action

Then I transform each best action in a red arrow showing the best action taken in this state. In blue we can see the best action given by the last run with an epsilon = 0, so with only exploitation (Figure 15). And in green it's the line showing the best pathway used to reach the goal.

If we run the algorithm again, it will find other pathways which are similar but they all go to the top of the grid because of the wind. Often, the agent goes to the north and goes east along the border because the wind can't push him further north.

In some runs, there is a gap between two blue arrows; it's when the wind brings the agent further than his direction choice. In our case, it doesn't append.

The right bottom is not enough explored to be significant. Indeed, the run stops when the agent reaches the goal so the goal will "attract" all his neighbors and the agent won't go further.

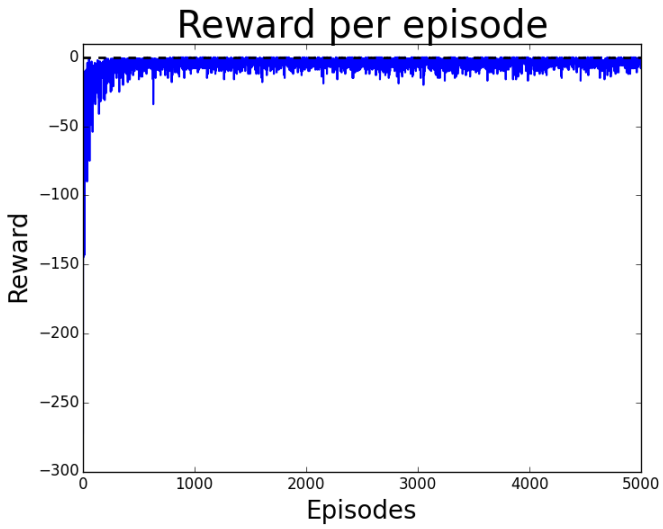


Figure (16): Total collected reward per episode with epsilon = 0.2

In figure 16, we can notice that the first few episodes give a very low reward because the agent has to make a lot of episodes to reach the goal and each step cost him -1 as reward. Then he will learn which pathways are better and so the rewards will increase to almost reach the 0. The shorter way to reach the goal is 8 steps (with a reward of -1) and the last giving a reward of 10. So the best reward an agent can have is 2. That why we finally vary around the 0. And because of the exploration, he won't take always the best pathway to have his best reward.

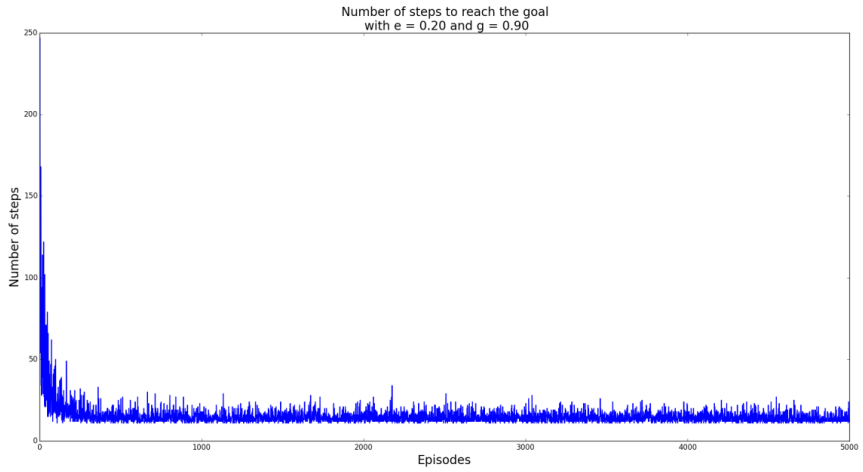


Figure (17): Number of steps to reach the goal per episode if e= 0.2

Logically, figure (17) looks like the reward plot (Figure 14) because the more steps the agent does, the lowest the reward is. So the first few steps, the agent has to do a lot of steps to find the goal. Then he learns the best directions to choose and he will vary around 10 steps.

2.2) Discussion

2.2.1) Epsilon = 0

Here I run my algorithm with a ε = 0 and I kept a gamma = 0.9. If we look at the best pathway taken in this algorithm, it's quite the same than with ε = 0.2

If ε = 0.2, there is some exploration. So the first few steps of ε = 0 can be longer than ε = 0.2 because ε = 0 cannot take the same action in a certain state twice because the reward of each step will decrease the Qsa of that action in that state.

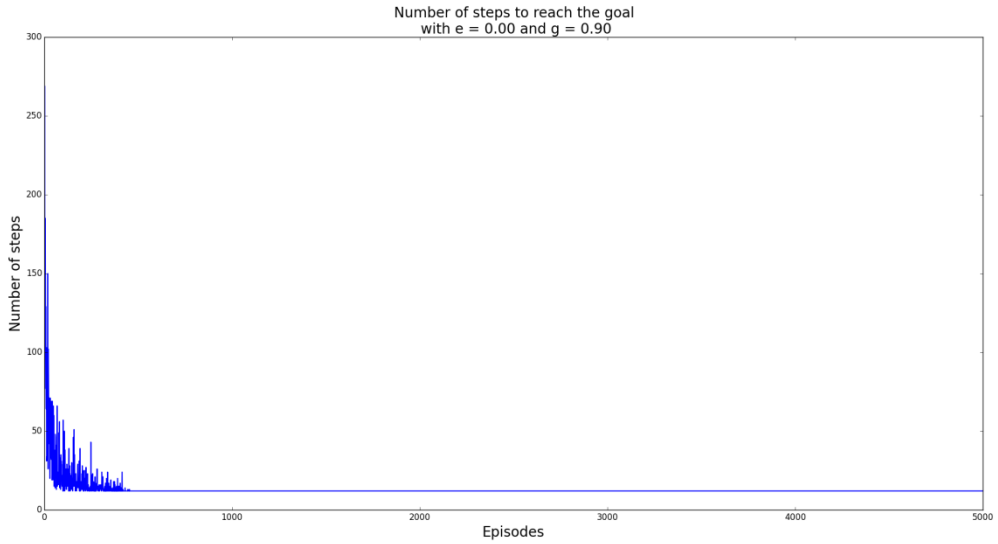


Figure (18): Number of steps to reach the goal per episode if e= 0

The ε=0.2 algorithm will reach faster its average number of steps and will vary a little around this because the exploration will sometimes increase the number of steps by choosing a non-optimal action (Figure 17).

The ε=0 will reach little bit later its average number of steps but when it does, it will be always the same number of steps to reach the goal (around 9), there is no more variations (Figure 18).

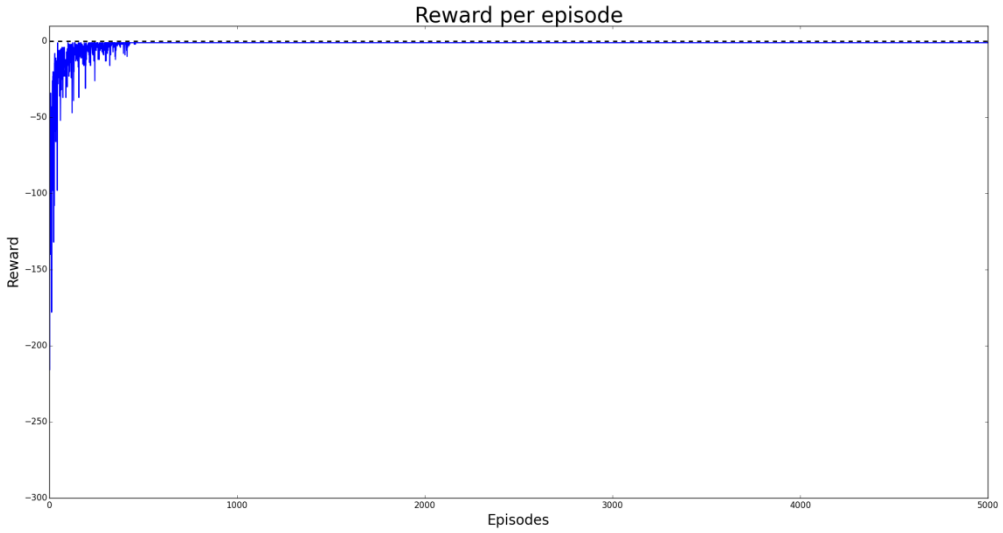


Figure (19): Total collected reward per episode with epsilon = 0

Logically we can notice the same plot than the number of steps but inverted because each step gives a negative reward. So the more steps we do, the lowest the reward is (Figure 19).

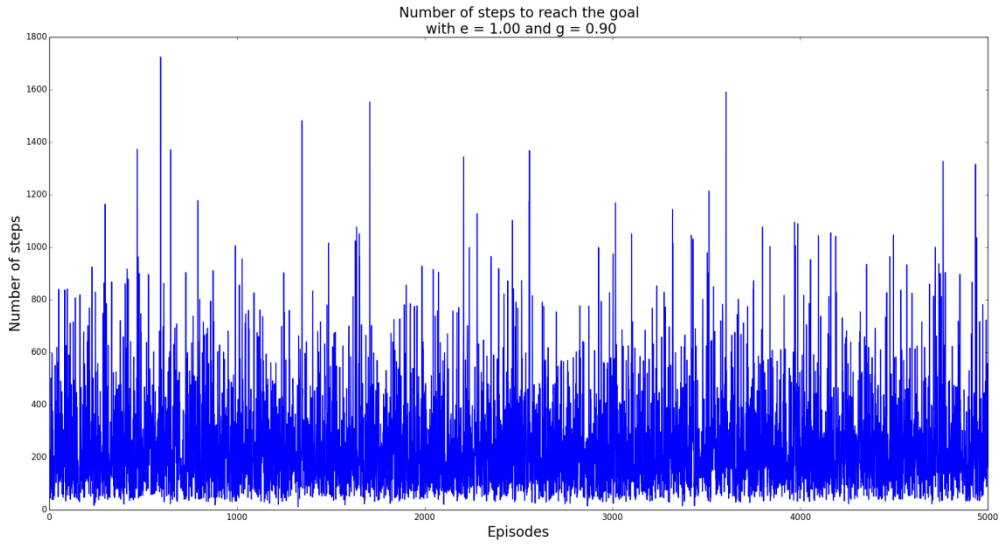


Figure (20): Number of steps to reach the goal per episode if e= 1

Here I try a ε = 1 and we can see that he will always choose a random pathway to reach the goal. He doesn't learn.

2.2.2) Gamma = 1

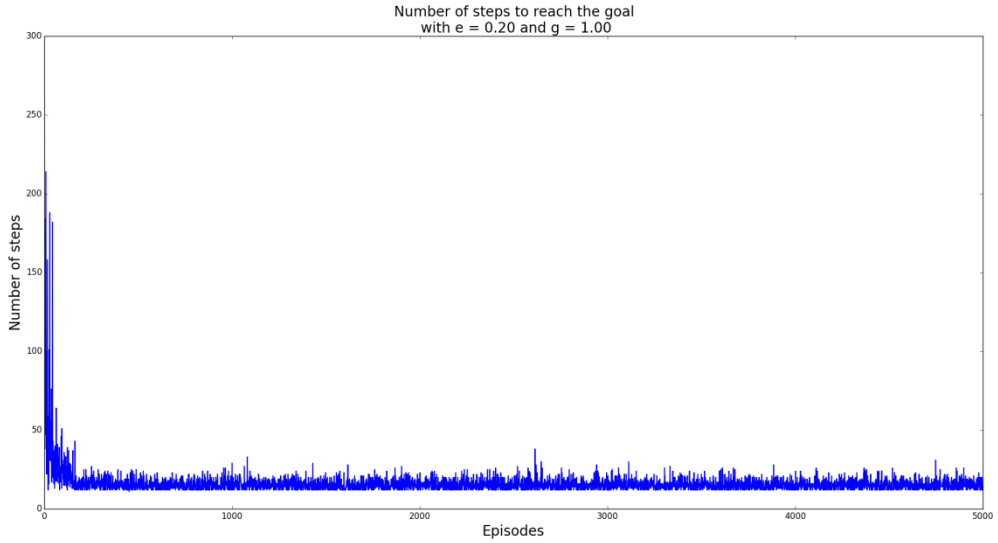


Figure (21): Number of steps to reach the goal per episode if e= 0.2 and gamma = 1

Gamma is the discount factor. The gamma value gives the importance of the next move and gives the long term view to the algorithm. If gamma is equal to 0, he will be lost in the grid because he won't consider the maxQt in the equation. If gamma is equal to 1 (Figure 21), he will reach faster his best pathway but with higher variation the first few steps because it will look too far from the origin.

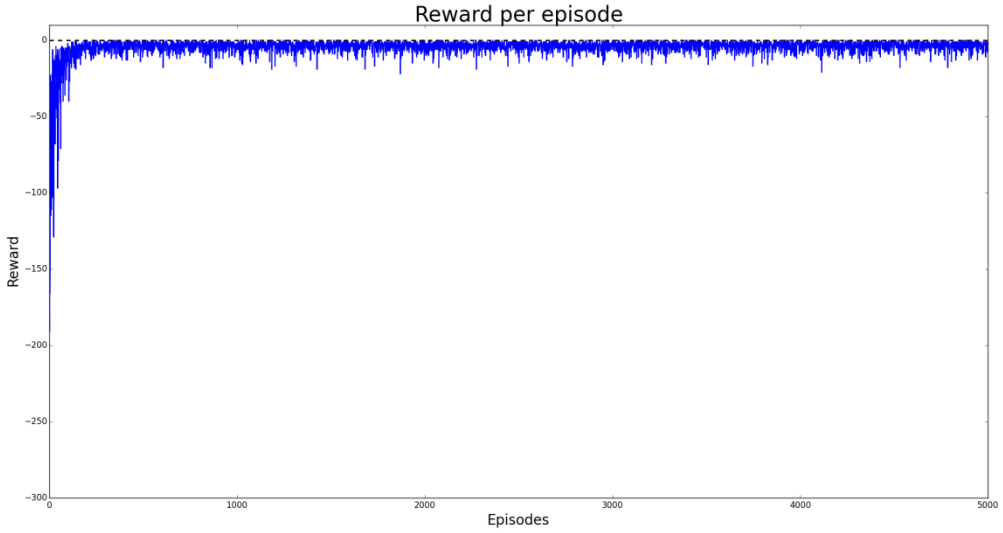


Figure (22): Total collected reward per episode with epsilon = 0.2 and gamma = 1

Logically we can notice the same plot than the number of steps but inverted because each step gives a negative reward. So the more steps we do, the lowest the reward is (Figure 22).

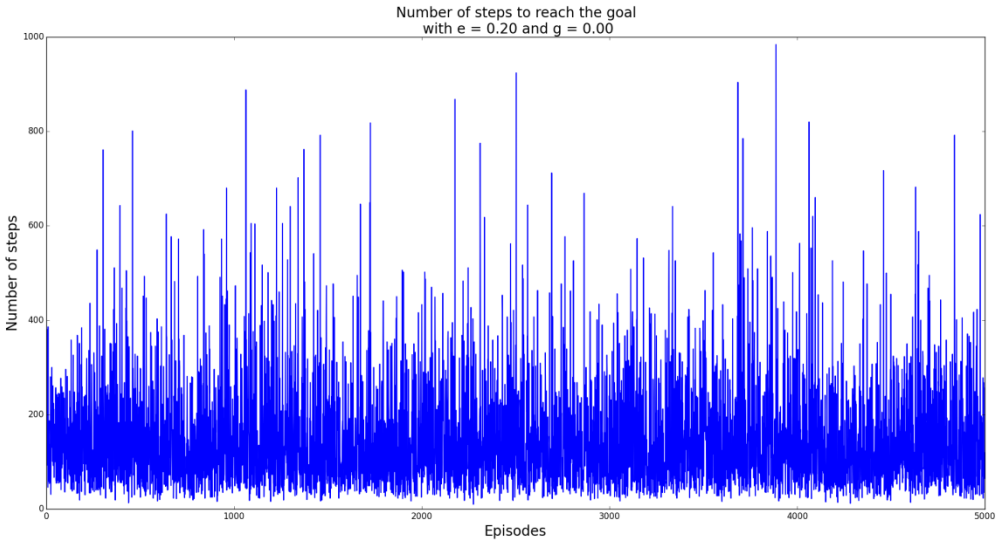


Figure (23): Number of steps to reach the goal per episode if e= 0.2 and gamma = 0

Here I tried with a gamma of 0 and we can notice that there is no learning. It takes always a random pathway to reach its goal because he doesn't take into account the maxQt (Figure 23).

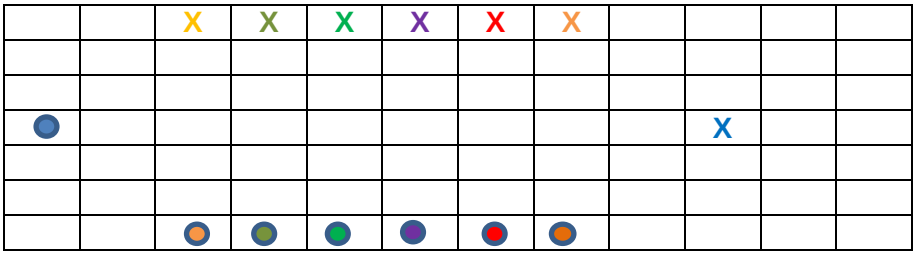
It seems the higher gamma is the faster will be the learning in our case.

3. Windy Gridworld: Discussion

The agent who wants to reach the goal is the first agent and the wind is considered as the second agent who goes upward. They can't cross each other so there is interactions between the 2 agents. They have to avoid collisions.

When they cross each other, the first agent will always goes up depending on the force of the second agent (the wind).

We can also consider the wind as many agents going upwards in each column where there is wind. Like this:



It must be an algorithm using interaction and where agents aren't acting independently.

Intuitively I would choose the Join State Action because we are in a Grid game and the agents have to interact with each other. We are also in a State Action game.