CS6700 : Reinforcement Learning
Programming Assignment #3

Author: Solène Butruille
Roll number: CS20F001

1 Hierarchical Reinforcement Learning

In order to visualize the Q values, I decided to print the grid word and to put in the squares the option which has the highest value in the Q Values table for this state. States represented with a "-" means that they were never chosen as starting state. Below you can find the legend.
0 : option in room 1, goal = right hallway
1 : option in room 1, goal = hallway down
2 : option in room 2, goal = left hallway
3 : option in room 2, goal = hallway down
4 : option in room 3, goal = hallway up
5 : option in room 3, goal = left hallway
6 : option in room 4, goal = hallway up
7 : option in room 4, goal = hallway right
D : down
R : right
L : left
U : up

1.1 Solution without intra-option Q learning

1.1.1 Q Values for a goal in a hallway

I started with a lot of tests on the parameters. I finally found that the α value doesn't affect much the final values as long as it is > 0 and we have an important number of iterations. In fact, the final result are not depending on the α value because I feel that they depend on the first chosen states which are actually random. As we can see bellow (they are 2 representations for $\alpha = 0.1$), with the exact same parameters, I can have results very different. It is not a real problem as they are multiple possibilities to reach the goal. The thing that doesn't change with the same parameters is the exploration (number of states with a value). I choose to take small values for α in order to have low computation. To have optimal results, I decided to run 10 000 episodes. I think it is a bit too much but it is not too slow and at least I am sure that it got close to convergence. I decided to plot my results for $\alpha = 0.3, 0.1(X2), 0.01$ as I explained above, the difference which we can see the most is the exploration.

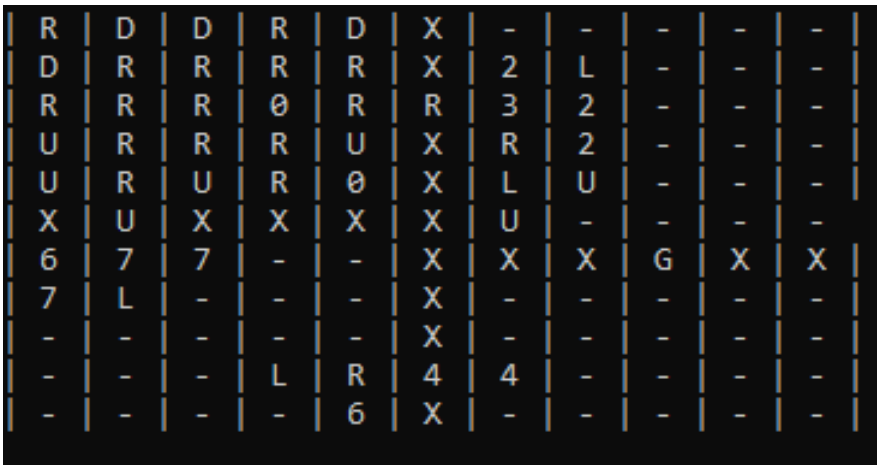


Figure 1: $\alpha = 0.3$

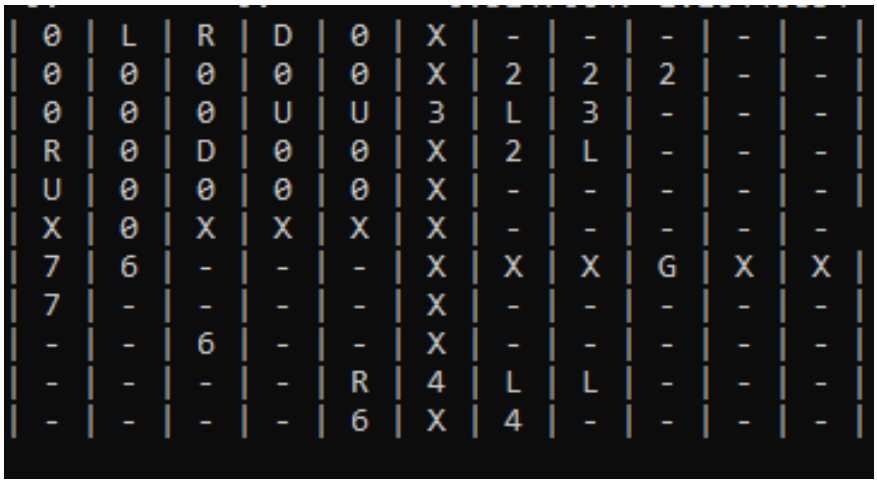


Figure 2: $\alpha = 0.1$

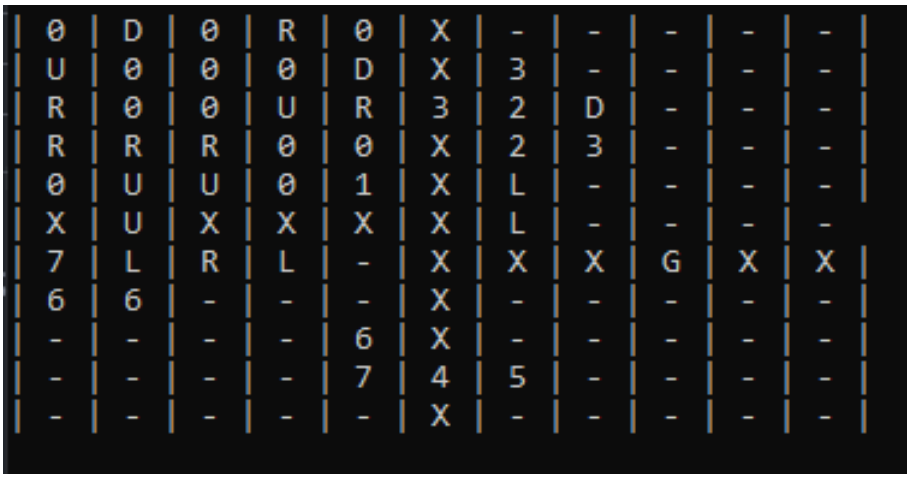


Figure 3: $\alpha = 0.1$

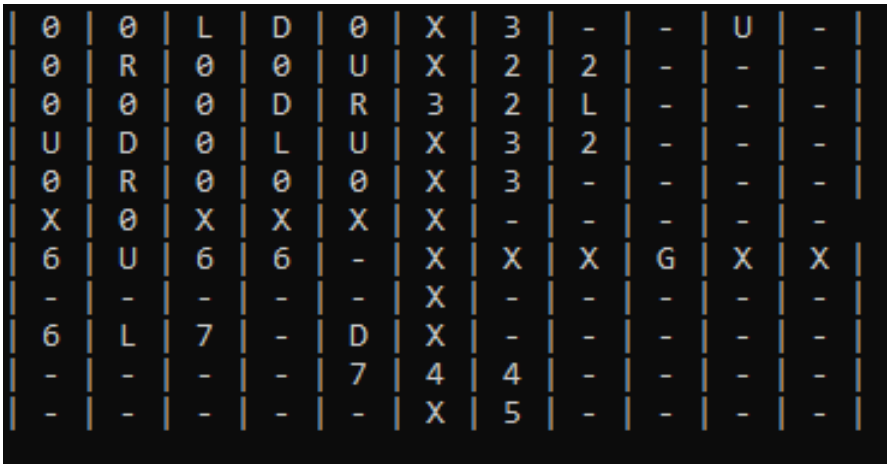


Figure 4: $\alpha = 0.01$

1.1.2 Q Values for a goal in a room

If the goal is now inside a room instead of in a hallway, it changes a lot because the options can't find it. As the option's goal are to go to a hallway, it happens that the option goes through the goal but doesn't stop because it is not the sub-goal at this time. Therefor, we can see that with this goal, it will need to explore more to find the goal.

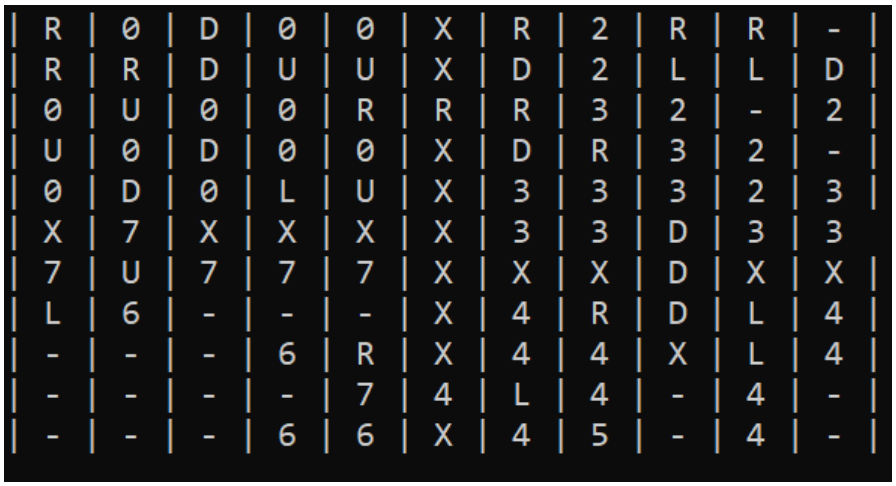


Figure 5: $\alpha = 0.1$, Goal inside the room

1.1.3 Comparison without noise

I also find it interesting to compare the result with the classic environment to results in an environment without noise. What I noticed is that sometimes a state is going to take quickly a big value (for example the hallway between the first and second room.) Then all states of room 1 will chose to go to that state even if there is another solution better closer. On the figure 5, we can even see that some states in the room under the room 1 are choosing to go back in room 1 and then to go to the hallway of room 2 despite the fact that it is longer. As explain above, my theory is that the hallway between room 1 and 2 as such an important value in the q table that it crushes all chances for the other possibilities.

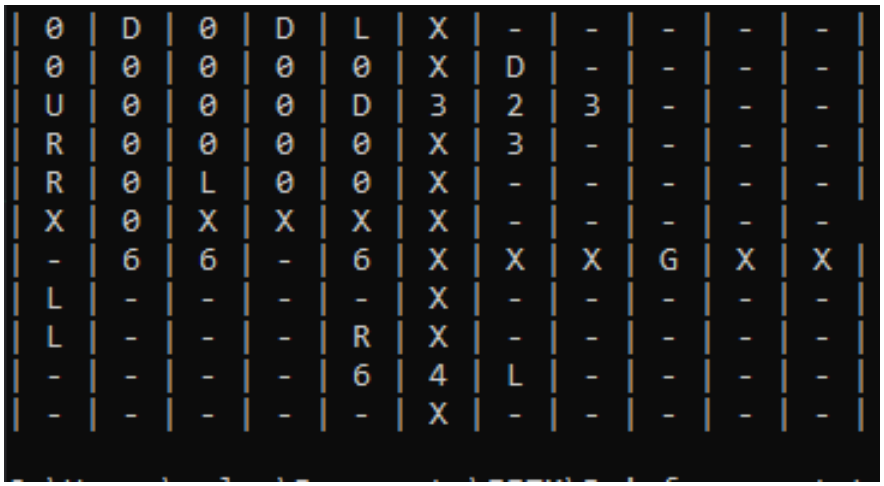


Figure 6: $\alpha = 0.1$, Goal in the hallway, no noise

1.2 Solution with Intra option Q learning

If we now implement q learning inside the options, it is going to take much time to converge to the optimal policy. Because the options will have to be learn, there values will not be very good and the algorithm will chose more often to take simple values (U,D,R,L) instead of options.

1.2.1 Q Values for a goal in a hallway

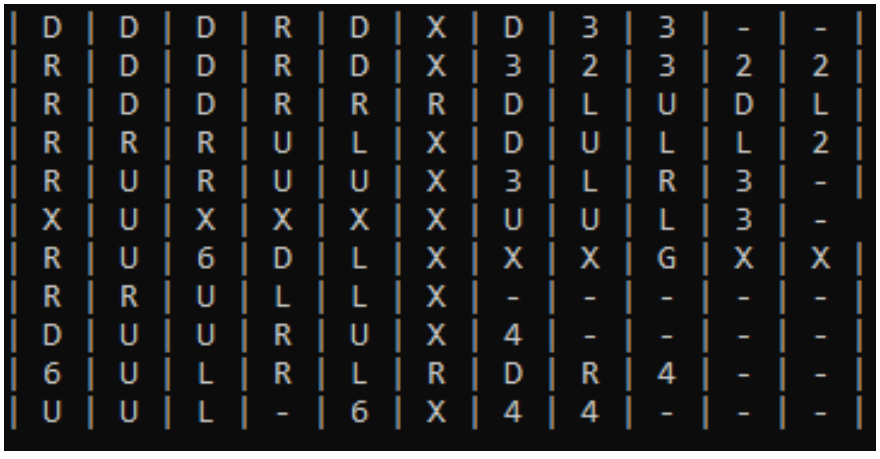
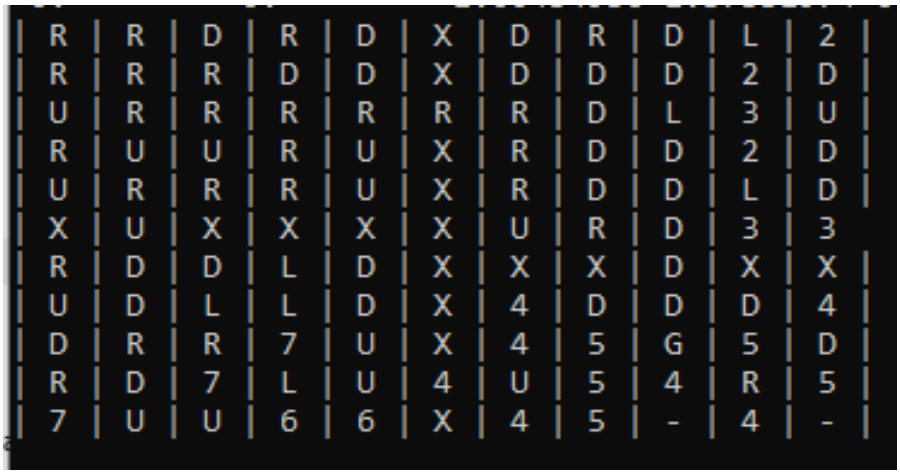


Figure 7: $\alpha = 0.1$, Goal in the hallway

1.2.2 Q Values for a goal in a room

In this special case, we can see that the exploration is almost maximal. Almost every state will be at least once a starting state, whereas in the other situations, a lot of states are never starting state.



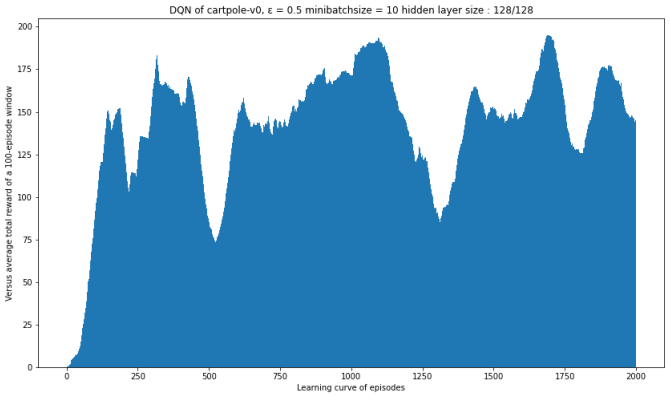
R	R	D	R	D	X	D	R	D	L	2
R	R	R	D	D	X	D	D	D	2	D
U	R	R	R	R	R	R	D	L	3	U
R	U	U	R	U	X	R	D	D	2	D
U	R	R	R	U	X	R	D	D	L	D
X	U	X	X	X	X	U	R	D	3	3
R	D	D	L	D	X	X	X	D	X	X
U	D	L	L	D	X	4	D	D	D	4
D	R	R	7	U	X	4	5	G	5	D
R	D	7	L	U	4	U	5	4	R	5
7	U	U	6	6	X	4	5	-	4	-

Figure 8: $\alpha = 0.1$, Goal in the room

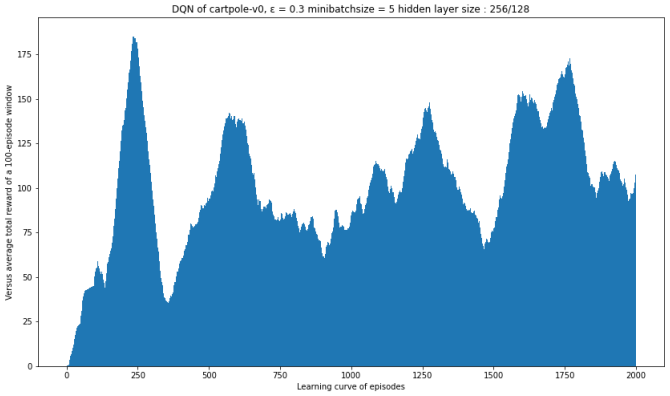
2 Deep RL

2.1 Plot of the DQN algorithm

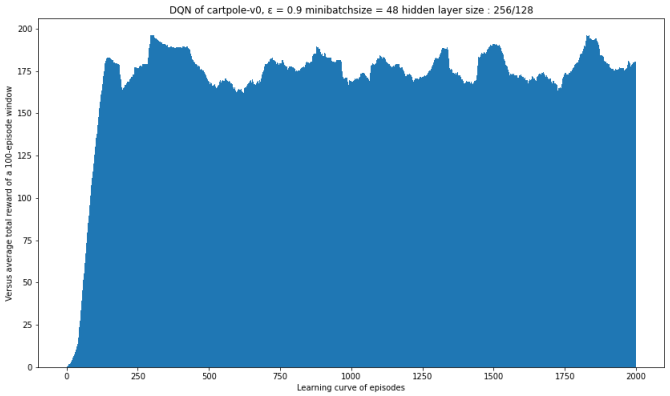
I did a lot of tests (more than 30) to look for the best set of hyper parameters. You can find the results in the boardin section 2.3. I decided to show only the more interresting plots here. I first started with the given set of hyper parameters :



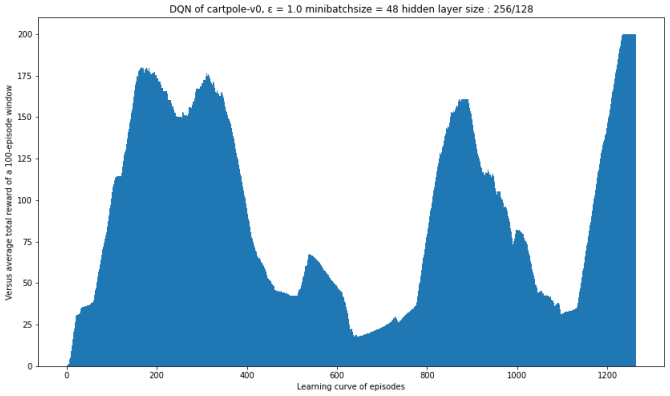
Then, I tried to change the epsilon/hidden layer/ minibatch size which helped me finding better parameters.

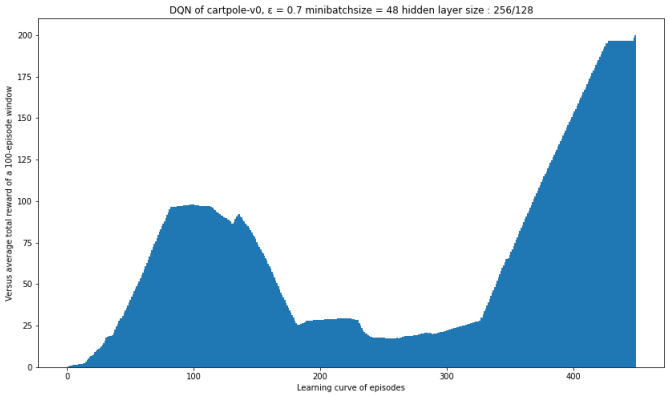


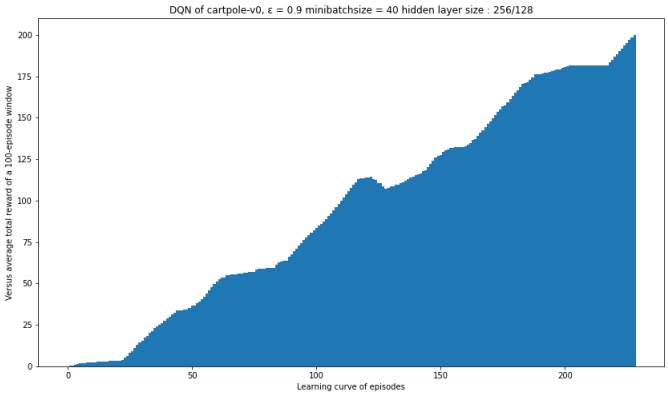
I finally found a good set of hyper parameters :



And then, I changed the discount factor from 0.9 to 1.0, the plot's form really changed. Before, it was more or less homogeneous but it never solved the problem and after, it was more up and downs and it solved the problem :







2.2 Best set of hyper parameters

The set of hyper parameters which is working the best for me is :
REPLAY MEMORY SIZE = 10000
EPSILON = 0.9
EPSILON DECAY = 0.99
EPISODES NUM = 2000
MINIBATCH SIZE = 40
DISCOUNT FACTOR = 1.0
TARGET UPDATE FREQ = 100
HIDDEN1 SIZE = 256
HIDDEN2 SIZE = 128
LEARNING RATE = 0.0001

indeed, with this set of hyper parameters, the problem is always solve and often it doesn't use more than 500 episode to solve it.

2.3 Observations on variation of certain hyper parameters

γ	Hidden layer(s)	ϵ	Minibatch	Solved	Max consecutive eps	Last Episode nb	Final Mean
0.9	128 / 128	0.5	10	False	18	1100	200
0.9	128 / 128	0.8	10	False	12	1018	83
0.9	128 / 128	0.3	10	False	29	1517	200
0.9	256 / 256	0.3	10	False	26	1037	199.72
0.9	512 / 256	0.3	10	False	12	502	123.66
0.9	256 / 128	0.3	10	False	28	429	153
0.9	256 / 64	0.3	10	False	14	1770	125.16
0.9	256 / 128	0.3	5	False	44	206	199.86
0.9	256 / 128	0.3	48	False	63	933	200
0.9	256 / 128	0.3	64	False	60	81	177.4
0.9	256 / 128	0.3	128	False	58	773	200
0.9	256 / 128	0.1	48	False	45	1714	108.32
0.9	256 / 128	0.9	48	False	47	88	199.18
0.9	256 / 128	1.0	48	False	19	1529	142.72
1.0	256 / 128	1.0	48	True	100	1263	200
1.0	256 / 128	0.9	48	True	100	371	200
1.0	256 / 128	0.5	48	True	100	875	196.04
1.0	256 / 128	0.8	48	True	100	175	200
1.0	256 / 128	0.7	48	True	100	449	200
1.0	256 / 128	0.8	25	True	100	356	200
1.0	256 / 128	0.8	64	True	100	1838	200
1.0	256 / 128	0.8	55	True	100	1461	200
1.0	256 / 128	0.8	40	True	100	187	200
1.0	256 / 128	0.8	45	True	100	856	200
1.0	256 / 128	0.8	48	True	100	623	200
1.0	256 / 128	0.9	40	True	100	218	200
1.0	256 / 128	0.9	40	True	100	228	200

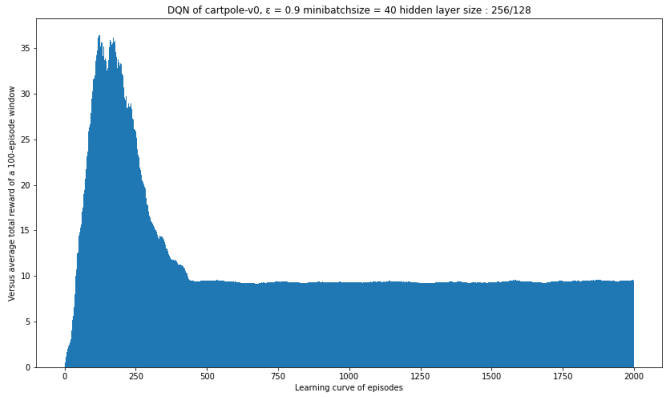
Actually, in this set of hyper parameters, one of the most important thing is the discount factor. Indeed, if it is less than 1 it is almost never going to solve the problem, if it is equal to 1,it will usually solve the problem even if the other parameters are a bit different from optimal ones. It will just be slower. From what I noticed, on the other hyper parameters, it seems like choosing small values won't give good results, but if values are too big then the results are not good either. It takes a really long time to find a value which is not too big but not too small. For example, for the minibatch size, I started with 10 and hen I tried 5 which was really too small and then I also tried with 128 which was too big this time. I finally found that the best values where 48 and 64 and I chose

to keep going with 48 because it had less computing. At the end, when I found a set of hyper parameters which was solving the problem, I decide to readjust the minibatch size and I found that 40 was even working better. I did the same method to find all my parameters. That is how I finally starting from epsilon = 0.5, I chose epsilon = 0.3 for most of my test but readjusting at the end, it happens that the best value was 0.9. And for the hidden layers, I found that with both layers with big values, it was not working well and moreover that too big value (512) was not helping so I finally chose 256/128.

2.4 Removing experience replay and/or target network

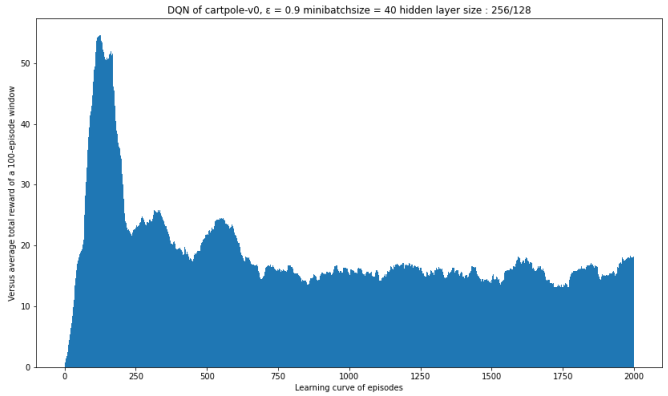
2.4.1 Removing experience replay

If I remove the experience replay and I make my target only on the state that I just compute, my model is going to learn a little bit a the beginning but then it will stop learning and even decreasing. It will never reach more than 100 steps and on the 100 consecutive episode more than 35.



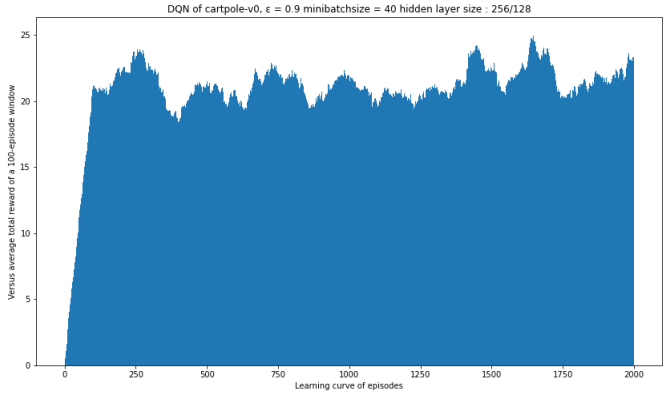
2.4.2 Removing target network

If I remove the target network and instead, my target is the reward if the episode is done and 0 otherwise. It will be able to go up to 200 steps in one episode but it will not learn on the long term, and of course it won't solve the problem.



2.4.3 Removing experience replay and target network

If we remove the experience replay and target network, our model is not learning anymore. It is computing the 2000 episodes very quickly but in the end, it never reaches 200 steps, his final mean is 9.22.



References

https://pylessons.com/CartPole-reinforcement-learning/