NEURAL NETWORKS AND DEEP LEARNING
Homework 5

Laura Iacovissi

September 5, 2020

1 Introduction

In the following report it is described the design process of the `training.py` script as it has been submitted. The aim is exploring the performance of various reinforcement learning algorithms, implemented with different features and structures.

1. In the first section is about the problem settings;
2. in the second section the hyperparameter tuning is described;
3. in the third section the results are presented.

2 Experiment settings

2.1 Environment

In the experiments described in this report, the Environment is a 10×10 lattice in which the Agent can move in order to reach a goal at position $(0, 2)$. The starting point is fixed at each episode at $(8, 8)$. Different obstacles can be defined:

- Walls: used in all the experiments. The Agent is not allowed to choose as next state a position corresponding to a wall;
- Holes: used as additional element. The Agent can choose as next state this kind of position, but it will result in having an action reward of -10 and the impossibility of exit from the hole;
- Spikes: used as additional element. The Agent can choose as next state this kind of position, having an action reward of -0.5 and the possibility of exit.

Other rules are: the maximum reward (equal to 1) is assigned when the Agent reaches and stays in the goal point; null reward is assigned when the Agent moves to a neutral position (i.e. no obstacles, no goal, no boundaries).

2.2 Agent

The Agent used for these experiments is able only to move horizontally or stay in its current position. The decision for the next action to do is made according to some algorithm and a behavior policy. Here the algorithm is chosen between *Q-learning* and *SARSA*; the behavior policy is the ε-*greedy* or the *softmax* one.

Each of these possible combinations tries to maximize the overall reward in a greedy way, because of the complexity of the exact problem. Since the behaviour policy are stochastic, the results will not be deterministic. A seed will be set for reproducibility.

λ	$\{0.1, 0.2, 0.3, 0.4, 0.5\}$
γ	$\{0.6, 0.9, 0.95\}$
ε decay	hyperbolic, from 1 to 0.01
episodes	4000
movements	50

Table 1: Algorithm parameters.

3 Hyperparameters

Some parameters have been fixed after some preliminary tests, in particular we have the number of training episodes set to 4000, the episode length (i.e. number of Agent movements for each episode) equal to 50 and the ε behaviour fixed as hyperbolic decay (from 1 to 0.01).

The discount factor γ is varied in the set $\{0.6, 0.9, 0.95\}$ in order to have an idea of how this parameter influences the convergence of the algorithm. For the same reason, the learning rate λ is varied in $\{0.1, 0.2, 0.3, 0.4, 0.5\}$.

The parameters are resumed in Tab. 1. They will be used for all the combinations of algorithm listed in the Agent description paragraph.

4 Results

Here the results are analysed in detail, in order to identify the most convenient algorithm, behaviour and parameter combination.

4.1 Only walls

In this simple scenario the performances under different learning rates and discount factors have been explored.

As regards the learning rate λ, it has been fixed to 0.5: as it can be noticed from Fig. 1, for this value of λ the convergence is faster and slightly more regular. The discount factor γ seems to have no influence on the reward behaviour when using Q-learning and ε-greedy, at least for this level of complexity of the game. For the other combinations, $\gamma = 0.6$ does not allow the algorithm to learn how to reach the goal in a stable way.

In general, almost all the combinations of algorithms have the same behaviour, i.e. follow the optimal path to the goal (see animated images in the GIFS directory). However, the faster algorithm seems to be the combination of SARSA and softmax policy with $\gamma = 0.95$.

4.2 With spikes

The second test has been performed by adding to the default environment some cells with spikes.

Looking at Fig. 2, we can notice that the reward behaviour in the worst (Q-learning + ε-greedy) and best (SARSA + softmax) cases show features similar to the previous experiment:

effect of the γ value only in Fig. 2(a), bad results in Fig. 2(b) for $\gamma = 0.6$, faster convergence for $\gamma = 0.95$. The main differences are: the maximum reward reached, which is smaller that before since the algorithm takes more risks; the poorer results for SARSA and softmax policy with $\gamma = 0.6$, which has less deviations from the zero reward, i.e. in less cases it manages to arrive to the goal.

Again, the combination with higher performances is SARSA and softmax policy with $\gamma = 0.95$.

4.3 With holes, with spikes and holes

The third simulation has only holes as additional obstacles.

Since it is the element with the harder penalization value, the rewards shown in Fig. 3 have stronger negative oscillations. The ε-greedy policy is not able to make the average reward converge even in 4000 episodes, even if combined with SARSA. On the other hand, the softmax policy highly stabilize the reward curve when $\gamma > 0.6$.

The faster convergence is observed again for SARSA and softmax policy with $\gamma = 0.95$.

The last environment has all the obstacles defined in the Experiment settings paragraph: holes and spikes.

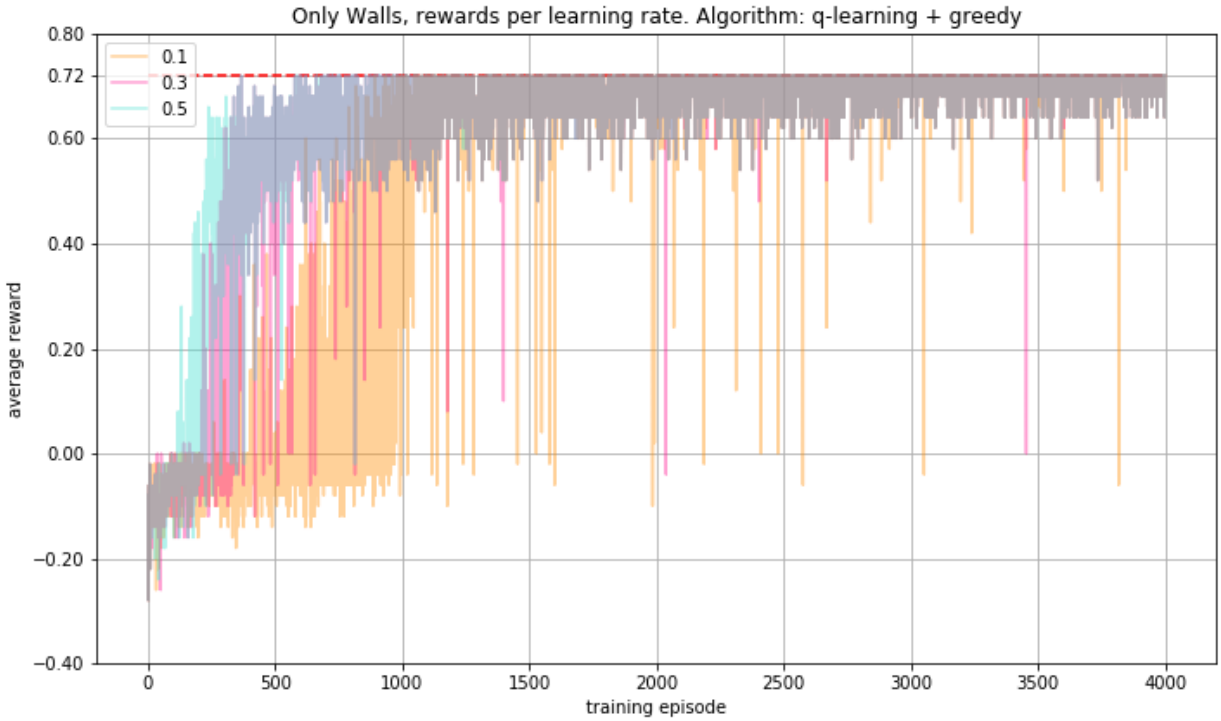
Apparently the element that influences more the reward behaviour is the presence of holes in the grid: the plots in Fig. 4 are almost equal to the one referring to the previous experiment. It was of course expected because of the definition of the hole, from which it is impossible to exit when chosen as position of the path.

The faster convergence is observed again for SARSA and softmax policy with $\gamma = 0.95$.

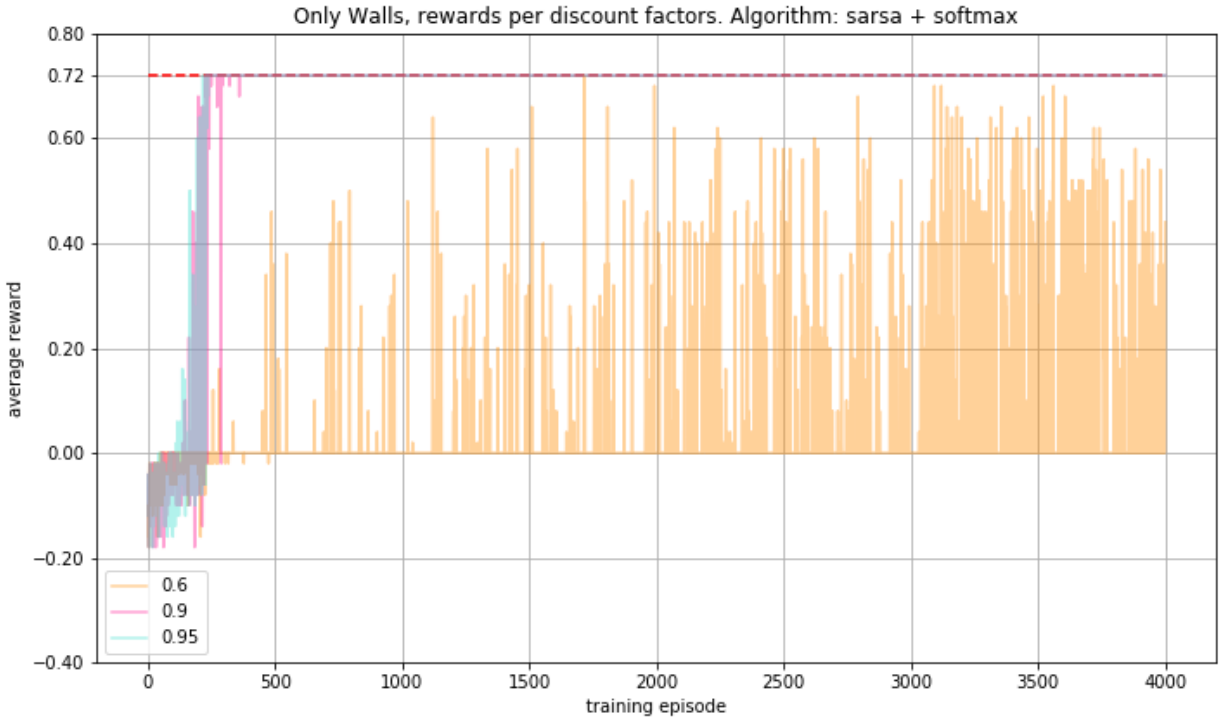
5 Conclusion

In all the experiments performed, the performance are maximized when the SARSA + softmax algorithm is employed. Here the higher value of γ used is 0.95, which is always the selected one in order to have a faster converge. It is probable that for γ's closer to one the speed of convergence would be even higher.

A Appendix: Images

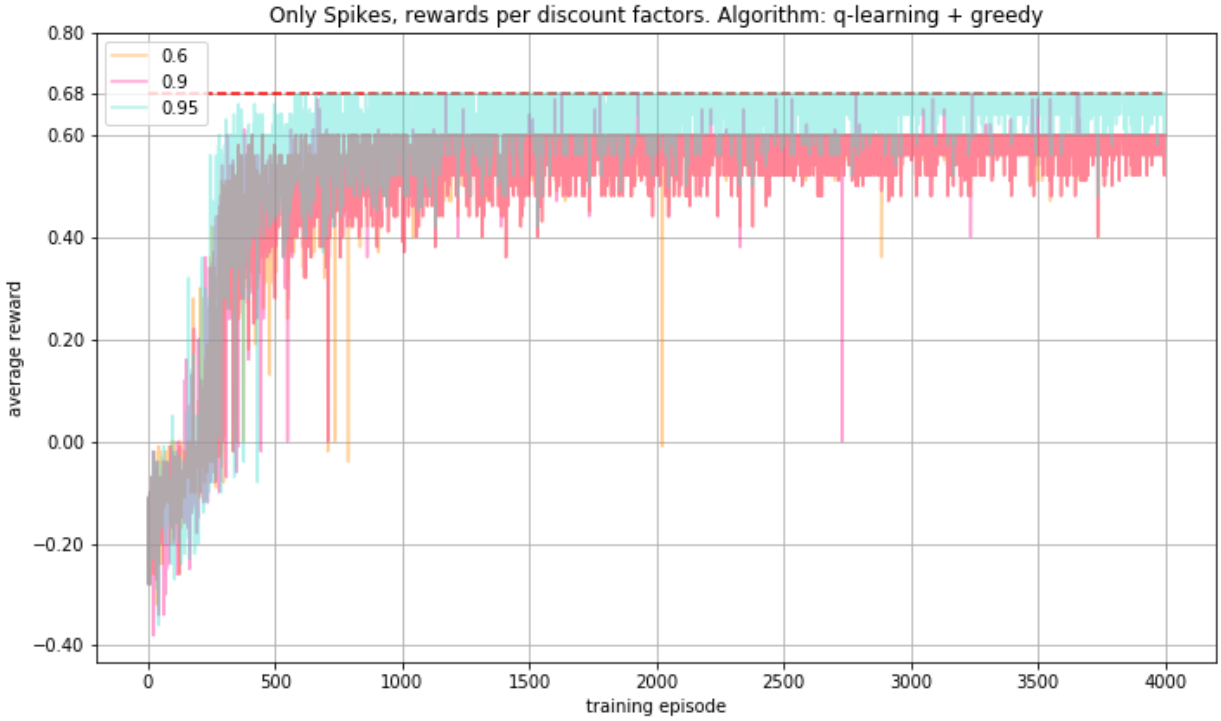


(a) Average reward for different values of λ, $\gamma = 0.9$

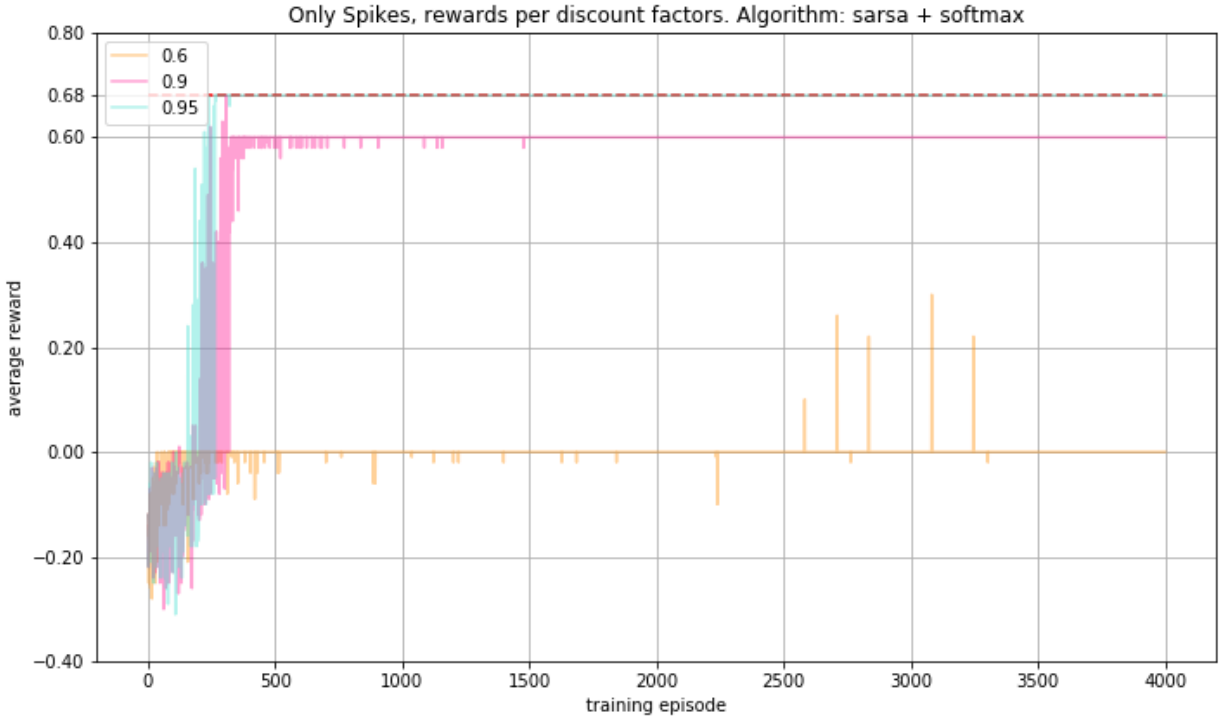


(b) Average reward for different values of γ, $\lambda = 0.5$

Figure 1

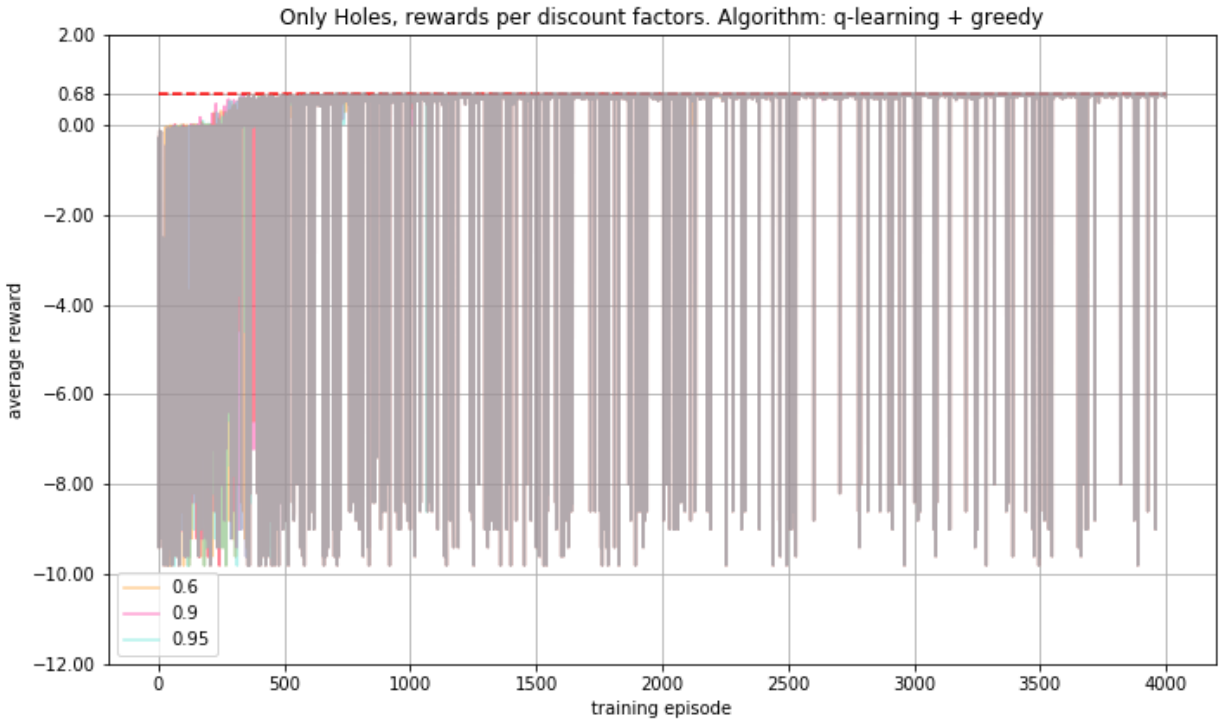


(a) Average reward for different values of γ, $\lambda = 0.5$

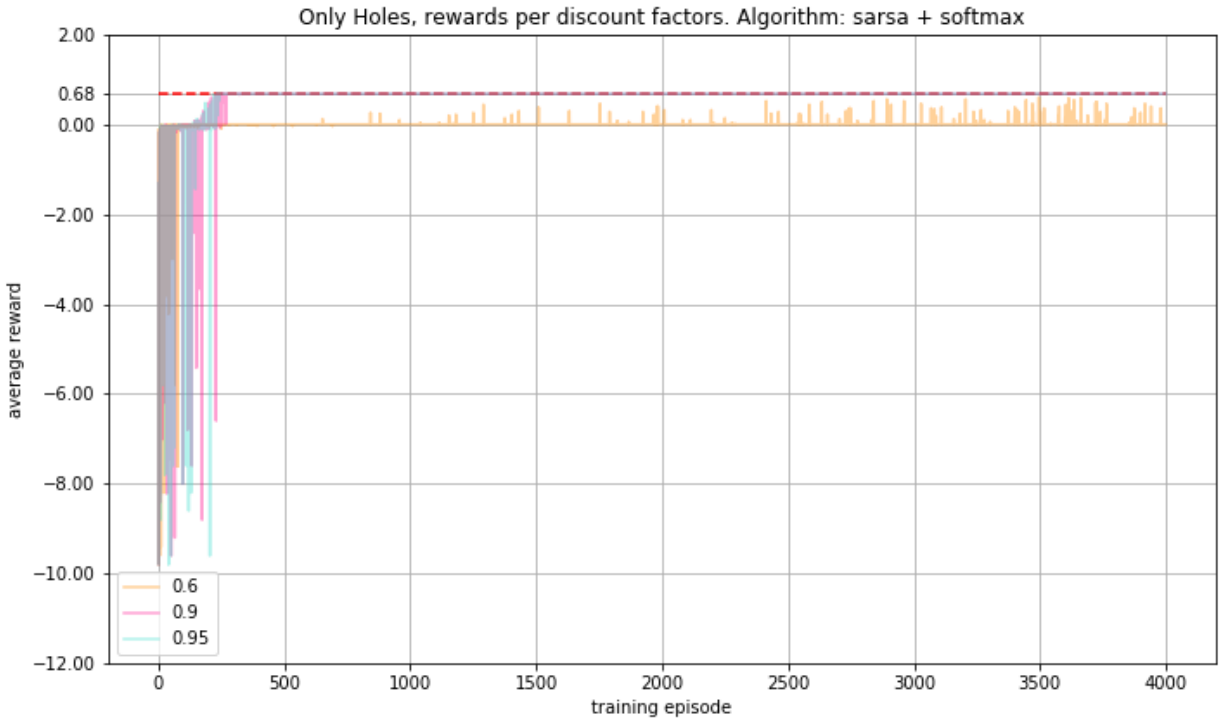


(b) Average reward for different values of γ, $\lambda = 0.5$

Figure 2

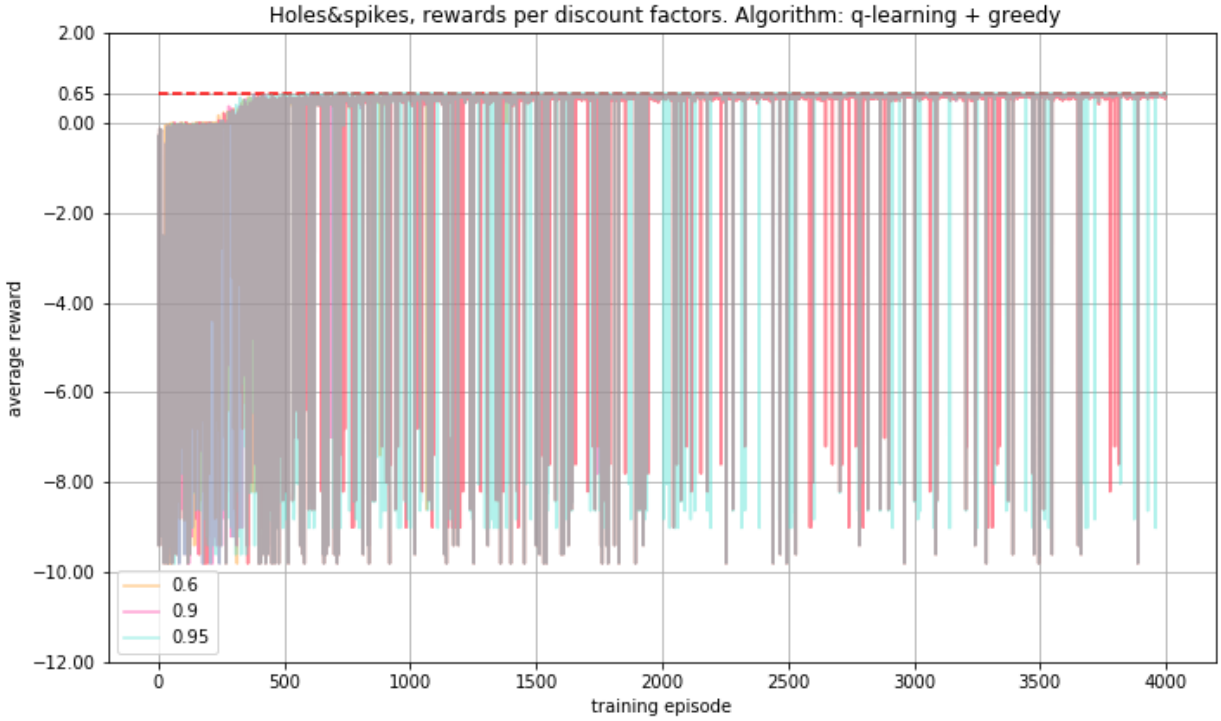


(a) Average reward for different values of γ, $\lambda = 0.5$

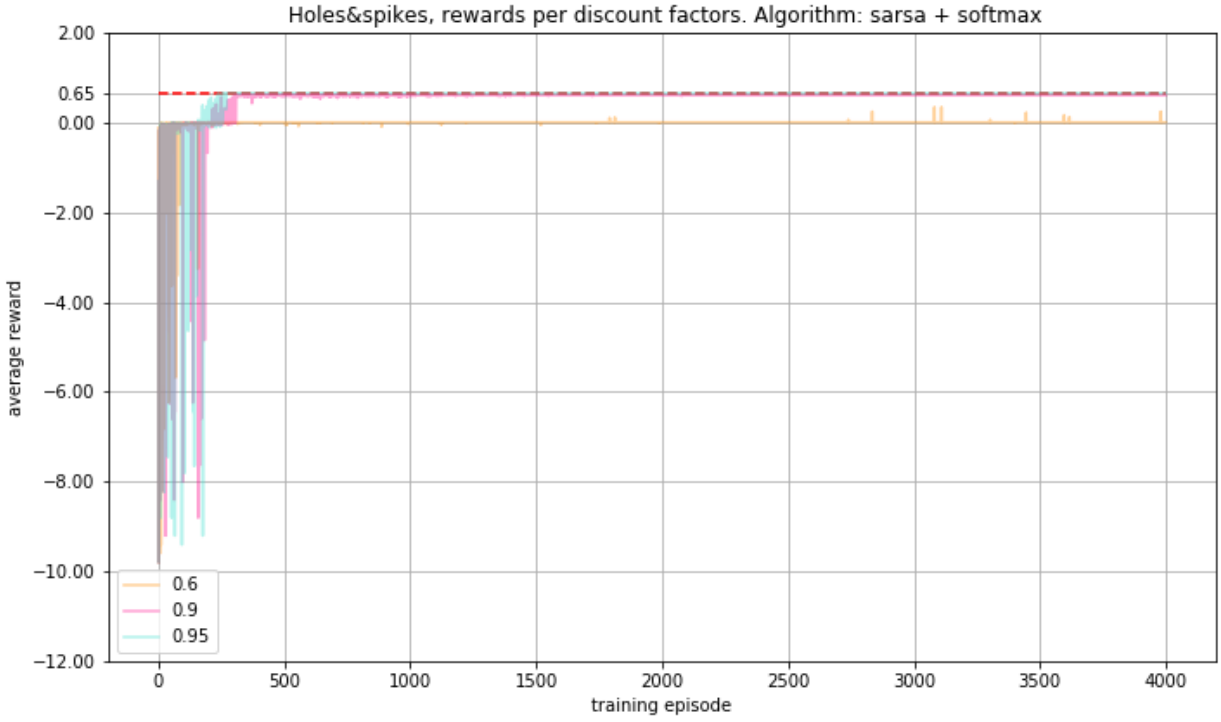


(b) Average reward for different values of γ, $\lambda = 0.5$

Figure 3



(a) Average reward for different values of γ, $\lambda = 0.5$



(b) Average reward for different values of γ, $\lambda = 0.5$

Figure 4