Reinforcement Learning Project Report

Unsupervised and Reinforcement Learning in Neural Networks

Han JU, Hao REN

19 décembre 2012

1 Introduction

In this project we implement a reinforcement learning experiment with continuous state space and a neural network model. Unlike the discrete state learning experiments, in a continuous state space we can't enumerate over all possible states, that makes the classic Q-learning algorithm not applicable here. Instead, with the help of a neural network model taught in the course, we are able to code the state space by a finite number of input neurons, namely the place cells. On the other side, we have the action cells, each represents an possible action of the agent in the experiment. Given this set up, the expected reward for a particular point in the state space is just the dot product of the point's activation of all place cells and the weight vector between the place cells and the action cells. So essentially we use the classic SARSA algorithm's framework, but what we update is the weight vector, and the Q-values are computed dynamically for choosing an action.

For the implementation, we adapt it from the given python code in the exercise session.

2 Learning curve

Right after the implementation according to the project set-up, we begin our experiment by conducting 10 independent runs with 50 trials each. Figure 1 is the resulting learning curve.

For this very first experiment, we set the epsilon parameter to 0.5, which means half of agent's moves are random. The leaning curve makes perfect sense : the first several runs are nearly random walks, which take over 5000 steps thanks to the randomness and the yet not very effective W-values. However, after several tastes of the goal, due to the effect of the eligibility trace, this information is quickly propagated throughout the weight vectors between the input space cells and the action neurons. As a result, half of the time the agent chooses the right direction that leads it to the goal area and avoids the wall hitting, the steps needed are decreasing drastically. We can observe that within 10 trials, this number goes down from 8000 to a stable value of aprroximately 100. Note that because of the randomness, there is some fluctuation on the curve.

As for the average reward, showed in red curve in the figure, it has generally the same trend as the latency curve. For example in the first several trials, the agent is essentially exploring the state space, so it hits often the wall and that causes a very low overall reward. As the agent's knowledge of the state space grows, its reward per trial converges to 10, which means it pursues directly the goal area without hitting too much the wall.

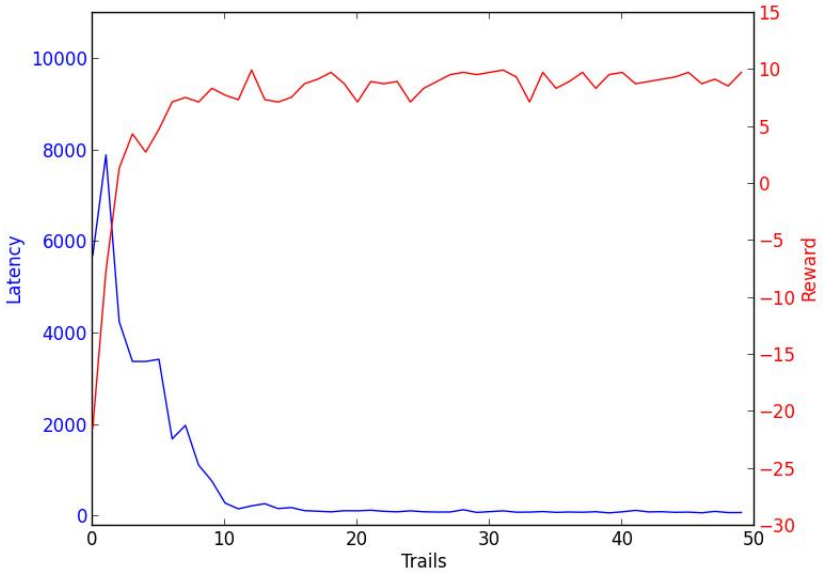

FIGURE 1 – Learning curve of a complete experiment

3 Exploration & Exploitation

The parameter ϵ controls the balance between exploration and exploitation. When ϵ is big, the agent has more possibilities to take a random step in order to explore more place in the environement, while small ϵ makes the agent more likely to take a step according to the ϵ-greedy policy. The implementation of the algorithm SARAR depends on the choice of the ϵ, we have to make a tradeoff. If the ϵ is too big, the agent will seldom take a step with the greedy policy, the whole learning process is more stochastic, so the learning result is not significant. However, if ϵ is too small, the agent will be more likely to exploit the state around where it is, because the weights are not largely updated, the agent sometimes will travel around several same states, sometimes even not get to the goal within a given time step limit.

The best solution is to start with a big ϵ so as to explore the space as widely as possible. The ϵ decreases after each trails Thus, the first several trials with big ϵ will collect enough information about the weights (Q values), and the later trials with small ϵ can take a good use of these information via the greedy policy which makes the agent more likely to get into the goal area.

The learning curves and the average latency of the last 10 trails with ϵ=0.8, 0.6, 0.4, 0.2 are shown as below :

$\epsilon = 0.8 \Rightarrow$ 219 steps

$\epsilon = 0.6 \Rightarrow$ 113 steps

$\epsilon = 0.4 \Rightarrow$ 77 steps

$\epsilon = 0.2 \Rightarrow$ 79 steps

The result is exactly what we expected. When ϵ equals to 0.8, the agent quickly finish the first trail (2000 steps average), but it takes more steps in the end, due to the more random choice of the next step. For the small ones, it takes 9000 steps to get to the goal at the begining, but the final averge step is less.

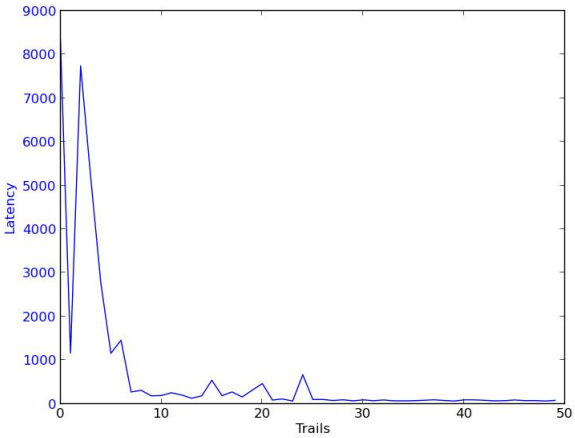

FIGURE 2 – $\epsilon = 0.2$

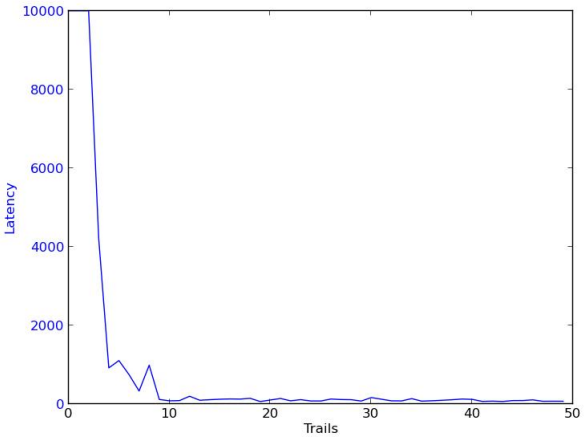

FIGURE 3 – $\epsilon = 0.4$

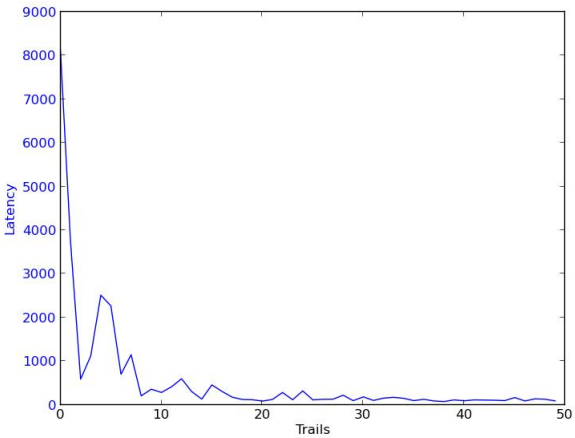

FIGURE 4 – $\epsilon = 0.6$

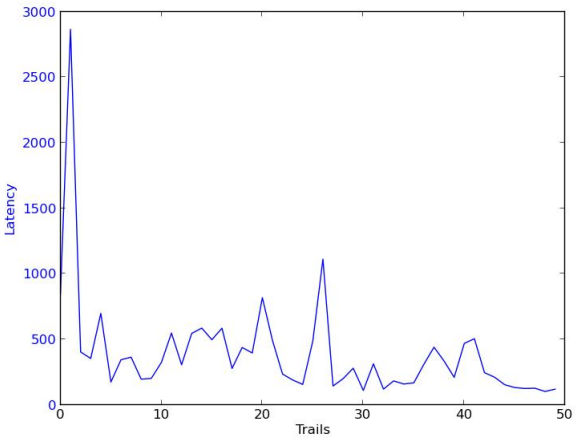

FIGURE 5 – $\epsilon = 0.8$

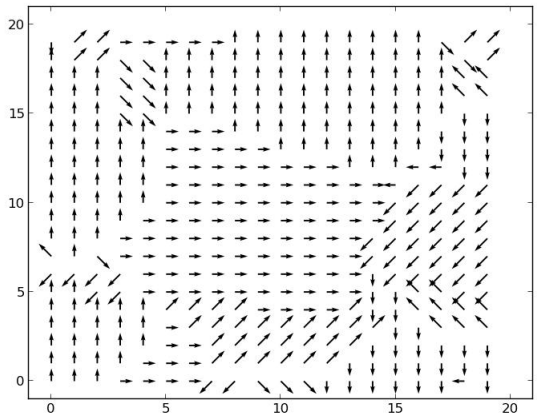

FIGURE 6 – Navigation map after 10 steps

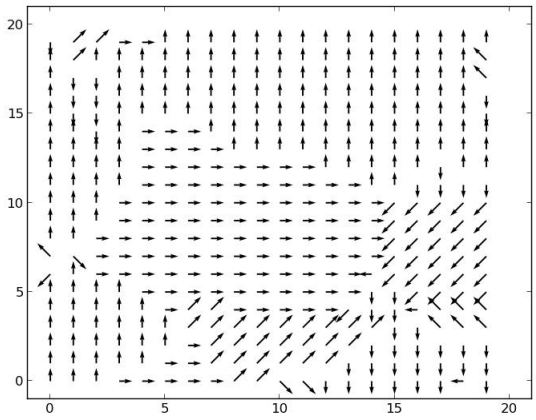

FIGURE 7 – Navigation map after 30 steps

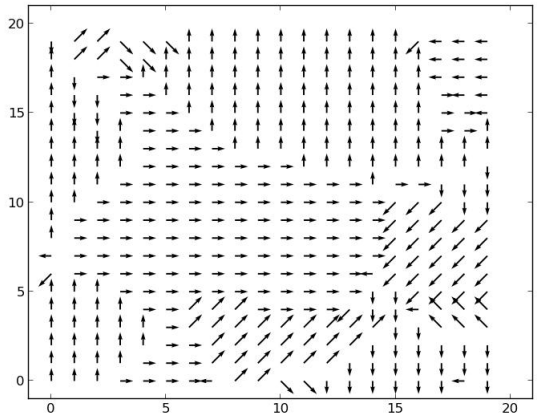

FIGURE 8 – Navigation map after 50 steps

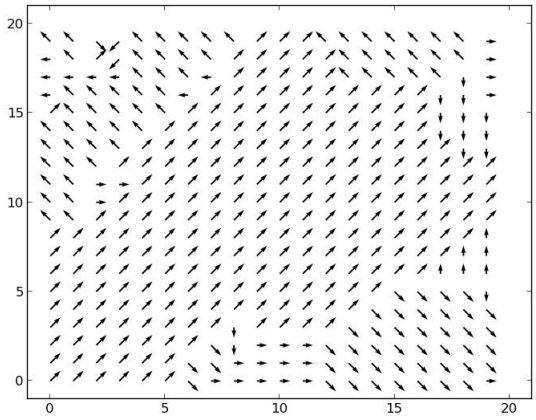

FIGURE 9 – Best navigation map

4 Navigation Map

The naviagation map is shown as below :