Unsupervised and reinforcement learning in neural networks
Project 2
Reinforcement learning using a rate-based neuron model

Joachim Muth and Sebastien Speierer

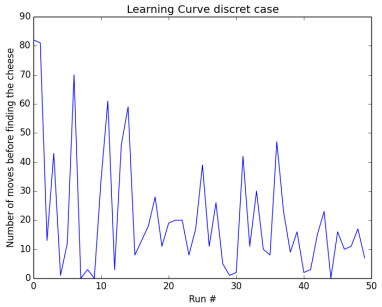
December 31, 2015

Project description

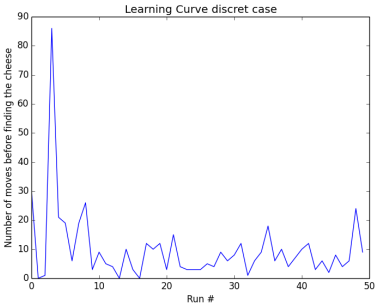
For this project, we're given a code implementing a reinforcement learning paradigm of a rat placed in an arena learning the position of a cheese (its goal). The arena is a discrete grid and the rat is moving step by step. Our goal is to implement a reinforcement learning algorithm using a rate-based neuron model on the same setup such that the position and movement of the rat is not dependent of the grid anymore. In other words, the arena is now a continuous world and the cheese will be represented as a circle area in the arena.

1 Initial discrete-world simulation

In the initial simulation we tests the behavior of a point-like rat in a 4x4 square world where the rat begin in (0,1) and the cheese is in (0,0). The basic neural network is already implemented and we can illustrate its performance by showing the latencies curve without learning and with it, as show on Fig 1



(a) Without learning (Q's are reset each iteration)



(b) With learning, well illustrated by the decreasing latency before finding the cheese.

Figure 1: Learning curve in the discrete-world model of a 4x4 playground

2 Continuous-world simulation

We implement the new model providing a continuous-1-square-world simulation in which our rat is still a point but the cheese is represented by a 0.1-radius circle placed in (0.8, 0.8). The FIG 2 show us the general display of the situation and illustrate the paths taken by the rats at some run. The FIG 3 below represent the state of the rat's neurons, and display the favorite direction of each of them.

2.1 Learning curve

The learning of the rat can be represented by the number of step needed to reach the goal. In the FIG 4 we observe the decreasing number of moves made in each run of his training. Some observations must be made:

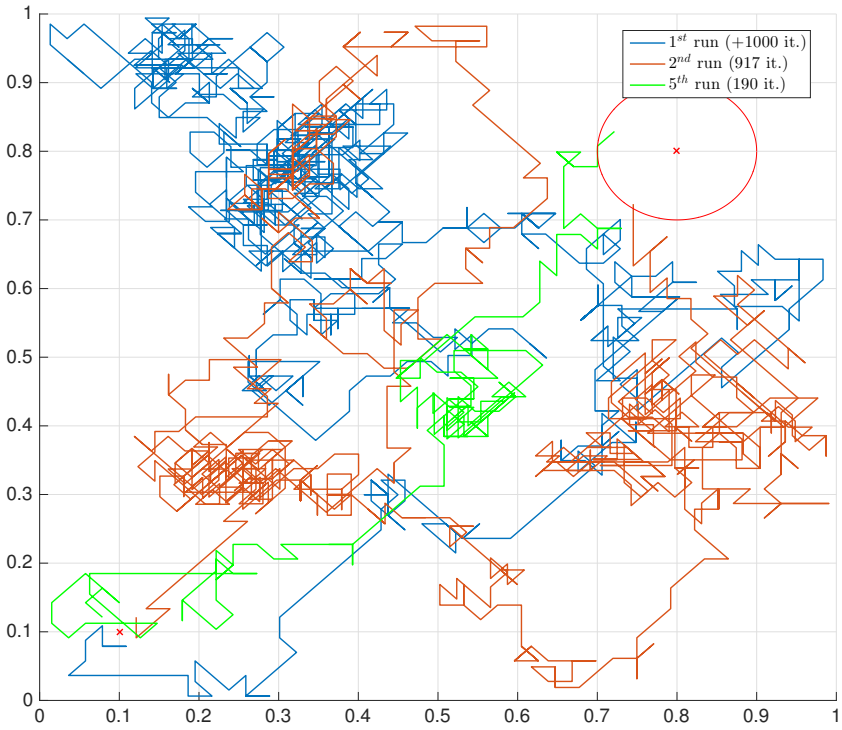


Figure 2: Paths taken by the rat in different run. The first path (blue) did not reach the cheese before the timeout. The second one reach the cheese and learn to the rat where is it. The fifth path is already quite good, taking the optimum direction but having some difficulties in the middle.

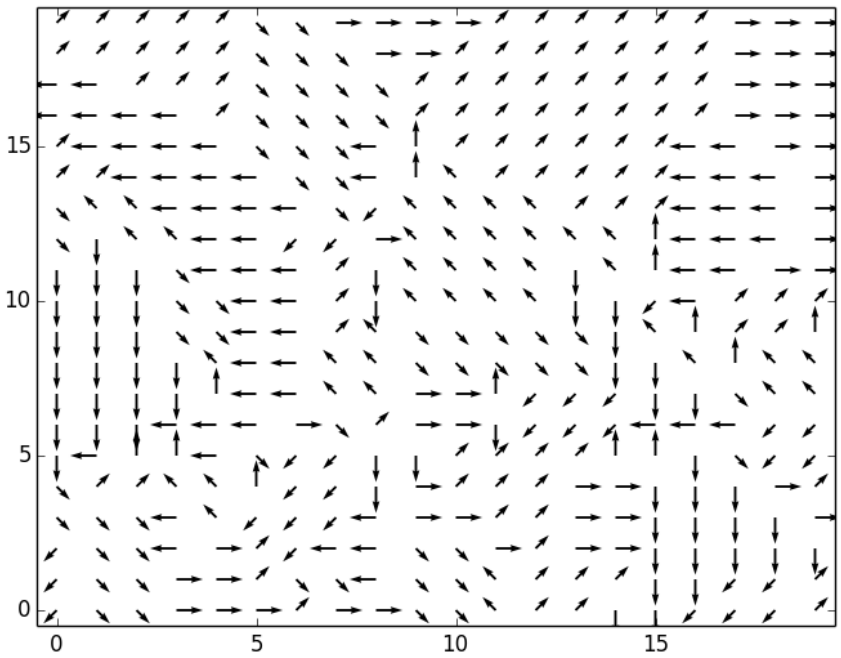


Figure 3: Neural map of the final state of the neurons of the rat.

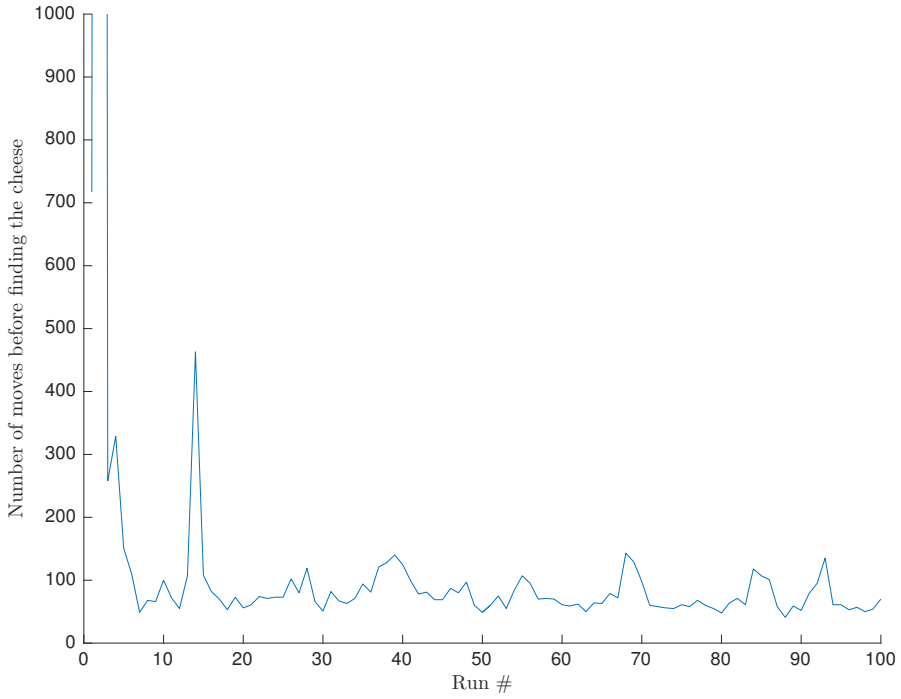


Figure 4: Learning curve of the rat in continuous-world simulation

First of all it exists a limit above which the run is aborted (in this case 10'000 iterations). The first pics illustrate this case. In the third iteration the rat reach the cheese in approximately 300 moves and then we observe the number decrease.

The second observation is about the other pics, such as the 12^{th} run of the 40^{th} one. What are their and why the curve doesn't just decrease? It's due to the epsilon-greedy strategy that we will talk about later. Some randomness is introduced in the model in order to make the rat explore the map and this can have bad impact such as "let's explore this apparently bad path".

Note that the optimal theoretical path should take around 20 moves and the optimal path found by the rat stagnate at 40 moves.

2.2 Integrated reward

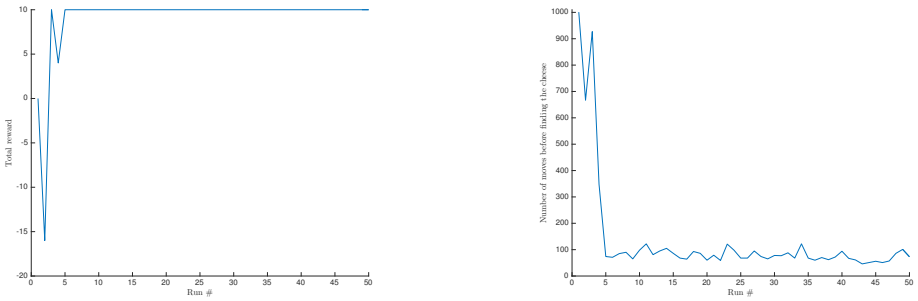
Is the reward correlated with the latency of the run? The answer should be yes, because the only information received by the rat is whether he gains or looses rewards. The FIG 5 illustrate both of the interpretations and shows that when the rat reaches the maximum reward (cheese without touching wall) he stops improving himself.

Thanks to the Neighborhood Gaussian treatment, the rewards are kind of "extended" over the map which incites the rat to minimize the length of his path.

2.3 Exploration-exploitation

The strategy of implementing a random factor inside the algorithm is called *epsilon-greedy*. It balance the behavior of the rat between exploration of new path and exploitation of previous knowledge.

- exploration: with a probability of ϵ the rat will take a random decision about the direction to choose. This allow to 1) discover new path, maybe better than the previous known. 2) not commit always the same errors.
- exploitation: with a probability of $1 - \epsilon$ the rat will follow the best known decision.

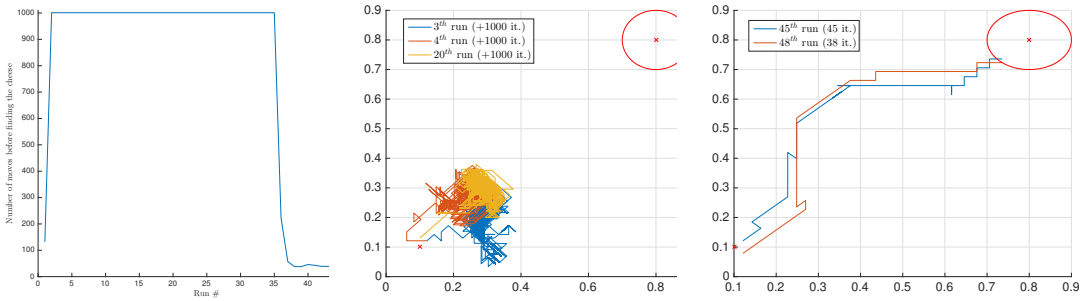


(a) Total reward of each run (b) Latency of each run

Figure 5: Final reward compared to latencies

2.3.1 Effect of a bad epsilon-greedy choice

It seems legit to ask if the epsilon-greedy strategy is really useful or not. Here we illustrate what can happen if we choose a too low random rate, for example $\epsilon = 0.1$. The main problem as illustrated in FIG 6 occurs when the rat didn't reach the goal and continuously repeats the same error again and again without improvement. In the FIG 6 we show that the first 40 run are useless because it just turn around. However, the last paths seem good and the rat just tries some little variation, without loosing himself.



(a) Learning curve (b) Repeated wrong paths (c) Repeated correct paths

Figure 6: Analyze of rat behavior with too low random rate $\epsilon = 0.1$

We reach at the following conclusion: The set up of a good neural network need a good balance between exploration and exploitation, above all at the begin of the learning, when the rat doesn't know anything about the playground. The optimal way is to begin with a high random rate and then gently decrease it as the rat learn better the playground, to avoid choosing a path already known as bad.

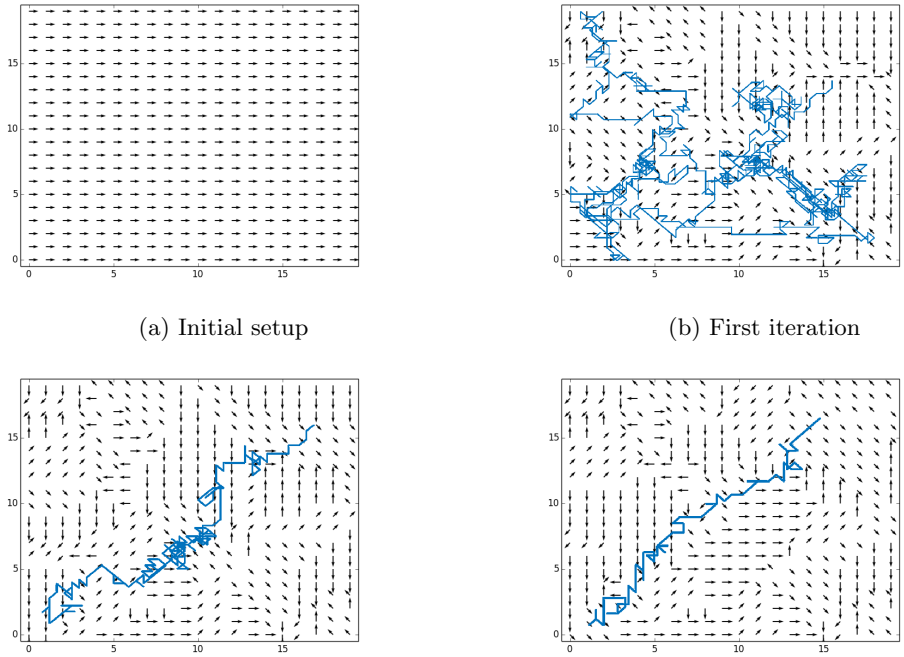
```
for trial in range(N_trials):
    # run a trial and store the time it takes to the target
    ...
    # decrease the epsilon-gready
    self.epsilon = max(0.996 * self.epsilon, 0.001)
```

2.4 Navigation map

The FIG 7 illustrates the evolution of the neural navigation map that takes form over the iterations.

Conclusion

The correct set up of each parameter inside a neural network is pretty hard and mostly empirical. The learning rate, the size of the Gaussian function, the ϵ random factor were chosen after plenty of simulations, and the



(a) Initial setup

(b) First iteration

(c) Second iteration

(d) 50^{th} and last iteration

Figure 7: Illustration of the neurons navigation map of the rat and the path chosen by him.

modification of any of them can corrupt the whole algorithm. It's one of the limitation of the neural network model.

Despite of all these constraints we reach to develop an efficient neural network in continuous-world simulation in which our rat needs only 5-10 tries to find a good path to the cheese.