Mini Project 2

Wei Ma
ma.wei@epfl.ch

Yuliang Zheng
yuliang.zheng@epfl.ch

January 2016

Abstract

In this report, we summarize the findings for the mini project 2. We implement a reinforcement learning paradigm using a rate-based neuron model, ultilizing SARSA(λ) algorithm in the continuous space. The map is an 1×1 area which x and y starts from the original point. The map is divided into 20×20 cells. We use Gaussian function to compute the activity of one cell. The eligibility trace is used to record the history memory. In the lab, we tried to definite how some certain parameter's changes effect the result.

1 Problem Description

A rat is put into a rectangular arena with unit area, it searches for a reward(+10) which is inside a small goal circle(centered at (0.8, 0.8), with radius 0.1). The rat starts at position (0.1, 0.1), if it hits the wall, it will receive a -2 reward and be placed back inside the arena. Once the rat reaches the goal circle to get the reward, or when the number of steps taken exceeds a setting constant, the trial will stop. In this report, we utilized reinforcement learning with a rate-based neuron model to solve the problem. And the on-policy TD control method – SARSA(λ) is the algorithm we chose.

2 Implementation

To implement SARSA(λ) algorithm, what we have done is shown in the flow graph of Figure 1. However, we will offer some detailed explanations below. Before choosing the first action, we are required to do initialization. In the Matlab, we created a struct "gg" containing all the needing information such as the rat's state position, eligibility trace, etc. We set the grid weights randomly and then normalize the vector weights, $weights = \frac{weights}{weights^T * weights}$, all the eligibility traces being zero and all other parameters as required. Put the rat at the starting point (0.1, 0.1), and then we can begin the trial.

Using $r(s) = exp(-\frac{(x-s_x)^2+(y-s_y)^2}{2\delta^2})$ to compute r first, and then using $Q(s,a) = \sum_j w_{aj} r_i(s)$ to get 8 action's Q values. r is the activity of the center cells. Based on the requirement, the selection of action is approached by the use of ε-greedy policy. Here, we did some tricks during the implementation: creating a random number(between 0 and 1), if it is bigger than $1-\varepsilon$, we select the action with biggest Q ($a* = argmax_a Q(a,s)$); otherwise we randomly choose one of the eight direction's actions. Taking the chosen action, it's easy to compute the next state position. As there are 8 directions with step length $l = 0.03$, we created a function named "goadvance" with 8 direction's actions, every two directions of 45° angular difference. Once getting the next state position, it's necessary to make two judgements. First, if the rat now is outside the arena, we put it back to its previous position and add a -2 reward(R) to it. Second, if the rat arrives inside the goal circle, we give the it a +10 reward. When the state is outside the arena, a trick about adopting which action is used. When the rat hits the wall, it will change its action of the vertical direction that is opposite to the obstacle wall.

Later, it's time to do some updatings. According to the ε-greedy policy, choose the next action(a') in the new state(s'). And then update the following parameters step by step:

1) Calculate TD error: $\delta = R - [Q(s,a) - \eta Q(s',a')]$

2) Update eligibility trace: $e_{aj}(t) = \begin{cases} \eta\lambda e_{aj}(t-\Delta t) + r_j & \text{if a = action taken} \\ \eta\lambda e_{aj}(t-\Delta t) & \text{otherwise} \end{cases}$

3) Update weights: $\Delta w_{aj} = \eta\delta e_{aj}$

After updating the above three parameters, we need to update action and state as well, which means we reset a' as a, s' as s. If the steps taken exceed the maximum or the rat already reaches the goal circle, the trail ends. Otherwise, it will continue to update.

After the implementation, we ran the corresponding Matlab prgram. Figure 6 gives the route of one rat in 50 trials, and the red circle is the goal area. It can be found that as the number of trials increases, the rat takes less steps to reach the goal circle, which illustrates a shorter and more direct line from the starting point to the red circle.

3 Analysis

Once we successfully implemented the environment and the neural network that controls the movement of the rat in the arena, we changed some of the parameters and did some analysis. To get a better result, we set the ε to be 0.95 from now.

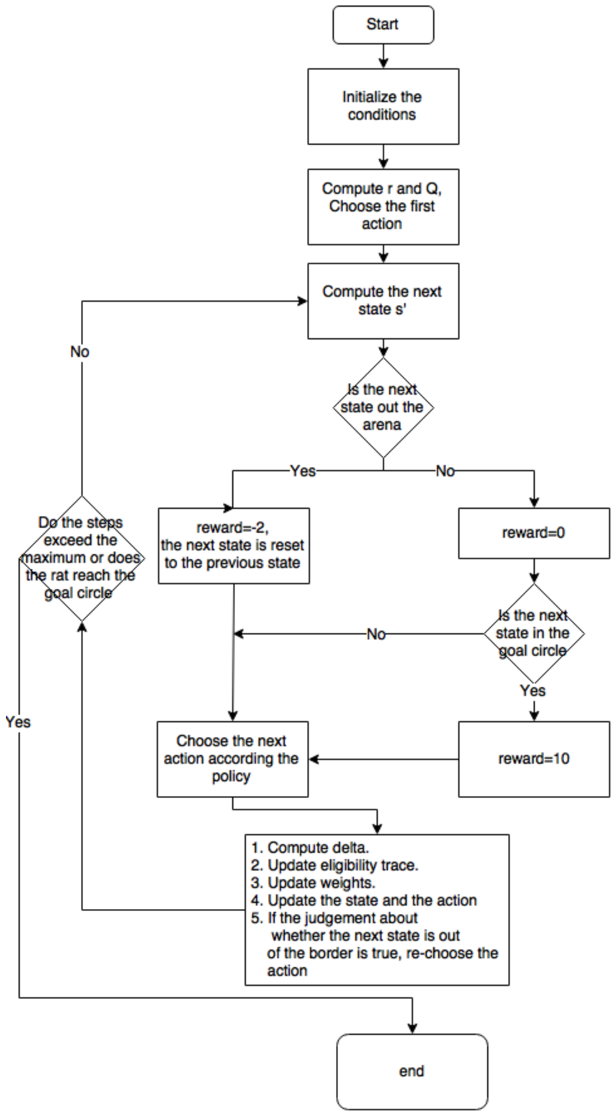
Start
Initialize the conditions
Compute r and Q, Choose the first action
Compute the next state s'
Is the next state out the arena
Yes
No
reward=-2, the next state is reset to the previous state
reward=0
Is the next state in the goal circle
No
Yes
reward=10
Choose the next action according the policy
Do the steps exceed the maximum or does the rat reach the goal circle
No
Yes
1. Compute delta.
2. Update eligibility trace.
3. Update weights.
4. Update the state and the action
5. If the judgement about whether the next state is out of the border is true, re-choose the action
end

Figure 1: The flow graph of the program

3.1 Learning curve

We simulated independent 10 rats that run 50 trials each, and set the maximum number of steps taken(N_{max}) to be 10000. Plot the learning curve, shown in Figure 2. In the left figure, each color line represents one rat's movement. It's obvious to find that, from the beginning every rat reached the goal circle and they took less and less steps in the other subsequent trails. The right one offers a more intuitive result, using the boxplot to illustrate the range of the 10 rats' number of steps taken in each trial, and using each trial's median step number to plot the red curve, the curve decreases and reaches a plateau(a very tiny number, we got a result around 40 steps) after a certain number of trials.

3.2 Integrated reward

When looking at the total reward that was received on each trial, we got the integrated reward curve, shown in Figure 3. The same as before, each color line represents one rat's movement. It's easy to obtain the conclusion that the total reward in a single trial increases from the initial negative number to approximately 0 in the end as the number of trials augmented. So, the result is consistent with the latency curve.

3.3 Exploration-exploitation

One of the challenges that arise in reinforcement learning is the trade-off between exploration and exploitation. To obtain a lot of reward, a reinforcement learning agent must prefer actions that

it has tried in the past and found to be effective in
producing reward. But to discover such actions, it
has to try actions that it has not selected before.
The agent has to exploit what it already knows in
order to obtain reward, but it also has to explore
in order to make better action selections in the future.[1]

Because based on the ε-greedy policy, the parameter ε determines the probability of choosing a certain action with the largest Q value or a random action among the 8 candidates, it controls the balance between exploration and exploitation. To see how the different values of $0 \leq \varepsilon \leq 1$ change the performance, we plot both the the several learning curves for different ε values and using the average latency in the last 10 trials as a performance measure, the result is shown in Figure 5. The same as before, each color line in the left one of Figure 5 represents one rat's movement. Compared this figure to the right one, we can conclude that when the ε decreases from 1 to 0, the number of steps taken being very large at $\varepsilon = 1$ and then to be a low value for a while, and then increases again. In a nutshell, the two endpoints $\varepsilon = 1$ and $\varepsilon = 0$ show the worst result, which means full exploration or exploitation will do bad effect. To get a good performance, we set $\varepsilon = 0.95$ in this report.

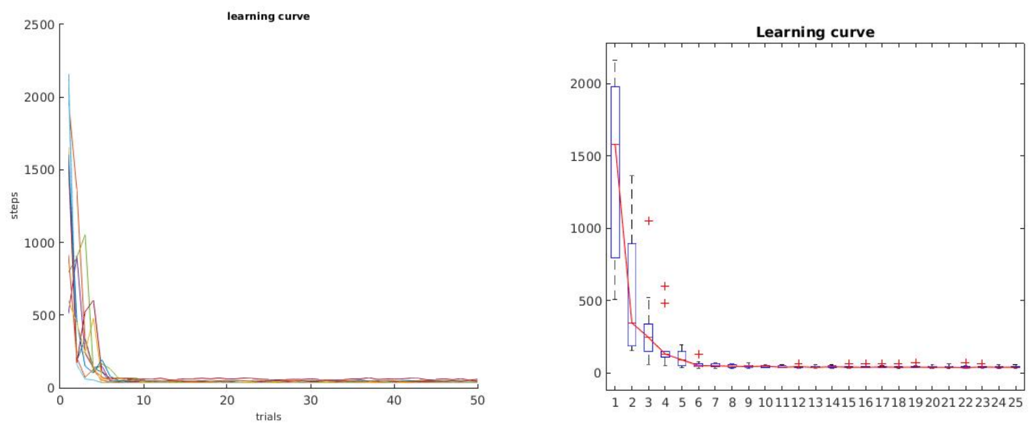

Figure 2: Learning Curve

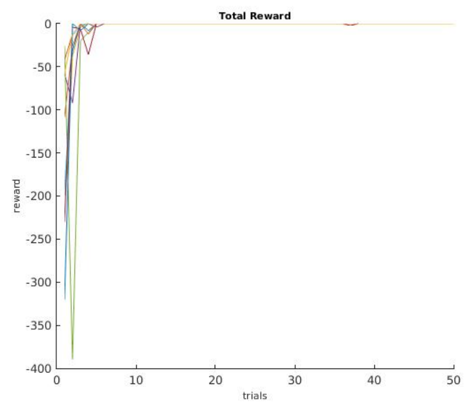

Figure 3. Integrated Reward Curve

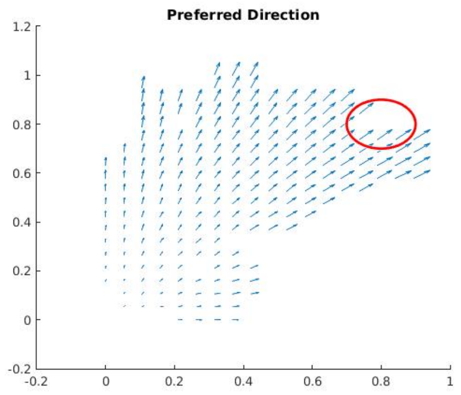

Figure 4. Preferred Directions

3.4 Navigation map

The Figure 4 demonstrates the preferred direction at each place field center. The preferred direction of the center is the direction which has the highest probability being taken. The blank blocks are the

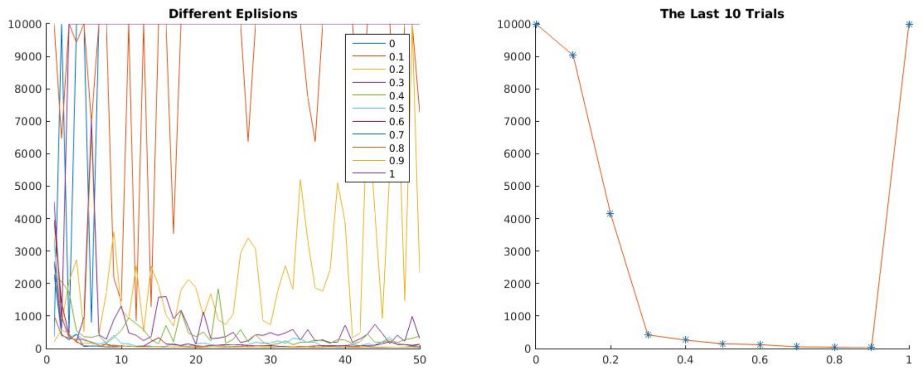

Figure 5: Exploration-exploitation Curve

areas that the rat didn't reach and the preferred direction is missing there. As a result, we can see that the preferred directions which aren't on the marginal have a higher preference to point to the goal circle.

4 Additional Observations

In this lab, we find that the initialized conditions are non-trivial. If the initialization goes wrong, the algorithm will perform badly. For example, the matrix of eligibility trace must be initialized to zero. It records the history memory so that it must be zero before the algorithm starts. Another example is the initialization of the weights. If the weights are not normalized, the algorithm sometimes will behaviour bad. The number of the steps and the total reward at each trail will converge. Besides, the algorithm will perform much worse if we utilize full exploration or full exploitation.

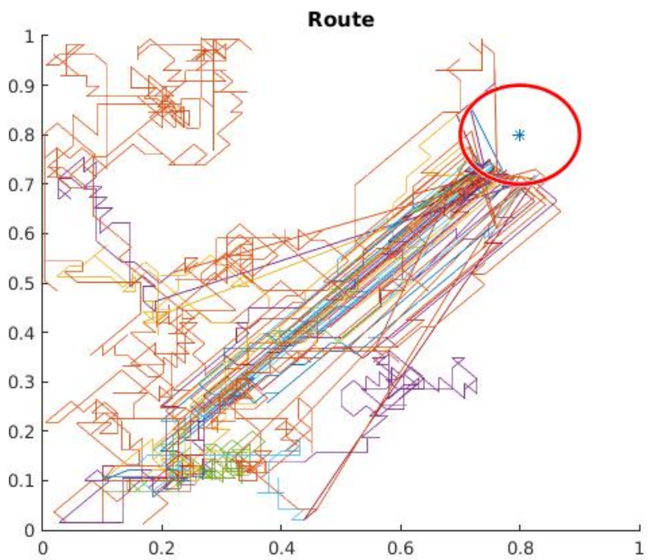

Figure 6: Route

References

[1] Richard S. Sutton and Andrew G. Barto: Reinforcement Learning: An Introduction