Train a Smartcab to Drive

1. **Implement a basic driving agent**

```
self.listOfActions = ["forward", "right", "left", None]
…
action = random.choice(self.listOfActions)
```

In this step the algorithm generates a random action without considering the received rewards or previous experience.

By observing the agent behavior, it is clear that it will only reach the goal by chance and after a long period of time. Moreover, the agent does not follow any traffic rules, thus obtaining penalties for most of its actions.

2. **Identify and update state**

I defined the state as the vector:

```
["light", "left", "oncoming", "next_waypoint"]
```

I do not consider the input information about the cars located to the “right”, given that this information is not relevant to the agent as long as it respects the traffic lights.

I decided not to include the “Deadline” counter because I believe that an optimal agent should not ignore the traffic rules according to how much time does it have left. Moreover, having an additional integer value in our state representation would scale the amount of possible states considerably.

Finally, having the optimal “next_waypoint” in our state representation should give the agent enough information regarding what it needs to do. Using this state representation I expect the agent to take the action given by “next_waypoint” every time the traffic rules allow it, or to stay put otherwise.

3. **Implement Q-Learning**

I implemented Q-Learning with α and γ equal to 0,8. The agent greedily selects the best action according to the Q values:

$$action = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

After running 100 trials I observed that, although the agent learns good policies for some states, the majority of the state-action pairs are never updated; it is necessary, therefore, to introduce some exploration component into the action selection process.

After observing the agent behavior during the simulation I was able to see that, although the agent reaches the goal more often, its behavior is not remarkable better than that one obtained by randomly selecting actions. The agent performance is still far from optimal.

4. Enhance the driving agent

- The first improvement has to do with the way the actions are selected. In order to encourage exploration, I implemented a ***softmax*** action selection policy where the probability of choosing action *a* is given by the Boltzmann distribution [1]:

$$P(a) = \frac{e^{Q(a)/\tau}}{\sum_{b=1}^{n} e^{Q(b)/\tau}}$$

Where *n* represents the number of possible actions and τ (*temperature*) is a positive tuning parameter. High temperatures cause all the actions to have similar probabilities, while low temperatures result in a greedy selection.

I tested and compared different initial temperature values. I made the temperatures decrease linearly with the number of trials in order to encourage more *exploitation* during the last trials.

- The second step involves tuning both α and γ. In order to assure convergence, I set α equal to $1/t$, where *t* represents the current trial.

I also tested different values for γ, but after analyzing the simulations I realized that large values of γ were misleading the algorithm. Sometimes, even after taking a *wrong* action and thus receiving a negative reward, the Q-values of some state-action pairs were increasing. I noticed that this was happening because, after taking the wrong action, the agent could end up on a new state with very large Q-values, which in turn would be *backpropagated* into the previous states. Because our agent has only a *local* view of the environment, I believe that we should not consider the rewards from the future states when updating our Q-values. The agent should learn how to solve each state locally, and thus not consider the values of future states.

I think that if the destination were located always on the same spot, then it would make sense to backpropagate the rewards from future states in order to take advantage of the reward received when reaching the goal. The same would happen if the agent had some idea about its current position with respect to the destination.

✓ Testing the algorithm performance:

Because we cannot simply use the received rewards as a comparison metric (long trajectories may receive higher rewards), I decided to use the percentage of time that the agent reaches the destination. I computed both the overall percentage (during 100 trials) and the % of the last 20 trials. The results presented below were the average over 3 runs.

AGENT PERFORMANCE FOR DIFFERENT γ VALUES		
$\alpha = 1/t$ | $\tau_{t=0} = 10.0$		
γ	% over 100 trials	% over the last 20 trials
0	45.0 %	82.6 %
0.2	43.3 %	84.0 %
0.5	36.3 %	75.0 %
0.8	36.3 %	73.3 %

AGENT PERFORMANCE FOR DIFFERENT τ VALUES		
$\alpha = 1/t$ | $\gamma = 0$		
τ	% over 100 trials	% over the last 20 trials
10	45.0 %	82.6 %
5	59.3 %	95.0 %
2	83.6 %	98.3 %

The best performance was obtained using a low initial temperature value ($\tau = 2.0$) and gamma $\gamma = 0$. The agent reaches the destination 83% of the times during 100 trials, and 98% during the last 20 trials (when exploration is significantly lower).

The reason why the agent reaches the destination 83% of the times is because the lack of exploration due to low temperature values and to the way the rewards are computed. Given that I am initializing the Q-values with zeros, taking a wrong action will immediately result in a negative reward which will cause the Q-value to become negative. After that, any unexplored action will result more attractive to the algorithm. If the action is correct, however, the Q-value will increase and will become the preferred action.

Basically, the algorithm right now is doing near-greedy selections and it happens to achieve a good performance given the way the rewards are computed and the initialization of the Q-values. The algorithm would have problems, however, if we increase the number of dummy agents in the environment. Until now, it is very unlikely to see another agent in an intersection, and those states are commonly not explored.

References

[1] Sutton R. and Barto A., *Reinforcement Learning: An Introduction*. MIT Press. 1998