# MLND Project 4 - Train a Smart cab to Drive

## Implement a basic driving agent

Q : In your report, mention what you see in the agent’s behavior. Does it eventually make it to the target location?

A :

* Because the actions are randomly produced, the agent has a rare chance to make it to the target.
* The reward/action table is listed below. The reward is randomly given depends on the random action, light and GPS.

|  |  |
| --- | --- |
| **Action** | **Reward** |
| None | 1 |
| Forward, if light is red | -1 |
| Left, if light is red | -1 |
| Else actions, if same as GPS | 2 |
| Else actions, if not same as GPS | 0.5 |

## Identify and update state

Q : Justify why you picked these set of states, and how they model the agent and its environment.

A :

* I choose Light, GPS, Oncoming and Left to define the set of states.
* The reasons of not including deadline:
  + It will dramatically increase the state space.
  + It will potentially influence our agent in making illegal moves
* And ‘else’ means the other actions could be taken, for instance, when light is red and the agent want to turn right, we only care if oncoming is turning left.

|  |  |  |  |
| --- | --- | --- | --- |
| **Light** | **GPS** | **Oncoming** | **Left** |
| red | right | left | forward |
| else | else |
| left | else | else |
| forward |
| None |
| green | left | forward | else |
| else |
| right | else |
| forward |
| None |

Following are the reasons why I choose these four variables and those values：

* Taking GPS into account because it will affect the amount of reward.
* Taking Light into account because it will affect whether the agent get reward or penalty.
* On a green light, turning left is okay only if there is no oncoming traffic at the intersection coming straight.
* On a red light, turning right is okay if there is no oncoming traffic turning left or traffic from the left going straight

## Implement Q-Learning

Q : After updating Q-values, what changes do you notice in the agent’s behavior?

A :

* The agent starts to choose the best action depends on which state it stays on.
* The agent tends to avoid choosing forwarding when the light is red.
* The agent tends to avoid choosing turning left when the light is red.
* The agent tends to follow the GPS.

## Enhance the driving agent

Q1 : Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform?

A1 :

*Q[s,a] ←(1-α) Q[s,a] + α(r+ γmaxa' Q[s',a'])*

Based on the above Q-Learning equation, I add some variables to primary agent：

|  |  |
| --- | --- |
| **Name of variable** | **Explanation** |
| self.trials | The number of finished trials |
| self.learning | Learning rate |
| self.discount | Discount factor |
| self.epsilon | For using epsilon-greedy strategy. |

* For updating learning rate：learning rate will be reset to 0.8/trials each trial.
* For updating epsilon：epsilon will be updated by multiplying a constant, like 0.9, at each time step. And it will be reset to 0.9/trials each trial.
* Changing the formula of updating parameters
* The third column shows how many incurred penalties each trial
* The fourth column shows how many net rewards each trial
* All formulas get all positive net rewards

|  |  |  |  |
| --- | --- | --- | --- |
| **Formula** | **Success Rate** | **Penalties** | **Rewards** |
| self.learning = 0.9 / self.trials  self.epsilon = 0.9 / self.trials | 34/100 | penalties.png | rewards.png |
| self.learning = 0.9 / self.trials  self.epsilon = 0.1 / self.trials | 37/100 | penalties.png | rewards.png |
| self.learning = 0.1 / self.trials  self.epsilon = 0.1 / self.trials | 6/100 | penalties.png | rewards.png |
| self.learning = 0.1 / self.trials  self.epsilon = 0.9 / self.trials | 75/100 | penalties.png | rewards.png |
| self.learning = 0.1 / self.trials  self.epsilon = 0.9 \*\* self.trials | 96/100 | penalties.png | rewards.png |
| self.learning = 0.1 / self.trials  self.epsilon = 0.9 \*\* (2\*self.trials) | 72/100 | penalties.png | rewards.png |
| self.learning = 0.5 / self.trials  self.epsilon = 0.9 \*\* (2\*self.trials) | 98/100 | penalties.png | rewards.png |
| self.learning = 0.5 / self.trials  self.epsilon = 0.9 \*\* (3\*self.trials) | 99/100 | penalties.png | rewards.png |

Q2 : Does your agent get close to finding an optimal policy, i.e. reach the destination in the minimum possible time, and not incur any penalties?

A2 :

Following is the parameters I choose：

|  |  |  |
| --- | --- | --- |
| **Name of variable** | **Value** | **Reason** |
| self.learning | 0.5 / self.trials | With higher value, agent can learn more from the environment. So I choose “0.5/trials” instead of “0.1/trials”. |
| self.epsilon(each trial) | 0.9 \*\* (3\*self.trials) | Because the agent need more exploration at beginning and more exploitation at the end, I choose to use power law. |

I think this agent is close to find the optimal policy, because two reasons

* Based on the following 6 test, the success rate has high possibility to be over 70/100.
* Based on the following 6 test, the agent has high possibility not to incur penalties.

|  |  |  |  |
| --- | --- | --- | --- |
| **Success Rate** | 99/100 | 83/100 | 100/100 |
| **Penality** | penalties.png | penalties.png | penalties.png |

|  |  |  |  |
| --- | --- | --- | --- |
| **Success Rate** | 39/100 | 72/100 | 99/100 |
| **Penality** | penalties.png | penalties.png | penalties.png |

Why agent sometimes incurs penalties after many trials?

* Because of epsilon-greedy algorithm, there is always a chance that the agent choose to follows the GPS and breaks the traffic law. For instance, when light is red, the agent choose to forward.

Why agent sometimes has low success rate?

* Because agent will go in circles based on the policy it learned. For example, agent will choose to turn right when the light is red and GPS says ‘left’.