> MLE Nanodegree
> Philipp Vogler
> March 2016

P4: Train a Smartcab to Drive

1. Implement a basic driving agent

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

The line action = random.choice([None, 'forward', 'left', 'right'])` makes the agent choose a random action. It makes its way through the world and to the target location by chance. Usually, it takes quite a lot of time until the target location is reached. The agent does not learn.

2. Identify and update state

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

I put everything into the state that the sensors record and that is relevant to the action decision. This includes the recommended next waypoint and the inputs about the environment (left, light, oncoming, right).
The deadline and the time step are not essential to the decision. They also would increase the number of states significantly, if recorded. I did not include deadline and time step in the state. They are used to adjust the learning and the exploration rate.

3. Implement Q-Learning

 ** What changes do you notice in the agent's behavior? **

 In the beginning, the actions are still random, due to the random initialization of the Q-table. The Q-table is updated based solely on the rewards the agent receives for its actions. Actions are chosen in accordance with the maximum Q-value of the neighboring states. No learning rate, exploration rate or discount jet.

  After about 30 runs, the agent starts to prefer certain actions in certain states. In some cases, this is useful to reach the target destination, which the basic agent does now more frequently than the random agent. This is the actual learning we are looking for.
  In other cases, it leads to problems like deadlock situations, a strong preference for the "None-action" or clockwise loops. This unwanted behavior takes quite some time to "unlearn" later on.

 Bottomline is that there is some learning happening, but it is slow to come. Not considering the future Q values of the neighboring states often leads to an agent trapped in a local minimum.

4. Enhance the driving agent

** 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? **

* Reduce the count of states/keys in the Q table
    In my fist Q table, there was the possibility for the recommendation for the next waypoint to be NONE, which is a state that the planner does not provide.
    Later I also scraped the 'right' input from the state, because it is handled by the traffic lights in a right-of-way environment.
    These measures reduced the number of states in the Q table significantly.

* Optimistic random initialization of the Q table
    Random initialization of the Q table works better than initialization with zeros. On a "Zero-table" the agent tends to repeat the action it picked first without ever trying the other options. This leads to cycling or long straight runs. With random values, this behavior is less common. The agent gets to am average reward per action of about 1.6 with a "Zero-table" after 30 runs. With a "Random-table" it is about 1.8. Because the agent has four possible actions to pick from I multiplied the random values with four to get a good spread over the space. It increases the average reward per action after 30 runs about 0.2. Spreading it further has not additional positive effect.

* Considering the future Q value in the Q table update
    I changed the update for the Q value in the Q table in the way that it considers the future Q values of the next possible states. This improves the learning significantly. High Q values from favorable states can trickle down to other good states now.

* Introduction of epsilon to solve the exploration-exploitation dilemma
    I introduced an exploration rate that should help to cover more states in the Q table. The way it is implemented with regard to the deadline minimizes the unwanted side effect of exploration under time pressure. The log function provides a quite stable exploration rate for the most time and then drops fast to zero (or less) when the deadline approaches "1". This is the behavior wanted here. The rate has been tweaked by trial and error. An exploration rate around 5% works best.
    It increases the average reward per action after 30 runs about 1.0 compared with the full or non-exploration scenario. It also helped in particular against the clockwise cycling of the agent.

* Introducing a discount gamma
    Considering the future states Q values helped the learning considerably. But it has to be balanced with the rewards for the action to get there. If the discount is not high enough, the chance for penalties increases.
    A discount of about 40% works best. The discount has been tweaked by trial and error. The average reward per step after 30

runes increases about 0.4, compared with the no discount scenario (gamma = 1).
    The introduction of gamma helped to handle the strong preference for the "None-action".

* Introduction of a learning rate
    The learning rate remarkably improved the speed of the agents learning. The results improve after fewer runs. A learning rate of about 80% works best. The rate has been tweaked by trial and error. The average reward per action after 30 runes increases about 0.5, compared with the full learning scenario (alpha = 1).

** 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? **

With the exploration rate, there is a random element in the decision process that prohibits 100% optimal behavior but helps to get it close to optimal.

That said, the policy gets very good. The reward per action, as well as the average reward, improves significantly. In the order of 20 to 40 runs the average reward asymptotes at about two. The reward per action in a run fluctuates but with an upward trend. Running in clockwise cycles or staying put for no reason becomes less and less often.
After about 50 runs the policy only improves marginally.

To get an "optimal policy", the Q table must be explored extensively, and the exploration rate must drop to zero at that point.

figure_1.png