Udacity Machine Learning Nanodegree
Project 4: Train a SmartCab to Drive

Wolfgang Steiner
wolfgang.steiner@gmail.com

August 2016

Task 1: Implement a Basic Driving Agent

Observe what you see with the agent's behavior as it takes random actions. Does the smartcab eventually make it to the destination? Are there any other interesting observations to note?

Given enough time, the agent will traverse the whole grid, following a random path and will reach the destination eventually. The rules of the game will not permit the smartcap to cross an intersection when the traffic lights are red or when another car has the right of way. In case of an illegal action, the smartcap will stop for the respective time step and a penalty (= negative reward) will be assigned. The world is toroid, which means that agents can drive off one edge of the world and reappear on the opposite edge.

Task 2: Inform the Driving Agent

What states have you identified that are appropriate for modeling the smartcab and environment? Why do you believe each of these states to be appropriate for this problem?

The following states are required for learning the agent's policy:

- **next_waypoint**
  The prime goal of the agent is to reach the destination in time. A rational agent will thus follow the direction that the planer has calculated for the current step.
  States: *left*, *right*, *forward*.
- **light**
  The second requirement for the agent is to follow the traffic rules. The state of the traffic light at the current intersection is thus crucial to avoid penalties.
  States: *red*, *green*.
- **oncoming**
  This input reports whether another car is oncoming at the current intersection and the direction it will continue in. When the traffic light is green, the agent is not allowed to turn left if there is an

oncoming car that goes either *forward* or *right*.
States: *none*, *left*, *right*, *forward*.

- **left**
  This input reports whether another car is coming from the left at the current intersection and the direction it will continue in. When the traffic light is red, the agent is not allowed to take a right turn if there is a car coming from the left that goes *forward*.
  States: *none*, *left*, *right*, *forward*.

The following inputs can safely be ignored by the agents:

- **right**
  This input reports whether another car is coming from the right at the current intersection and the direction it will continue in. When the traffic light is green, this other car has to wait and can safely be ignored. When the traffic light is red, cars coming from the right also do not interfere with the legal right turn. Because of this, this input can be ignored.
  States: *none*, *left*, *right*, *forward*.

- **deadline**
  The deadline reports how many steps the agent is allowed to take in the current trial before the deadline is reached and is decreased with every turn. Including the deadline does not make sense because the agent does not know its absolute position in the environment. Accordingly it does not know if a pending deadline will mean that it has to speed up (e.g. by breaking traffic rules) or that it is already close enough to not worry about it. Furthermore, including the deadline would increase the number of possible state by such a large amount that training the agent would not be possible anymore given the number of available trials/turns for training.
  States: Integer number $<\approx 70$.

How many states in total exist for the smartcab in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?

The number of states in this environment can be computed by multiplying the number of states for each input:

$$n_{states} = n_{next\_waypoint} * n_{light} * n_{oncoming} * n_{left} = 3 * 2 * 4 = 96 \quad (1)$$

There are 90 training trials, with about 30 steps per trial (random, based on initial distance to destination). Assuming that at least two encounters are necessary to train each state/action pair, the number of states trainable in this environment would be $90 * 30/4/2 \approx 337$. Thus, the number of trials/steps is sufficient to reliably train the agent using this state space. Furthermore, the number of other cars in the simulation is low which means that the states for *oncoming* and *left* will be *none* most of the time. This leaves $n_{states} = 6$ core states representing the basic behavior of the agent which will be easily learned given the number of available turns. Whether the agent will learn the remaining states involving other cars depends on the number of encounters during training which is pure chance. If not all of these additional states are encountered the result will be an agent that behaves well in the absence of other cars and could potentially take illegal actions when other traffic is present.

Task 3: Implement a Q-Learning Driving Agent

What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?

After implementing the Q-learning algorithm, the agent will at first behave like before: it will move around the environment randomly. When more trials are completed, the behavior of the agent starts to change: it will start to follow the direction of the planer and move into the general direction of the destination. When even more trials have been completed, the agent will improve in following the directions of the planer and will also avoid penalties due to traffic rule violations.

This behavior can be explained like this:

- At the beginning, the Q-table is empty and the agent explores the environment by choosing random actions.
- After every action, the q value is calculated:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha * (r_{t+1} + \gamma * \max_a(Q(s_{t+1}, a_{t+1})) - Q(s_t, a_t)), \tag{2}$$

  with reward r, learning rate α and discount factor γ.
- The updated (or new) Q-value is stored in the q-table.
- When the agent finds the current state in the Q-table it will choose the action with the highest Q-value, otherwise a random action is taken.
- The more states have been recorded in the Q-table, the better the agent becomes in choosing the right action.
- In order to sustain exploration, a random action is taken, based on the parameter ϵ.
- In later trials ϵ is diminished and the agent starts to rely on the best actions determined from the Q-table, thus following directions, avoiding traffic rule violations and reaching the destination more and more reliably.

Task 4: Improve the Q-Learning Driving Agent

4.1 Changes to the Q-learning Algorithm

4.1.1 Exploration vs. Exploitation

In the first implementation of the agent, ϵ was initialized to 1.0 and then diminished by a factor of 0.975 for every trial. This results in a value of $\epsilon = 1.0 * 0.975^{89} \approx 0.105$ at the 90th trial. In this way the agent will explore new states early in the trials and will then gradually rely more on the Q-table for choosing the best action. On the other hand, it would be advantageous if the agent would spend as much time as possible exploring, as the there are a relatively high number of states. This can be achieved by letting the agent explore for the first 90 trials ($\epsilon = 1$), thus discovering as many states as possible. In the last 10 trials ϵ is set to zero, so that the agent can apply the learned policy without being disturbed by random actions.

4.1.2 Learning Rate α

Usually the learning rate is initialized to some heuristic value and then diminished as the learning progresses. For highly stochastic environments a fixed value of α is also used. As I did not find an advantage of diminishing alpha during learning, I used the simpler implementation of a constant alpha. This also has the advantage that different values for α are easier to compare during grid search.

4.1.3 Discounting Factor

The discounting factor γ determines how strongly future rewards are incorporated into the Q-value. In order to implement this, the action a_{t-1}, reward r_{t-1} and state s_{t-1} of the previous step are retained to update $Q(s_{t-1}, a_{t-1})$ in time step t while computing $\max_a(Q(s_t, a_t))$ for the current state s_t and action a_t. For the case of $\gamma = 0$ an alternative code path is added which implements the much simpler

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha * (r_{t+1} - Q(s_t, a_t)). \tag{3}$$

Note that future awards are of limited utility in this environment as the next state the agent encounters is randomly generated by the environment. Increasing γ will therefore tend to introduce more randomness into the Q-table (which might be advantageous in some cases).

4.2 Metrics

- **Success rate**
  The number of successful trials is divided by the number of trials in a sample:
  $$\text{success\_rate} = \frac{n_{success}}{n}.$$
- **Penalty per step**
  Training the agent to minimize the total penalty is crucial for a save driving experience. To derive a metric that is comparable across trials, the total penalty (= sum of negative rewards) is accumulated during the trial and divided by the number of steps taken by the agent during the trial:
  $$\text{penalty\_per\_step} = -\frac{1}{n_{steps}} \sum_{i, r_i < 0} r_i.$$
- **Efficiency**
  The agent should follow the directions given by the planer as accurately as possible in order to minimize the total distance traveled to the destination. To compare this performance between trials, the minimum number of steps from the starting position/heading is divided by the number of steps taken by the agent in the trial:
  $$\text{efficiency} = \frac{n_{steps,min}}{n_{steps}}.$$
  When the agent succeeds in finding the shortest route, the efficiency is equal to one. Note, that in some rare cases the efficiency can be greater as one. This happens when the agent starts at the edge of the world and has to make a u-turn as its first move. When the agents jumps to the opposite edge during this maneuver, the planer might then accidentally find a shorter route to the target. This case has not been considered when computing the minimal distance.

4.3 Method

One *experiment* is defined as a set of 100 trials that are repeated 40 times. After set of 100 trials, the Q-table is cleared and the parameters γ, α and ϵ are reset to their starting values. Then, the mean values for the three metrics (success_rate, penalty_per_step and efficiency) are computed. In order to find the optimal values for γ and α, a grid search is performed by computing the mean values of the metrics while varying these two parameters. Code has been added to *simulator.py* in order to compute the metrics during grid search.

4.4 Results

Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?

The results of the grid search over γ and α for the full state space are shown in Fig. 1. Performance is worst for high γ and high α. Here, future rewards, which are basically random, have a high influence on the learning algorithm. When this is combined with a high learning rate, the agent has a high randomness in its decisions. This can be seen in the relatively low success rate (0.94), high penalty per step (0.17) and especially in the low efficiency (0.47). The low efficiency means that the agent does not follow directions closely but instead meanders around before reaching the destination.

Success rate increases with reduced gamma and approaches 100% for $\gamma <= 0.25$. Both penalty per step and efficiency clearly have the best values for $\gamma \leq 0.125$ (≤ 0.001 and ≥ 0.99 respectively). The differences for $\gamma = 0.125$ and $\gamma = 0.0$ are small for all three metrics which suggests that the simpler implementation for $\gamma = 0.0$ could also be used.

The influence of α is less pronounced than that of γ but can be summarized as follows: For $\gamma \geq 0.25$ increasing γ will degrade performance. For $\gamma \leq 0.125$ the learning rate has nearly no influence on success rate and efficiency while tending to decrease penalty per step.

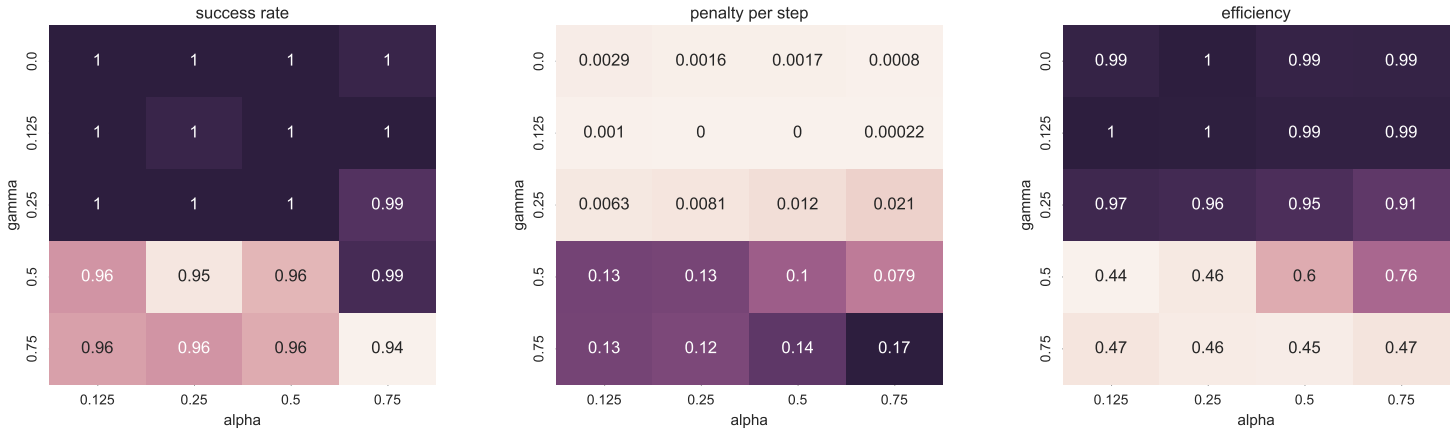

Figure 1: Results of the grid search over parameters γ and α for the full state space.

Based on these result I have implemented the final version of the agent with $\gamma = 0.125$ and $\alpha = 0.25$, where perfect scores where found during grid search. The first 90 trials are used for exploration ($\epsilon = 1$), while the last ten trials use the q-table ($\epsilon = 0$).

4.5 Discussion

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? How would you describe an optimal policy for this problem?

I would argue that the agent is quite close to finding an optimal policy based on the scores of the three metrics defined. Success rate is nearly 100%, penalties per step approach zero (0.00044 was reported during grid search) and the agent follows directions nearly perfectly (efficiency is found to be 1). In my opinion the optimal policy for the self driving cab could be described like this (in order of importance):

1. Obey the traffic rules and do not endanger passengers and other road users.
2. Follow directions of the planer as closely as possible.
3. Reach the destination on time (follows from 1. and 2., if at all possible).