# Train a Smart Cab to Drive

*Project 4 for Machine Learning Engineering Nanodegree*
*Udacity*

## Implement a basic driving agent

In the basic implementation of the driving agent, the deadline was disabled and the policy returns a random action $a \in \{None, forward, left, right\}$. The agent merely wanders around with no particular destination and ignoring traffic rules. It hits the target location completely by chance, but not actually aiming to get to it.

## Identify and update sate

The agent needs to learn to obey the traffic rules and follow the direction given by the planner.

*Traffic light*. The agent needs to learn that green is go and red is stop, except in special cases.

Special case 1: On green light, it can turn left only if there is no oncoming traffic at the intersection coming straight.

Special case 2: On red light, it can turn right if there is no oncoming traffic turning left or traffic from the left going straight.

In any of the special cases, it only matters if there is a car on *oncoming* and *left* traffic. It doesn't matter whether or not there is a car on the right traffic.

*Next waypoint*. The agent doesn't know its current location, and it doesn't know where the target location is. So, the agent needs to rely on the planner and learn to follow the given direction in order to get to the target location.

*Deadline*. Given that the agent doesn't know its current location, or the target location, or how far it is from the target location, the deadline deems irrelevant in the learning process. Whether there are 1000 or 1 time steps left, the agent would not know how close it is to the target location and decide whether or not it is better violate traffic rules to get to the destination faster. It will have no way to decide which next state is better based on remaining time. It therefore doesn't make sense to incorporate deadline into the state.

With this analysis, the states can be represented with 4 variables (light, oncoming, left, waypoint), with the following possible values:

$$\begin{aligned} light &\in \{ green, red \} \\ oncoming &\in \{ None, forward, left, right \} \\ left &\in \{ None, forward, left, right \} \\ waypoint &\in \{ None, forward, left, right \} \end{aligned}$$

This makes the size of the state space to $2 \cdot 4 \cdot 4 \cdot 4 = 128$, and the size of the Q-table (with 4 possible actions for each state) to $128 \cdot 4 = 512$.

The sate space can be reduced by merging the states that can be considered similar. Supposed the state of an intersection can be classified as either *busy* or *clear*. On green light, and considering special case 1, the intersection is *"busy"* if there is an oncoming traffic coming straight, otherwise the intersection is "clear."

On red light, and considering special case 2, the intersection is "busy" if there is an oncoming traffic turning left or if there is traffic from left going forward, otherwise the intersection is "clear."

$$intersection = \begin{cases} green_{busy}, & if\ light = green, oncoming = forward \\ green_{clear}, & if\ light = green, otherwise \\ red_{busy}, & if\ light = red, oncoming = left\ ||\ left = forward \\ red_{clear}, & if\ light = red, otherwise \end{cases}$$

This reduces the state variables to 2 *(intersection, waypoint)*, with the following possible values:

$$\begin{aligned} intersection &\in \{ green_{busy}, green_{clear}, red_{busy}, red_{clear} \} \\ waypoint &\in \{ None, forward, left, right \} \end{aligned}$$

The size of the reduced state space is now $4 \cdot 4 = 16$, and the size of the Q-table is down to $16 \cdot 4 = 64$.

## Implement Q-Learning

Q-Learning algorithm was implemented to help the agent decide which action to take. The formula used to update the each state and action pair $(s, a)$ in Q-table is

$$Q(s,a) = (1 - \alpha) \cdot Q(s,a) + \alpha \left[ r + \gamma \cdot \max_{\forall a' \in \mathcal{A}(s')} Q(s', a') \right]$$

where $\alpha$ is the learning rate, and $\gamma$ is the discount factor. Initially, these parameters were set to $\alpha = 0.5$ and $\gamma = 0.3$.

The policy picks the action with the best $Q\ (s, a)$ for a given state with the formula:

$$\pi(s) = \underset{\forall a \in \mathcal{A}(s)}{\operatorname{argmax}}\, Q(s,a)$$

For cases with multiple maximum $Q\ (s, a)$, action is randomly selected among the actions with maximum Q value.

When Q-learning was implemented, the agent lingers around with no particular destination during the early times of the simulation. Over time, it tends to take action on green light, and stop on red, while following the direction given by the planner. It eventually hits the target location.

When the agent follows the waypoint, and the light is green, and no other cars in the intersection, it receives a reward, making the $Q\ (s, a)$ for this sate-action pair bigger. So the next time it comes to this state, it always selects the action given by the way point.

When the light is red, and the agent takes no action, it doesn't receive any awards. But if it takes action, it receives a penalty (except for turning right, and no other cars in the intersection), making the $Q\ (s, a)$ negative. Hence, it selects to stop on red light.

The agent also receives a penalty, although smaller, if it takes action different from the next waypoint. However, there is no penalty or reward if it takes no action. Hence, when the light is red, it prefers to take no action rather than to follow the next waypoint (except when the next waypoint is turn right).

The environment only has 3 dummy agents, so the chances of our agent running on busy intersection are low. As can be seen in Figure 1, states with clear intersections are frequently visited while rarely on busy intersections. While the agent have learned to drive very well on clear intersections, it sometimes gets confused on the occasions when other agents are present in the intersection since it has less exposure/training to these states.

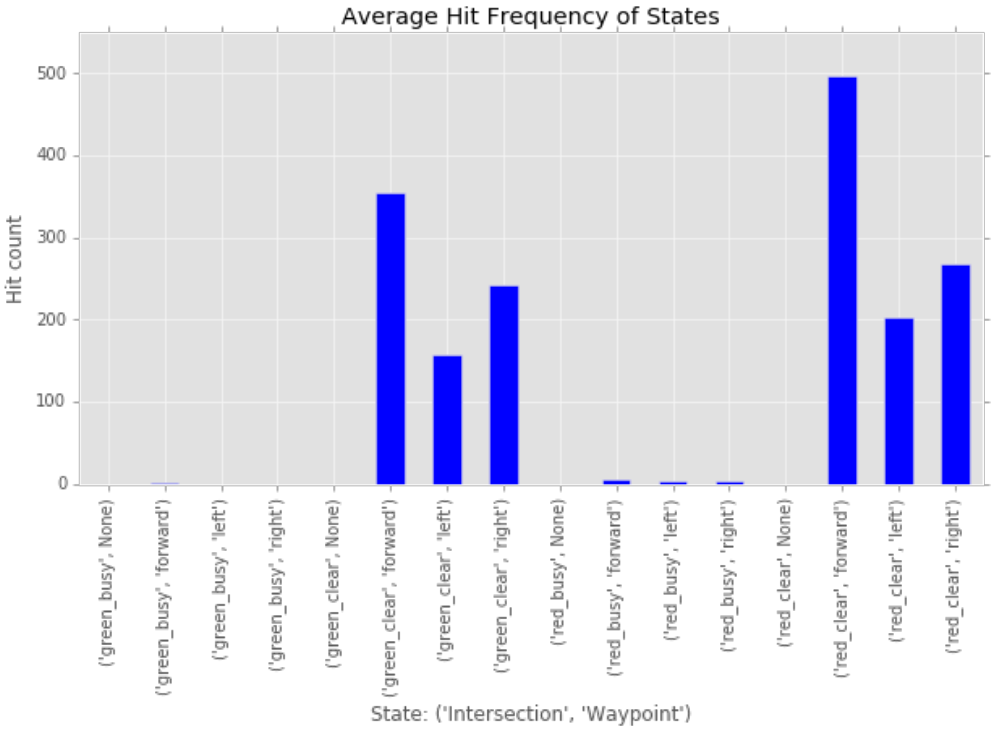


**Figure 1: Average hit frequencies per sate. States with clear intersections are frequently visited, while rarely on busy intersections.**

## Enhance the driving agent

The policy always picks the action with best value in the Q-table. As a result, some other actions are never tried out.

To allow exploration of other actions, we implement ε-greedy method. In ε-greedy, the policy picks random action with probability ε at a time, while picks the best action 1 – ε at a time:

$$\pi(s) = \begin{cases} \underset{\forall a \in \mathcal{A}(s)}{\operatorname{argmax}} Q(s,a), & \Pr = 1 - \mathcal{E} \\ \operatorname{random}(a), & \Pr = \mathcal{E} \end{cases}$$

where $0 \leq \varepsilon \leq 1$. Higher ε means choosing more random actions; more exploration. Lower ε means less exploration and more exploitation of the action with maximum $Q$ value.

It would be good to set higher ε value at the early stage of learning to let the agent explore more, and then decay the value of ε over time and to let the agent exploit what it have learned.

In our environment however, as seen in Figure 1, only 6 out of 16 states are frequently visited. If we use the same value of ε for all states, at the later part of learning, the agent will treat those rarely visited states the same as those visited frequently. As a result, the agent will assume that it also have learned the best action for those rarely visited states.

Instead of decaying ε with respect to time, a better option for our environment would be to let the agent exploit on states that it has already learned, and explore on states that were not yet visited. This can be done using an extension of ε-greedy method called ε-greedy VDBE-Boltzman[1].

ε-greedy VDBE-Boltzman uses the value difference of the current $Q_t\ (s,a)$ and its value in the next time step $Q_{t+1}\ (s,a)$. Larger difference indicates that the agent has not enough knowledge about the state, while small difference indicates that the agent is more certain about the state. The formula used for ε-greedy VDBE-Boltzman is

$$f(s,a,\sigma) = \frac{1 - e^{\frac{-|Q_{t+1}(s,a) - Q_t(s,a)|}{\sigma}}}{1 + e^{\frac{-|Q_{t+1}(s,a) - Q_t(s,a)|}{\sigma}}}$$

where $\sigma$ is a positive constant called *inverse sensitivity*. Smaller values of $\sigma$ cause the function to be very sensitive even to the smallest of value changes. Higher values cause the function to be less sensitive and react only to large value changes.

This formula is used to determine the value of $\varepsilon$ for each state

$$\varepsilon_{t+1}(s) = \delta \cdot f(s_t, a_t, \sigma) + (1 - \delta) \cdot \varepsilon_t(s)$$

where $0 \leq \delta < 1$ is the influence of the selected action to the exploration rate. The value of $\delta$ is set to the inverse of the number of actions $\delta = \frac{1}{|\mathcal{A}(s)|}$. Initially, the exploration rate is set to 1, $\varepsilon_{t=0}(s) = 1$, for all states to let the agent do full exploration.

Several combinations of learning rate $\alpha$, discount factor $\gamma$, and inverse sensitivity $\sigma$ have been tried out, with a total of 125 parameter combinations:

$$\begin{aligned} \alpha &\in \{0.01, 0.1, 0.5, 0.9, 1.0\} \\ \gamma &\in \{0.01, 0.1, 0.3, 0.9, 1.0\} \\ \sigma &\in \{0.01, 0.1, 0.5, 0.8, 1.0\} \end{aligned}$$

Each column in Figure 2 shows representatives of bad ($\alpha = 0.9, \gamma = 0.9, \sigma = 0.8$), fair ($\alpha = 0.1, \gamma = 0.01, \sigma = 0.01$), good ($\alpha = 0.01, \gamma = 0.3, \sigma = 0.8$) combinations of parameters ($\alpha, \gamma, \sigma$).

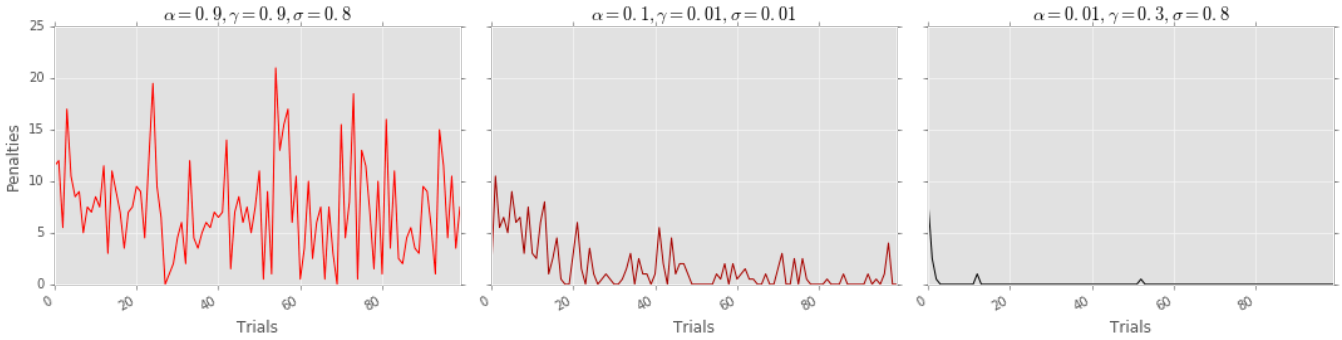


Figure 2(a): Accumulated penalties for each trial

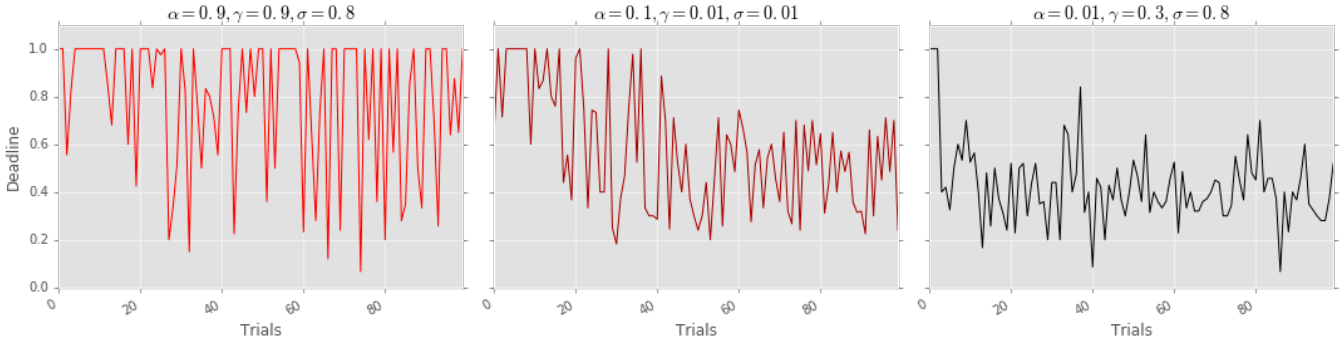


Figure 2(b): Time the agent took to reach the target destination in proportion to the set deadline for each trial

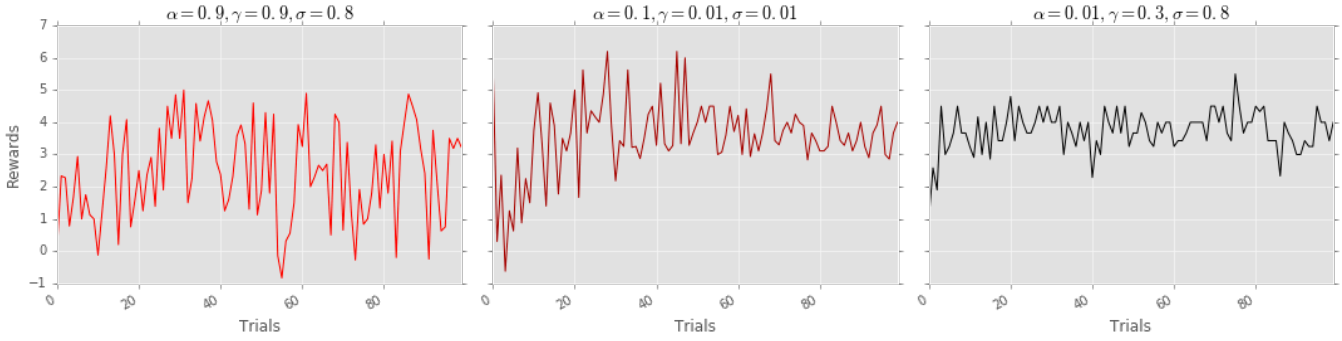


Figure 2(c): Accumulated rewards for each trial normalized by distance between initial potion and target destination

*Penalties*. Figure 2(a) shows absolute total penalties for each trial (rewards that are less than 0 are considered penalties). The bad parameter combination shows that the agent never seems to learn, and continues to run into accidents. In the fair parameter combination, the accidents continue to decrease until around the 30$^{th}$ trial, and the agent doesn't seem to learn fast enough after that. In the good parameter combination, the agent learns to avoid accidents in just less than 10 trials. It commits occasional accidents for few cases when there are other cars in the intersection.

*Deadline*. Figure 2(b) shows whether or not the agent reached the target destination within the set deadline for each trial. A value of 1.0 means that the agent ran out of time, otherwise it reached the destination in time proportionate to the set deadline. The first column shows that the agent frequently fails to get to the target destination. In the second column, it gets on time around the 40$^{th}$ trial onwards. In the third column, the agent is consistently on time after just a few trials.

*Rewards*. Figure 2(c) shows the accumulated rewards (positive and negative) for each trial normalized by the distance between the starting point and the destination. Since each correct action is rewarded, longer distances would accumulate more rewards in the optimum path, while shorter distances would accumulate smaller rewards; hence normalization over the distance is necessary in order to be consistent. It can be seen in the first column some trials accumulated negative rewards, while the third column consistently accumulated positive rewards.

When the inverse sensitivity is too small $\sigma < 0.1$, the agent becomes too sensitive to very small $Q$ value changes and cause to choose random actions even on the frequently visited states. The ideal value of $\sigma$ for our agent appears to be $0.5 \leq \sigma \leq 1.0$.

The combination of learning rate and discount factor $(\alpha, \gamma)$ is more delicate. When both $(\alpha, \gamma)$ are high, the agent doesn't appear to be learning. When one of them is very much lower than the other $(\alpha \ll \gamma \; OR \; \alpha \gg \gamma)$, the agent learns a feasible policy quickly. The agent also yields good results, although slower, when both $(\alpha, \gamma)$ are very low $(< 0.1)$.

## Conclusion

The planner gives directions to the agent of the shortest path to the target destination. Since the agent is not aware of its current location or the target location, an ideal policy for our agent would be to follow the direction set by the planner. In doing so, the agent must also follow traffic rules and avoid accidents, (i.e. (a) stop on red light, except when turning right and the intersection is clear; (b) go on green light except when turning left and there is an oncoming car).

After implementing Q-learning algorithm, enhancing the policy function, and fine tuning the parameters, our agent is able to learn the optimal policy quickly. With just few trials, the agent is able to get to the target destination on time, while not incurring accidents in the process.

[1] Michel Tokic: Adaptive ε-greedy Exploration in Reinforcement Learning Based on Value differences. Institute of Applied Research, University of Applied Sciences
http://www.tokic.com/www/tokicm/publikationen/papers/AdaptiveEpsilonGreedyExploration.pdf