Get out of the *Way!* *Training a Smartcab to Drive with Reinforcement Learning*

Kevin Nguyen[1]

[1]Udacity Machine Learning Nanodegree Candidate

ABSTRACT

This project examines the role of states, actions, rewards, models, and policy in a simplified world that characterizes driving in a modern urban city. We applied reinforcement learning techniques for a self-driving agent in this simplified world to aid it in effectively reaching its destinations in the allotted time. To do this, we first investigated the environment the agent operates in by constructing a basic driving implementation based on random actions. Next, we identify possible states the agent can encounter such things as traffic lights and oncoming traffic at each intersection and make him a self-driving agent. To create a smartcab, we implement a Q-Learning algorithm that guides the agent towards its destination within allotted time constraints. Our smartcab performs best when we adjust its parameters to have a decaying the learning rate and a stochastic action-selection function while setting the discount factor to 0.1 and initializing Q-values with 0.

Note: This report is based on Udacity's Reinforcement Learning project. Accompanying code for the report can be found here.

Introduction

Imagine in the not-so-distant future, taxicab companies across the United States no longer employ human drivers to operate their fleet of vehicles. Instead, the Taxi cabs are operated by self-driving agents — known as smartcabs — to transport people from one location to another within the cities those companies operate. In larger metropolitan areas, such as Chicago, New York City, and San Francisco, an increasing number of people have come to rely on smartcabs to get to where they need to go as safely and efficiently as possible. Although smartcabs have become the transport of choice, concerns have arisen that a self-driving agent might not be as safe or efficient as human drivers, particularly when considering city traffic lights and other vehicles. To alleviate these concerns, we use reinforcement learning techniques to construct a demonstration of a smartcab operating in real-time to prove that both safety and efficiency can be achieved.

Definitions

Environment

The smartcab operates in an ideal, grid-like city (similar to New York City), with roads going in the North-South and East-West directions. Other vehicles will certainly be present on the road, but there will be no pedestrians to be concerned with. At each intersection is a traffic light that either allows traffic in the North-South direction or the East-West direction. U.S. Right-of-Way rules apply[1]:

- On a green light, a left turn is permitted if there is no oncoming traffic making a right turn or coming straight through the intersection.
- On a red light, a right turn is permitted if no oncoming traffic is approaching from your left through the intersection.

Inputs and Outputs

We assume that the smartcab is assigned a route plan based on the passengers' starting location and destination. The route is split at each intersection into waypoints, and you may assume that the smartcab, at any instant, is at some intersection in the world. Therefore, the next waypoint to the destination, assuming the destination has not already been reached, is one intersection away in one direction (North, South, East, or West).

The smartcab has only an egocentric view of the intersection it is at. It can determine the state of the traffic light for its direction of movement, and whether there is a vehicle at the intersection for each of the oncoming directions. For each action, the smartcab may either idle at the intersection, or drive to the next intersection to the left, right, or ahead of it. Finally, for each trip has a time determined by the passenger to reach the destination. If the allotted time becomes zero before reaching the destination, the trip has failed.

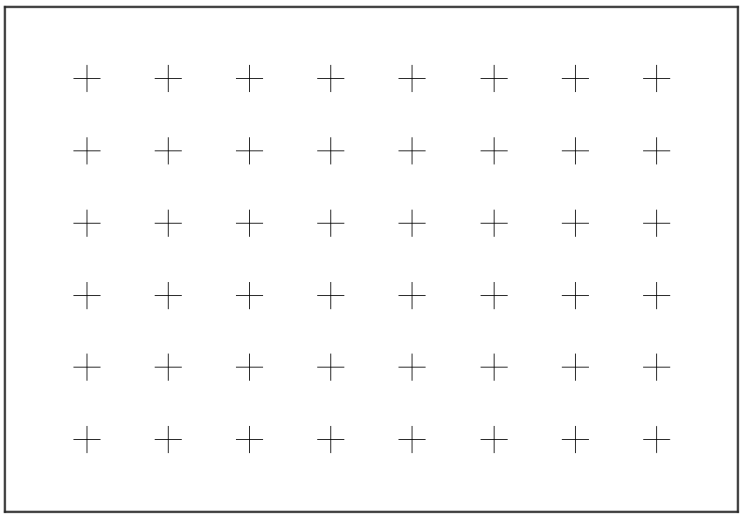


Figure 1. The environment. The smartcab will navigate around a six by eight grid plan with three other cars to reach its destination.

Rewards, Goal, and Evaluation

The smartcab receives a reward for each completed trip and also receives a smaller reward for each action it executes successfully that obeys traffic rules. The smartcab receives a small penalty for any incorrect action, and a larger penalty for any action that violates traffic rules or causes an accident with another vehicle. Based on the rewards and penalties the smartcab receives, the self-driving agent implementation should learn an optimal policy for driving on the city roads while obeying traffic rules, avoiding accidents, and reaching passengers' destinations in the allotted time. We identify the following metrics to assess our driving agent during their trips, with an overall goal of learning over time:

Number of times the agent reaches its destination (Success):
How many times does the driving agent reach its destination? To measure this, we count the number of times it reaches the destination.

Number of accidents and bad turns (Mistakes):
Can the driving agent learn the rules of the road and avoid accidents? To measure this, we measure how many accidents and moves it makes counter to the move assigned by the waypoint.

Learning over time (Intervals of 10 trips):
Our main objective is measuring how well does the driving agent learns over time. In 100 trips, we look at the number of times the agent reaches its destination and the Mistakes it makes broken down into intervals of 10. More specifically, we group the results into 10 sequential intervals and take the aggregate sum of the metrics mentioned (ie. trips 1 through 10, 10 to 20, etc...) over the course of its 100 trips.

Implementing a Basic Driving Agent

To implementing a basic driving agent, the cab is instructed to select an action randomly out of the set of possible actions (right, left, forward, and none) for every state the cab encounters during their trip to implement a basic driving agent. Keep in mind; we do not enforce a deadline (time constraint) during this phase.

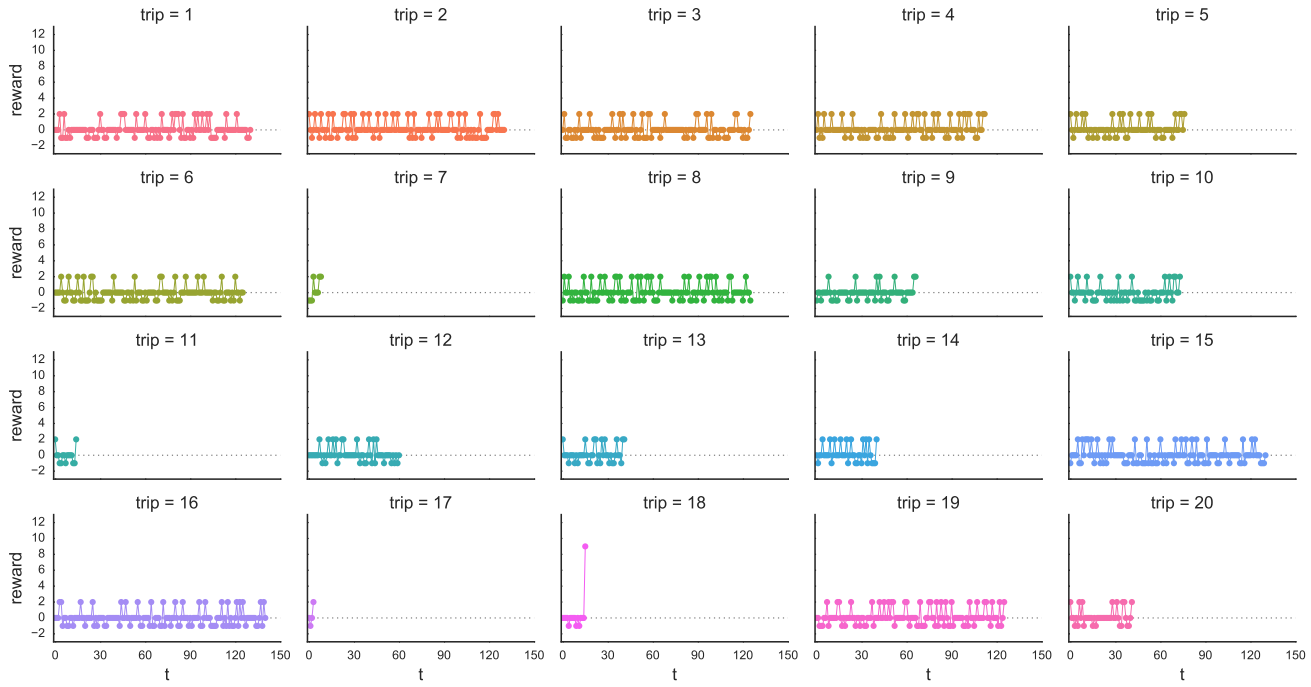


Figure 2. First 20 trips with random actions. By taking random actions, the smartcab can eventually reach its destination after driving around the city. However, along the way it makes many mistakes and takes a while to reach its destination.

Informing the Driving Agent

To improve upon the driving agent taking random actions, we define several inputs and outputs that make up the state space:

Waypoint (the next move):
Based on the cab's current location, the waypoint is the next instructed action to reach its destination (imagine a GPS telling you the next step in your route). There are three possible outputs with next waypoint: left, right, forward. Waypoint is important because it informs the next action a cab should take to reach its destination. Balancing waypoint with the environment (traffic laws, oncoming traffic, etc...) is crucial for the cab to arrive at the destination given a deadline.

Traffic light at an intersection:
At every intersection, the cab will see a traffic light. There are two possible outputs for a traffic light: green or red. This input gives the cab's sense of order, what is right or wrong (the rules of the road).

Presence of oncoming vehicles:
At every intersection, the cab will have a chance of dealing with oncoming traffic. There are four possible outputs with oncoming traffic include: left, right, forward, and none. Learning oncoming helps our cab avoid accidents. The presence of oncoming vehicles is useful when making left or right turns.

Positioned to the left:
At every intersection, the cab will what the car to its left is going to do into its decision process. Possible outputs with the car on our left include: left, right, forward, and none. Learning what actions of cars surrounding the cab helps prevent accidents.

Input	Output
Traffic light	Green or red
Waypoint	Left, right, or forward
Oncoming traffic	Left, right, forward, or none
Position left	Left, right, forward, or none

Table 1. Mapping inputs and outputs to Design a state space for our smartcab. The total number of states in our environment is 96 (3 for waypoint * 2 for traffic light * 4 for oncoming traffic * 4 for left position).

The size of the state space we designed is 96, which is manageable taking into account our cab will go on 100 trips. For every trip, the smartcab picks up a passenger who gives it an arbitrary deadline (number of moves i.e. 20, 30, 60, etc..) to take them to the destination. While the smartcab is driving around the city (for each action made during a trip), the cab explores a state. Given the goal of Q-learning (learn and make informed decisions about each state), it's important for our environment to be small enough for our cab to get to explore most the states a few times before to learn the environment. In table 2a and table 2b, we see an informed smartcab with no deadline completes more trips compared to with a deadline and makes a similar number of bad decision.

Interval	Success	Mistakes
01 - 10	1	79
11 - 20	2	82
21 - 30	2	99
31 - 40	0	71
41 - 50	2	70
51 - 60	4	74
61 - 70	1	77
71 - 80	1	76
81 - 90	2	82
91 - 100	2	76

(a) 100 trips with a deadline

Interval	Success	Mistakes
01 - 10	1	253
11 - 20	1	162
21 - 30	3	231
31 - 40	4	203
41 - 50	5	240
51 - 60	4	227
61 - 70	4	258
71 - 80	7	155
81 - 90	6	170
91 - 100	3	153

(b) 100 trips with a deadline with no deadline

Table 2. Random actions. Taking random actions in a six by eight traffic grid with no deadlines, does not prevent the smartcab from eaching its destination. It just needs a little extra time. The downside is the smartcab commits many "bad moves" (commits traffic violations and gets into accidents) while random driving around the city.

Implement a Q-Learning Driving Agent

The next goal is to demonstration the cab's ability to operate in real-time while being both safe and efficient while get around the city with reinforcement learning techniques. To do so - we implement the Q-learning algorithm[2] which instructs the cab to selects an action with the highest future reward given a state. After initializing values to all possible states with 0, the cab explores the city and makes updates to its Q-table (which can be thought of a look up table where it keeps track of the q-values for all the states it visits). Mathematically, this can be understood as:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

$Q(s_t, a_t)$ is the quantity of a state-action combination for a given time step. $Q(s_t, a_t)$ on the left-hand side is the new estimate and to the right of the equation is the old estimate. Alpha or α is the learning rate, which is the extent new information overrides

old information for our smartcab. A learning rate close to 0 means the cab will absorb new information less quickly whereas an alpha close to 1 means new information will be processed faster. R(s) is the reward we get from transitioning to a new state. The discount factor or γ is applied to the future q-value of the state to determine the future worth of rewards compare to the value of immediate rewards. A factor of 0 will make the smartcab "short-sighted" by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward. Argmax is the maximum expected reward based on all possible actions made in when in the next state. Simply put, the new action value = the old value + the learning rate X (new information - old information). Jake Bennett, CTO of POP breaks this down well in his article on the Q-learning algorithm.[3]

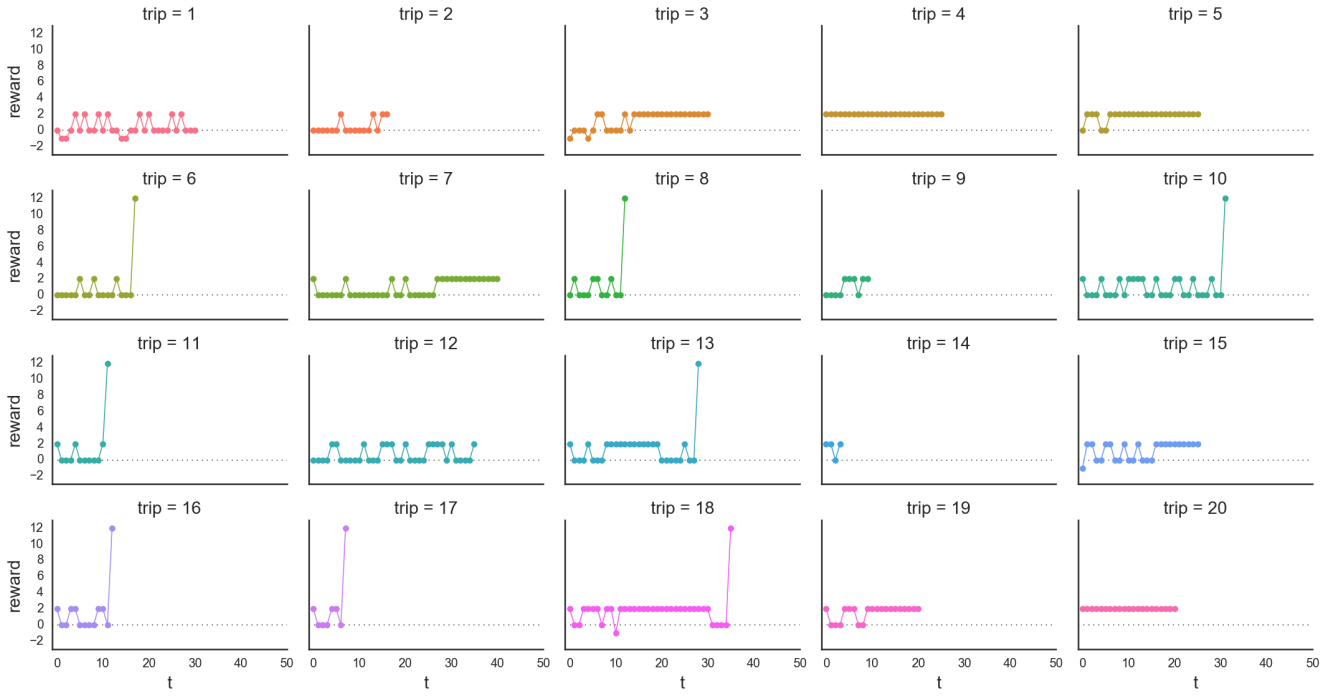


Figure 3. Exploring the first 20 trips for Q-Learning driving agent. The Q-Learning driving agent learns to not make mistakes fast, committing its first 6 after 3 trips. When is the cab is informed about our state space, there is an increase in successful trips and a reduction in the number of mistakes it makes during the simulation compared to simulations for our random action driving agents.

Interval	Success	Mistakes
01 - 10	6	0
11 - 20	5	2
21 - 30	6	0
31 - 40	4	0
41 - 50	6	1
51 - 60	6	0
61 - 70	9	0
71 - 80	1	0
81 - 90	6	1
91 - 100	8	0

(a) Informed Driving Agent

Interval	Success	Mistakes
01 - 10	4	7
11 - 20	5	0
21 - 30	5	3
31 - 40	6	1
41 - 50	6	1
51 - 60	5	2
61 - 70	2	3
71 - 80	2	3
81 - 90	7	1
91 - 100	5	0

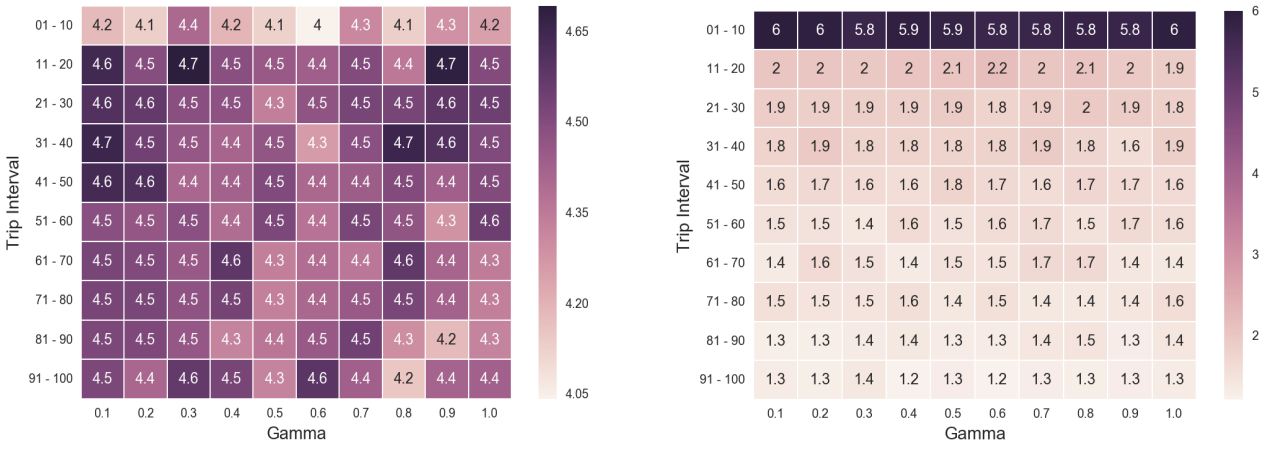
(b) Driving Agent with Q-learning

Table 3. Making less mistakes. Compared to the informed driving agent, the Q-Learning driving agent makes slightly more mistakes but to reaches its destination at a comparable rate with the informed driving agent. Notably, the smartcab makes mistakes early on but learns to makes less mistakes as it completes more trips.

Improving the Q-Learning Driving Agent

We find the smartcab improves its success rate while slightly increasing its mistakes when slowly decaying its learning rate and using cross entropy to manage the exploration rate. Because of computing limitations, a small grid search was performed to estimate the best discount factor. And following that, we 1000 simulations are run to compare the original Q-Learning driving agent to our proposed agent. The following are our final parameters:

1. **Learning rate:** Slow decay with $\frac{1}{\ln(t+2)}$
2. **Discount factor:** 0.1
3. **Exploration rate:** Cross entropy with -ln($sigmoid(t+2)$)
4. **Initialization:** Initial Q-values are set to 0 in all possible states



(a) Average Success After 300 Simulations

(b) Average Mistakes After 300 Simulations

Figure 4. The discount factor. We performed a grid search by running 300 simulations, each having the driving agent attempt 100 trips. Our best results for gamma was 0.1. All of our simulations had a initialization rate of 0.

Interval	Success	Mistakes
01 - 10	4.2	5.463
11 - 20	4.533	1.529
21 - 30	4.514	1.385
31 - 40	4.423	1.296
41 - 50	4.317	1.123
51 - 60	4.356	1.16
61 - 70	4.242	1.046
71 - 80	4.264	0.958
81 - 90	4.217	0.952
91 - 100	4.093	0.931

(a) Q-Learning Driving Agent

Interval	Success	Mistakes
01 - 10	4.209	5.95
11 - 20	4.507	2.04
21 - 30	4.57	1.789
31 - 40	4.547	1.81
41 - 50	4.43	1.609
51 - 60	4.464	1.508
61 - 70	4.577	1.527
71 - 80	4.39	1.417
81 - 90	4.45	1.349
91 - 100	4.504	1.291

(b) Improved Q-Learning Driving Agent

Table 4. Finding the optimal policy. To estimate the optimal policy, we used the parameters mentioned above and ran 1000 simulations for our improved and original Q-Learning driving agent. We found that on average the improved driving agent completes more successful trips while deceasing mistakes over time.

References

1. Marsland, S. *Machine learning: an algorithmic perspective* (CRC press, 2015).

2. Wikipedia. Q-learning (2016). URL https://en.wikipedia.org/wiki/Q-learning. [Online; accessed 09-August-2016].

3. Bennett, J. Machine learning, part 3: The q-learning algorithm (2016). URL http://www.wearepop.com/articles/secret-formula-for-self-learning-computers. [Online; accessed 09-August-2016].