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 set parameters to have a decaying the learning rate and a stochastic action-selection.

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 both safety and efficiency of using Q-Learning to train a smartcab.

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.

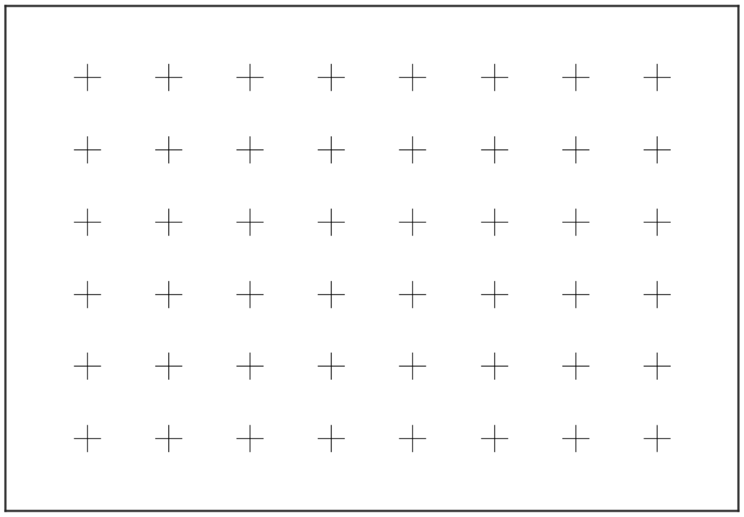


Figure 1. The environment. The smartcab will navigate around a six by eight grid plan. Three other cars will also be driving around as the smartcab tries to reach its destination.

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 minor 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 occasions 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 ten sequential intervals and take the aggregate sum of the metrics mentioned (i.e.,. 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.

Informing the Driving Agent

To improve upon the driving agent taking random actions, we select the following inputs and outputs to define the state space:

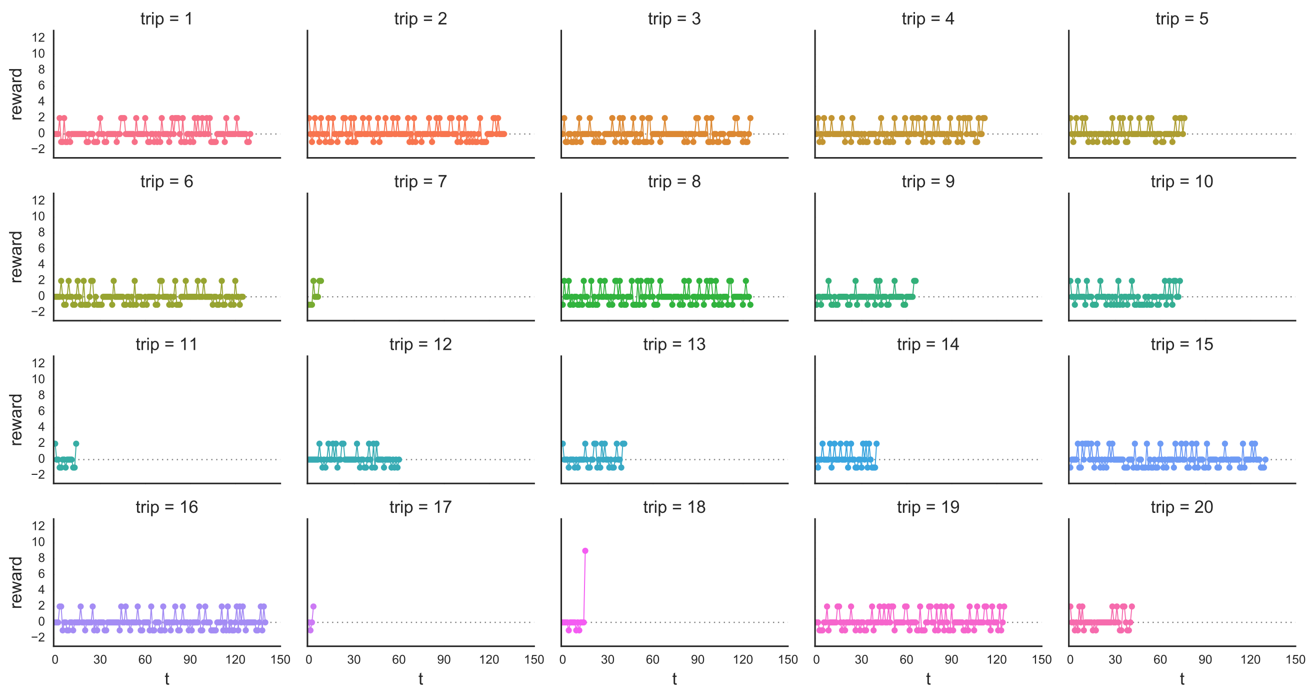


Figure 2. First 20 trips with random actions. By taking random actions, the smartcab can eventually reach its destination after driving around the city. However, along the way it makes many mistakes and takes a while to reach its destination.

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).

Taking into account our cab will go on 100 trips, the size of the state space we designed is manageable at 96. We made the choice, to exclude inputs such as a cab's deadline or the position to their right. For instance, if we added the position to the right, our state space would expand from 96 to 384. By excluding these inputs, the number of states are manageable and allow the agent more time to learn and explore more state. 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 time limit and makes a similar number of mistakes.

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 "mistakes" (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 getting 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 (think 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]

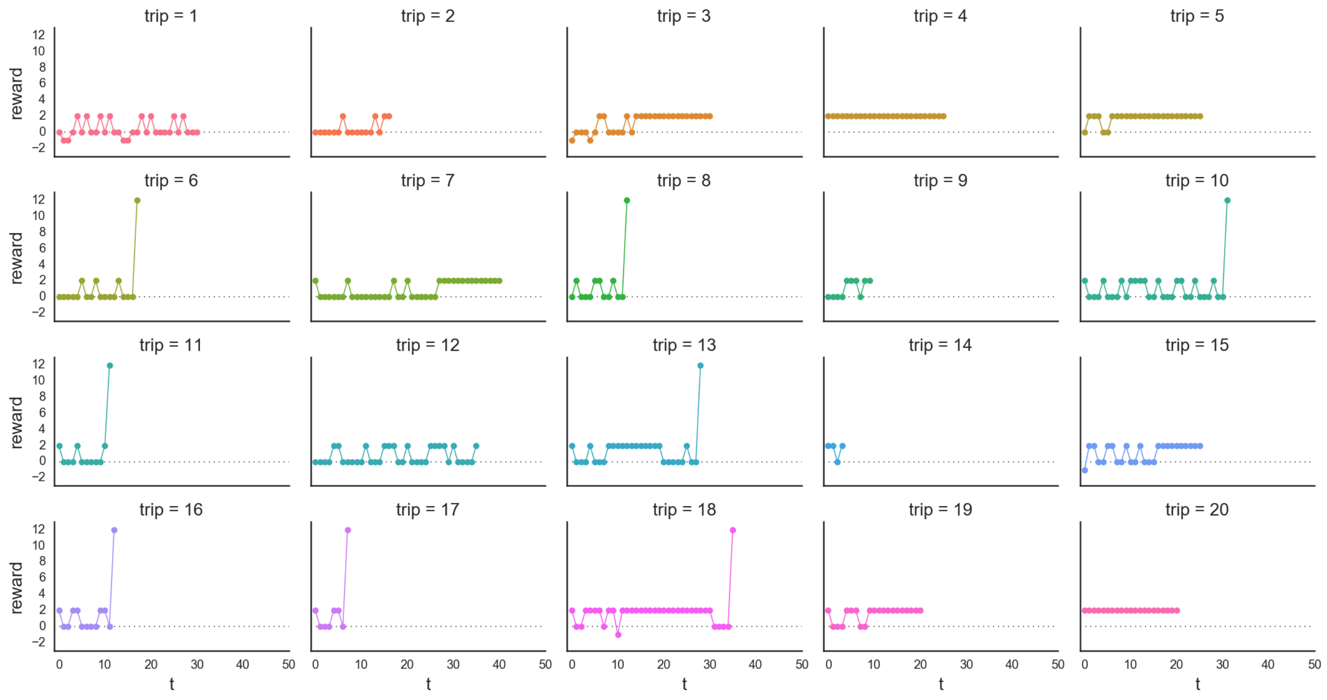


Figure 3. Exploring the first 20 trips for Q-Learning driving agent. The Q-Learning driving agent makes mistakes early but learns to reduce the number of mistakes (committing its first six after three trips) within a short period. When is the cab is informed about our state space, there is an increase 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

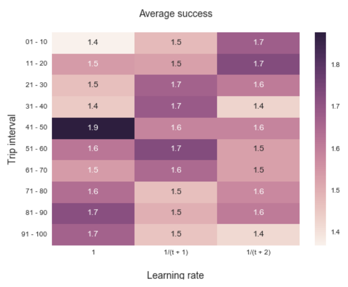
Table 3. Making fewer mistakes. Compared to the informed driving agent, the Q-Learning driving agent makes slightly more mistakes but to reaches its destination at a comparable rate with the informed driving agent - learning to make fewer mistakes as it completes more trips.

Improving the Q-Learning Driving Agent

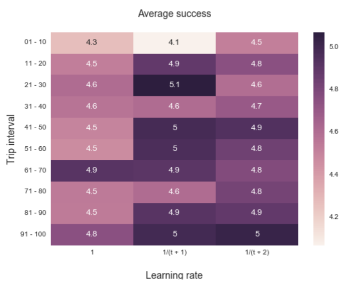
To improve our Q-Learning driving agent, we performed a grid search to find a more optimal learning and exploration rate. The following parameters were used for the learning rate : 1, 1/t, and -ln(sigmoid(t+2) and for exploration rate: 1, 1/(t+1), 1/(t+2). Due to computational limitations, we limited this exercise to 100 simulations. We found that a learning rate of 1/(t+2) and exploration rate of -ln(sigmoid(t+2) on average had the best balance of successful trips and mistakes made (**Figure** 4).

Next, we performed a grid search to estimate the best discount factor from 0.1 to 1.0 (ten even intervals) with 300 simulations. Base on our results (**Figure** 5), we find inconclusive evidence for an optimal a discount factor given our previous assigned parameters. More specifically, the average success and mistakes did not have noticeable variation from one another.

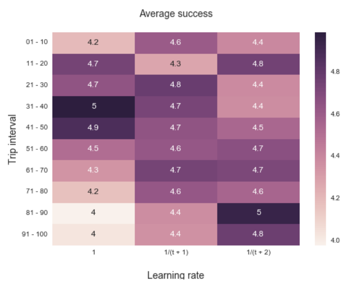
Finally we ran 1000 simulations (**Table** 4) to compare the original Q-Learning driving agent against our proposed agent (learning rate of 1/(t+2), ln(sigmoid(t+2), and a discount factor of 0.1, which was arbitrary selected). We found that on average, our proposed agent reached its destination more frequently and committed a similar amount of mistakes (which is what we hoped).



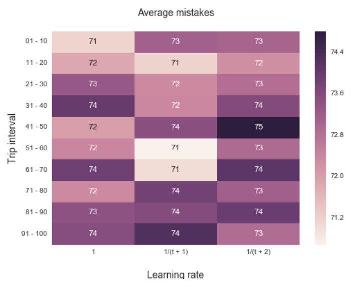
(a) $\varepsilon = 1$



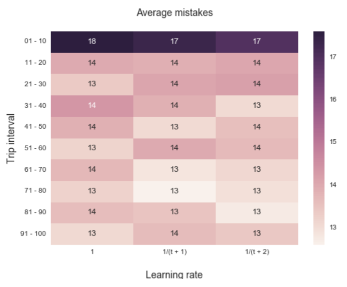
(b) $\varepsilon = 1/t$



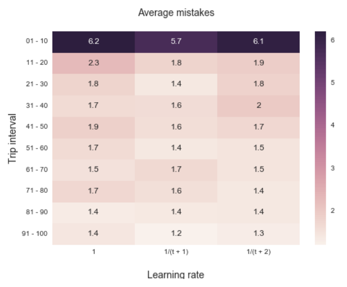
(c) $\varepsilon = -\ln(sigmoid(t+2))$



(d) $\varepsilon = 1$



(e) $\varepsilon = 1/t$

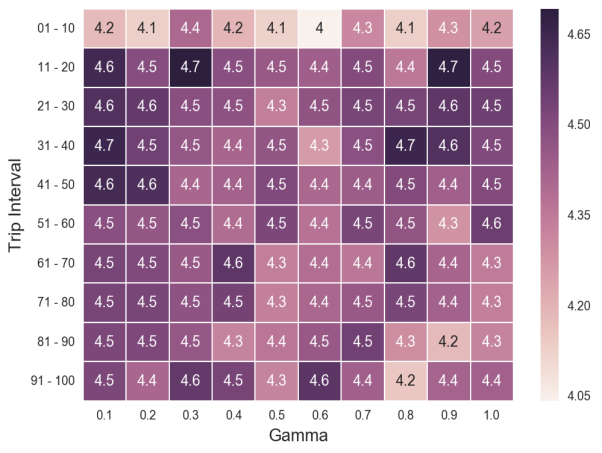


(f) $\varepsilon = -\ln(sigmoid(t+2))$

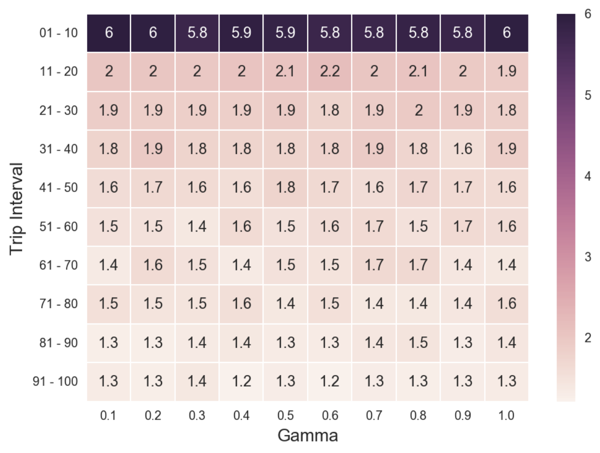
Figure 4. Our goal is to pick parameters with a balance high success and low mistakes. In the first row, we compare the average success rates for (a), (b), and (c). If we just consider the first row, (c) does not look optimal, however we are interested in both success and low mistakes (safety). Therefore we factor the second row (average numbers of mistakes made) into our evaluation. We conclude the third column has the optimal policy because in the second row, its corresponding mistakes are far below than others (compared (d) and (e) to (f) shows the agent on average, makes the fewest mistakes) and we are willing to make that trade off. Moving forward we use the learning rate of 1/(t+2) and exploration rate of $\log(sigmoid(t+2))$

Discussion

The ideal policy for our driving agent gets its destination while not committing driving mistakes (both efficiency and safety). With the proposed parameters comes close to the stated optimal policy; the driving agent can drive around the city safely and reach its destination (most of the time). However, we believe it acts too conservative (abides by the rules of the road too closely) to be optimal. For instances, by examining **Table** 5 - we see our driving agent develops an instinct to get stuck at a red light and not adjust its strategy. The agent stays at the red light, and unable readjust strategy in anticipation of a long wait. A truly optimal policy would allow for the agent to reroute itself to deal with red lights. If a human were driving, they would probably turn right and readjust their strategy if they approached a traffic light they knew would take a long time. A visual representation of the driving log can be seen in figure’s 6 and 7.



(a) Average Success After 300 Simulations



(b) Average Mistakes After 300 Simulations

Figure 5. The discount factor. We performed a grid search by running 300 simulations, each having the driving agent attempt 100 trips. We did not find a gamma that stood out from our parameters search so we select 0.1 for demonstration purposes. Results show that our improved agent on average does better than the original Q-Learning agent.

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 decreasing mistakes over time.

Summary of our proposed agent parameters

1. **Learning rate:** Slow decay with $\frac{1}{\ln(t+2)}$
2. **Discount factor:** An arbitrary choice of gamma 0.1 was selected because no optimal parameters were found.
3. **Exploration rate:** Cross-entropy with -ln$(sigmoid(t+2))$
4. **Initialization:** Initial Q-values are set to 0 in all possible states

Index	Trip	Move	Destination	Location	Light	Waypoint	Oncoming	Left	Action	Reward	Moves left in trip
1828	91	19	[1, 6]	[1, 2]	green	right	None	forward	None	0	1
1832	92	2	[3, 2]	[4, 4]	red	forward	None	None	None	0	18
1835	92	5	[3, 2]	[3, 3]	red	left	None	None	None	0	15
1836	92	6	[3, 2]	[3, 3]	red	left	None	None	None	0	14
1837	92	7	[3, 2]	[3, 3]	red	left	None	None	None	0	13
1838	92	8	[3, 2]	[3, 3]	red	left	None	None	None	0	12
1839	92	9	[3, 2]	[3, 3]	red	left	None	None	None	0	11
1842	93	1	[7, 3]	[4, 5]	red	forward	None	None	None	0	34
1843	93	2	[7, 3]	[4, 5]	red	forward	None	None	None	0	33
1845	93	4	[7, 3]	[5, 5]	red	forward	None	None	None	0	31
1846	93	5	[7, 3]	[5, 5]	red	forward	None	None	None	0	30
1848	93	7	[7, 3]	[6, 5]	red	forward	None	None	None	0	28
1849	93	8	[7, 3]	[6, 5]	red	forward	None	None	None	0	27
1851	93	10	[7, 3]	[7, 5]	red	left	None	None	None	0	25
1852	93	11	[7, 3]	[7, 5]	red	left	None	None	None	0	24
1854	93	13	[7, 3]	[7, 4]	red	forward	None	None	None	0	22
1855	93	14	[7, 3]	[7, 4]	red	forward	None	None	None	0	21
1869	94	12	[6, 1]	[6, 4]	red	forward	None	None	None	0	8
1870	94	13	[6, 1]	[6, 4]	red	forward	None	None	None	0	7
1871	94	14	[6, 1]	[6, 4]	red	forward	None	None	None	0	6
1875	95	1	[5, 5]	[2, 6]	red	forward	None	None	None	0	39
1876	95	2	[5, 5]	[2, 6]	red	forward	None	None	None	0	38
1877	95	3	[5, 5]	[2, 6]	red	forward	None	None	None	0	37
1878	95	4	[5, 5]	[2, 6]	red	forward	None	None	None	0	36
1881	95	7	[5, 5]	[5, 6]	red	left	None	None	None	0	33
1882	95	8	[5, 5]	[5, 6]	red	left	None	None	None	0	32
1884	96	0	[6, 2]	[8, 5]	green	right	None	None	None	0	25
1910	97	0	[2, 5]	[4, 1]	red	left	None	None	None	0	30
1911	97	1	[2, 5]	[4, 1]	red	left	None	None	None	0	29
1912	97	2	[2, 5]	[4, 1]	red	left	None	None	None	0	28
1913	97	3	[2, 5]	[4, 1]	red	left	None	None	None	0	27
1914	97	4	[2, 5]	[4, 1]	red	left	None	None	None	0	26
1916	97	6	[2, 5]	[3, 1]	red	forward	None	None	None	0	24
1917	97	7	[2, 5]	[3, 1]	red	forward	None	None	None	0	23
1919	97	9	[2, 5]	[1, 2]	green	left	None	forward	left	0	21
1945	98	4	[4, 4]	[4, 3]	red	left	None	None	None	0	21
1946	98	5	[4, 4]	[4, 3]	red	left	None	None	None	0	20
1949	99	1	[4, 4]	[7, 5]	red	left	None	None	None	0	29
1950	99	2	[4, 4]	[7, 5]	red	left	None	None	None	0	28
1951	99	3	[4, 4]	[7, 5]	red	left	None	None	None	0	27
1953	99	5	[4, 4]	[6, 5]	red	forward	None	None	None	0	25
1955	99	7	[4, 4]	[5, 5]	red	forward	None	None	None	0	23
1956	99	8	[4, 4]	[5, 5]	red	forward	None	None	None	0	22
1959	100	0	[3, 6]	[6, 1]	red	forward	None	None	None	0	40
1960	100	1	[3, 6]	[6, 1]	red	forward	None	None	None	0	39
1961	100	2	[3, 6]	[6, 1]	red	forward	None	None	None	0	38
1965	100	6	[3, 6]	[3, 1]	red	left	None	None	None	0	34
1966	100	7	[3, 6]	[3, 1]	red	left	None	None	None	0	33
1969	100	10	[3, 6]	[3, 3]	red	left	None	None	None	0	30
1970	100	11	[3, 6]	[3, 3]	red	left	None	None	None	0	29
1971	100	12	[3, 6]	[3, 3]	red	left	None	None	None	0	28
1972	100	13	[3, 6]	[3, 3]	red	left	None	None	None	0	27
1973	100	14	[3, 6]	[3, 3]	red	left	None	None	None	0	26

Table 5. **Reviewing non-rewards and mistakes** Examining one of the simulations show that our driving agent is not quite smart enough to find a strategy to get around a red light. Instead of realizing if a light just turns red and making an adjustment by taking a new route (such as turning right, left, left, and right to move forward) it opts to stay put at a red light when instructed to go forward or left.

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].

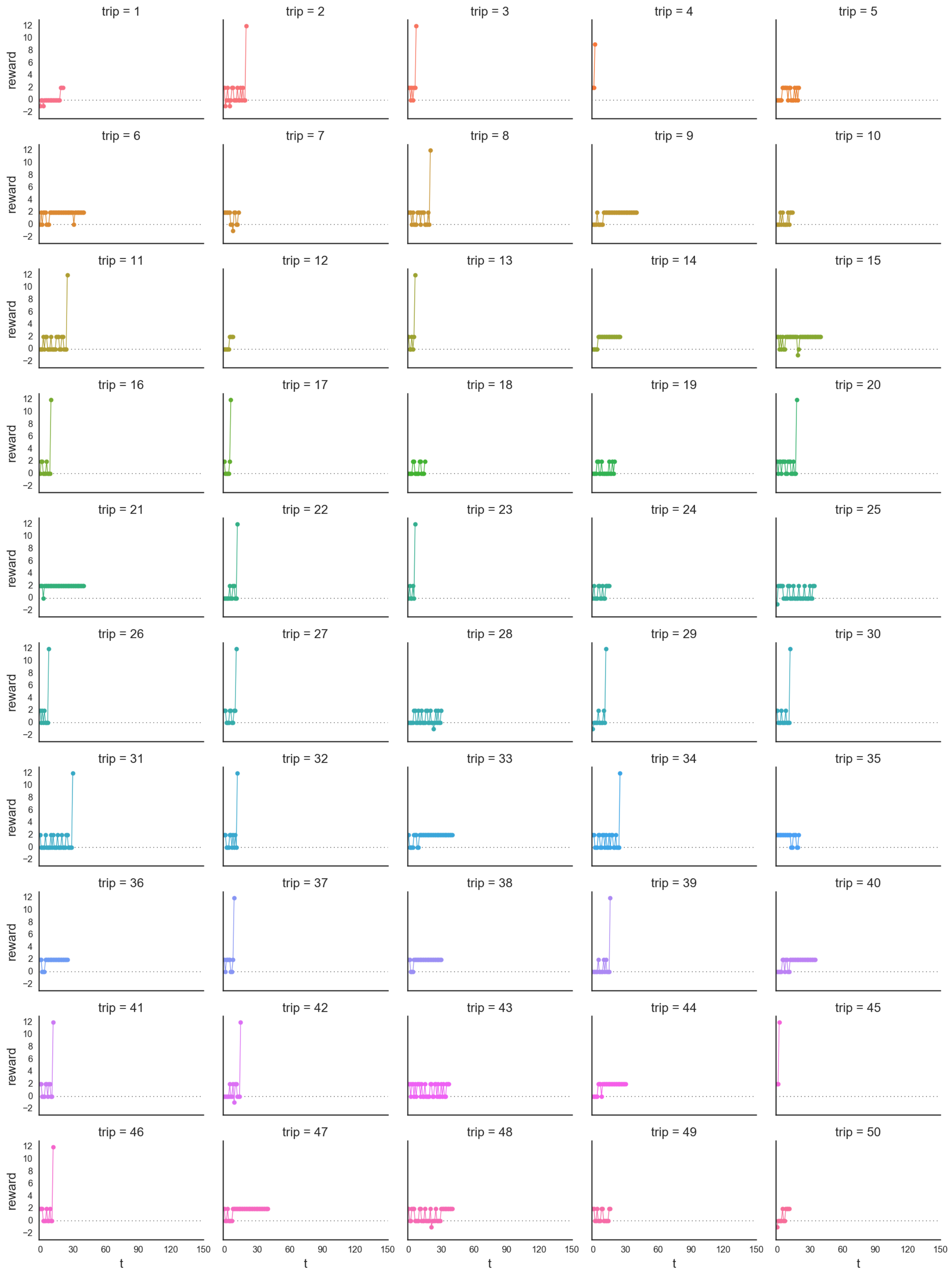


Figure 6. First 50 trips. An example of our smartcab's first 50 trips when parameters are set according to our policy.

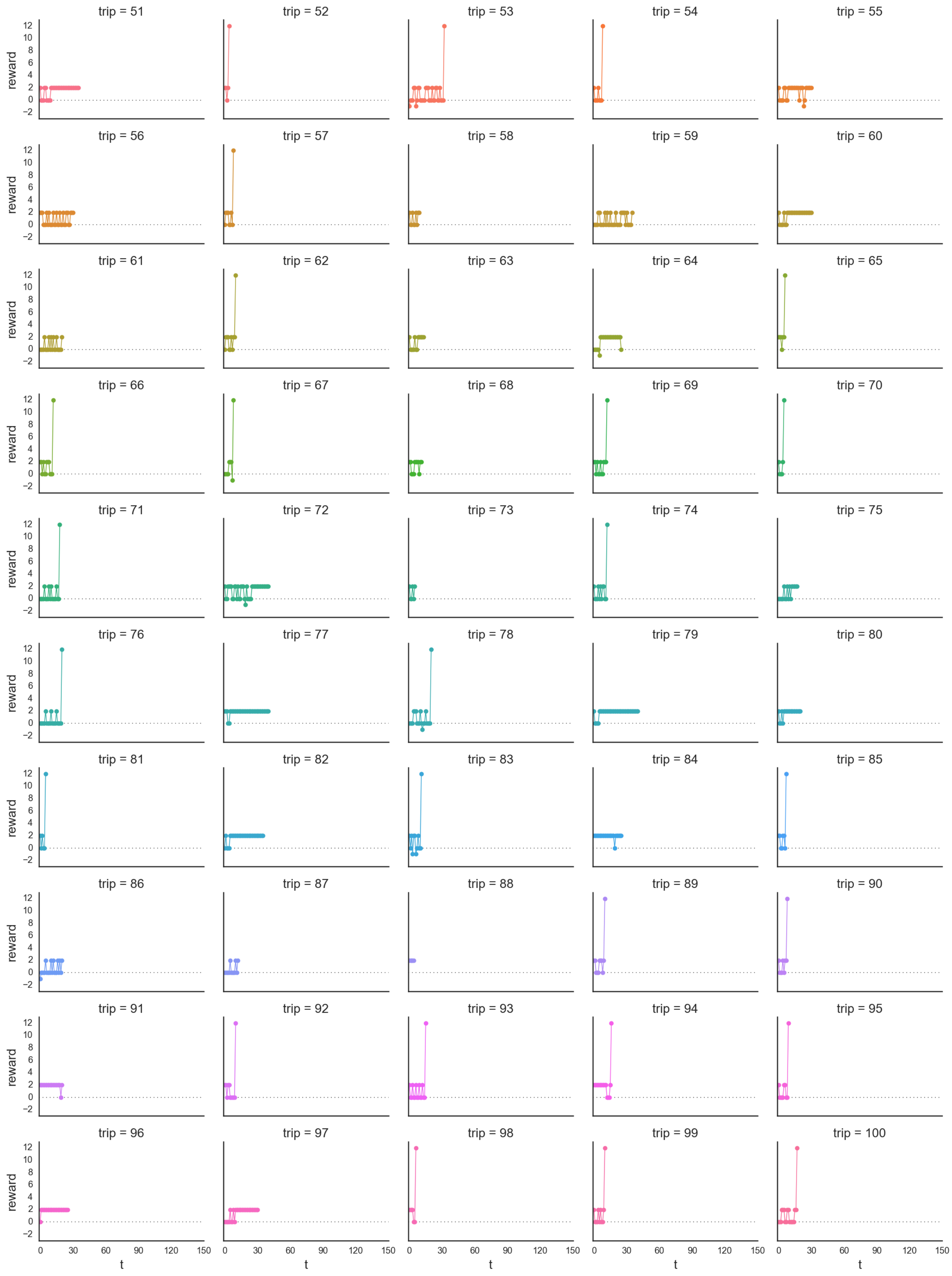


Figure 7. Last 50 trips. Continuation of our example smartcab's, it performs fairly well in the last 50 trips when parameters are set according to our policy.