

Machine Learning Engineering Nanodegree

Project 4: Train a Smartcab How to Drive

version 0.4

Jouni Huopana
1/17/2016

CONTENTS

0. Introduction	2
1. Simulation environment study	2
2. Implementing a basic driving agent.....	3
3. Implement Q-Learning and enhancing the driving agent	5

0. INTRODUCTION

This document contains information relating to a Smartcab training code, which is used for the development of a machine learning model to guide a virtual Smartcab. Basic analysis is made of the training environment and inputs. These inputs are then used to establish the state of the Smartcab and actions. After defining the relevant states and actions, a Q-learning algorithm is used to train the Smartcab to work independently. Finally the software is tested in the simulated environment. The Python code can be found in accompanied Python code – file (agent.py). This document has been created as a project work for the Udacity Machine Learning Engineer Nanodegree.

1. SIMULATION ENVIRONMENT STUDY

The goal is to train a Smartcab to drive in a grid world from a random starting point to a random goal. A view from this world can be seen in figure 1. The world consist of an 8 x 6 grid. If a car continues over the boundary of the world it will be transferred to the opposite side of the grid. All intersections have traffic lights which are either red or green. The world has also three other cars driving and reacting to the environment. All the cars are assumed to follow simple traffic rules: *On a green light, you can turn left only if there is no oncoming traffic at the intersection coming straight. On a red light, you can turn right if there is no oncoming traffic turning left or traffic from the left going straight.*

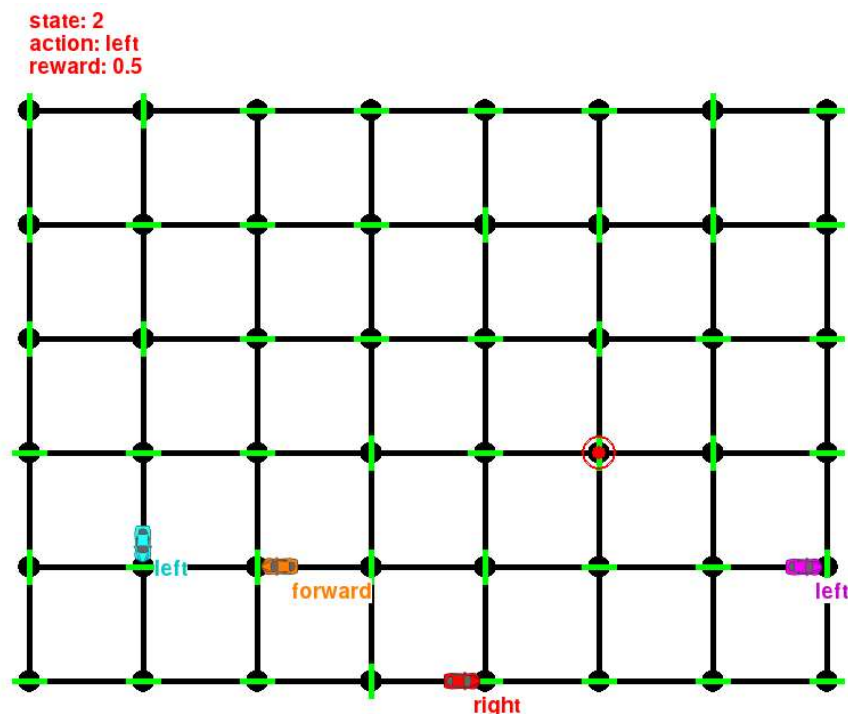


Figure 1 Grid world with cars, green light direction indicators and trip target (red circle).

2. IMPLEMENTING A BASIC DRIVING AGENT

The Smartcab software has the following flow:

- *Inputs* of the current environment answer to the following questions:
 - Is the traffic light red or green?
 - Is there other cars in the intersection and in which direction the other cars are going?
 - Where should the Smartcab go next?
- Produce a *state* for the Smartcab.
- Determine the *action* based on the learned environment and *rewards*
- Move to the next point in the grid world

Inputs are used to determine the state of the Smartcab. In figure 2 you can see an interpretation of an intersection with cars in every direction and traffic lights. If all the possible combinations just for this intersection would be taken into account, the number of possible states would be 128 (2 light colors * 4 oncoming car options * 4 right car options * 4 left car options). It could be possible to use all of these states to but as there is limited time for training, it is more sensible to only those which are relevant for the Smartcab or combine them in efficient manner.

In this case the *state* is defined with the following possibilities:

- Is the light green or red? (2 options)
- Is there an oncoming car in the intersection? (2 options)
- Is there a car coming from the left? (2 options)
- In which direction the Smartcab should go next? (3 options)

With these inputs it is possible to follow the given traffic rules assuming that all other cars in the world also follow the traffic rules. Table 1 shows the all the possible 24 combinations.

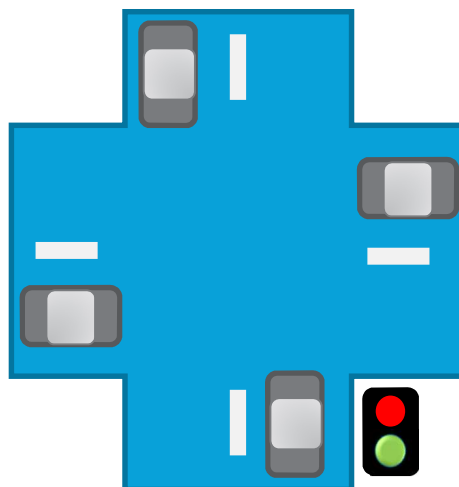


Figure 2 Intersection with cars and traffic light options

Table 1 Possible states according the inputs and occurrence in random iteration

STATE	TRAFFIC LIGHT	ONCOMING	FROM LEFT	NEXT DIRECTION	COUNT IN 100 RANDOM ITERATIONS
1	green	0	0	forward	1506
2	green	0	0	left	1426
3	green	0	0	right	2716
4	green	0	1	forward	39
5	green	0	1	left	32
6	green	0	1	right	53
7	green	1	0	forward	33
8	green	1	0	left	15
9	green	1	0	right	64
10	green	1	1	forward	0
11	green	1	1	left	1
12	green	1	1	right	0
13	red	0	0	forward	1684
14	red	0	0	left	1545
15	red	0	0	right	3057
16	red	0	1	forward	18
17	red	0	1	left	18
18	red	0	1	right	24
19	red	1	0	forward	37
20	red	1	0	left	40
21	red	1	0	right	95
22	red	1	1	forward	0
23	red	1	1	left	0
24	red	1	1	right	0

The presented *state* definition was implemented in the software with a random *action* function. This means that in each state the following action was a random choice between forward, left, right or none. To study how often each of these states happens, number of trials was set 100. This means that a random goal and start point will be assigned to the world and after that the Smartcab starts randomly drive around the grid until it reaches the destination. From the last column of table 2, you can see that the most common state was number 15. Right turns seem to be high as it allows the Smartcab to move when the traffic light is red. Also, it was clear that the Smartcab can eventually reach the destination even with random actions. The number of steps varied between 10 and 400, but the average step count for 100 trials was 116.

3. IMPLEMENT Q-LEARNING AND ENHANCING THE DRIVING AGENT

Q-learning is a simple iterative process where the Q-matrix, which is basically matrix where all the states-action combinations available, is updated with a *reward* function. The reward function is a given function with the other environmental information of the grid world. The reward function returns larger values when the taken action has a favorable outcome and smaller values when the effect of the action is negative one or against the rules of the world. For example violating a red light in an intersection.

$$Q(s, a) = Q(s, a) + \alpha \left(R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a) \right) \quad (1.)$$

Equation 1 shows the used Q-learning model for the Q-matrix. The Q is dependent on state and action as is the R for the rewards. The initial taken action is random. Variables α and γ are learning parameters. α being the learning rate and γ being the discount factor. Setting the learning rate to 0 means that the Q-values are never updated, meaning that there is no learning. The discount factor can be set between 0 and 1 which scales the effect of any future action. Parameter γ was set to 0.35 and α to 0.1. During the testing it was noticed that γ values over 0.5 seemed not to converge.

Table 2 shows the learned Q-matrix in the training. Higher value indicates better action. There are some zero rows as the earlier testing suggested. Also it can be seen that in most cases red light means no action, but in some cases a right turn is preferred, which indicates that the model has learned the simple traffic rules. When in use the rewards stayed always positive. Taken actions were always random. After the training was done, the model was tested without update cycle and with a deadline on the iterations. In this case the trained model reached the destination 50% of the time within the deadline.

Another Q-learning model with slight modification was also implemented. In this case the action taken was random 50% of the time. Also when on a green light, Smartcab has to move and with red either stop or take a right. With this model a similar training test was done, resulting Q-matrix is shown in table 3. Training parameters α and γ were kept the same. When tested in a deadline environment this model achieved a 30% success rate.

To limit the randomness of the action a linear decay was introduced to the model. In this case the action taken was random 55% of the time at the start of the algorithm, but only 5% at the last iteration. With this model a similar training test was done, resulting Q-matrix is shown in table 4. Training parameters α and γ were kept the same. When tested in a deadline environment this model achieved a 25% success rate, which did not make a significant improvement.

Table 2 Q-matrix achieved with equation 1

STATE	NONE	FORWARD	LEFT	RIGHT
1	1,538	3,335	1,578	1,748
2	1,538	1,740	3,079	1,756
3	1,538	1,733	1,578	3,241
4	1,104	1,930	0,542	1,310
5	1,056	1,234	1,876	1,218
6	1,221	1,127	1,351	2,118
7	1,219	2,750	1,039	1,027
8	0,495	0,592	0,836	0,427
9	1,386	1,405	1,284	2,915
10	0,000	0,000	0,000	0,000
11	0,000	0,000	0,000	0,000
12	0,000	0,158	0,000	0,308
13	1,538	0,263	0,078	1,740
14	1,538	0,263	0,078	2,029
15	1,538	0,228	0,078	3,344
16	0,721	0,033	0,026	0,427
17	0,529	0,061	0,008	0,813
18	0,630	0,097	0,016	0,873
19	1,186	0,087	0,063	0,966
20	0,942	0,099	0,043	1,151
21	1,330	0,121	0,069	2,806
22	0,000	0,000	0,000	0,000
23	0,000	0,000	0,000	0,000
24	0,000	0,000	0,000	0,000

Table 3 Q-matrix achieved with equation 1 and 50% random action

STATE	NONE	FORWARD	LEFT	RIGHT
1	0	1,5725	1,9538	1,7834
2	0	1,6696	1,7449	1,5221
3	0	1,8336	1,8395	1,966
4	0	0,854	0	0,3601
5	0	0,1586	0,7333	0
6	0	0,2575	0,1165	1,0019
7	0	0,1653	0	0,3819
8	0	0	0	0,2244
9	0	0	0,6347	0,1834
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	1,355	0	0	2,0647
14	2,0789	0	0	1,4315
15	2,1188	0	0	1,8894
16	0,1162	0	0	0,1672
17	0,3936	0	0	0,1158
18	0	0	0	0,8218
19	0,5585	0	0	0
20	0,0537	0	0	0,4545
21	0,2588	0	0	1,3409
22	0	0	0	0
23	0	0	0	0
24	0	0	0	0

Table 4 Q-matrix achieved with equation 1 and decaying random action

STATE	NONE	FORWARD	LEFT	RIGHT
1	1,2743	2,0173	1,0682	1,0541
2	1,5324	0,7765	0,9595	1,0709
3	1,5229	1,1456	1,0106	1,2162
4	0,4623	0	0,1352	-0,0606
5	0,6159	0	0	0
6	1,2192	0	0,1769	0
7	0,7644	0,1306	0	0,2539
8	0,13	0,0917	0	0
9	0,6095	-0,0528	0,091	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0,8644	0,9905	1,0765	1,5489
14	0,2938	0,2713	0,5295	1,5031
15	1,4547	0,9727	0,8863	1,1128
16	0,0776	1,2151	0	0
17	0	0	0	0
18	0,4675	0,0743	0	0
19	0,05	0	0	0,4687
20	0	0	0,2885	0,2681
21	0,1	0,3785	0	0,6922
22	0	0	0	0
23	0	0	0	0
24	-0,0786	0	0	0