Udacity Project 4: Training a Smart Cab

Brian Lester

June 13, 2016

Implement a basic driving agent

> *"In your report, mention what you see in the agents behavior. Does it eventually make it to the target location?"*
>
> - Question 1, *Tasks 1*

Using random move selection the agent makes almost no progress. It just moves randomly (obviously) which means that it only has about a $\frac{1}{4}$ chance to make a move that helps it towards the goal. While no all the trials I ran it did eventually reach the goal this is not guaranteed. This randomness leads to it often going far over the allotted time. While some times it did make all the right moves and got to the goal before the deadline, it went far over the deadline most of the time (many runs ran until -100 time-steps or more). Not only does a random agent take a long time it also tries to make illegal turns a lot. It also can cause accidents. This is show cased when I recorded the total rewards for trips. These rewards were often hugely negative which shows that this random behavior is far from ideal.

Identify and update state

> *"Justify why you picked these set of states, and how they model the agent and its environment."*
>
> - Question 2, *Tasks 1*

The set of states I used are all the possible combinations of the following inputs:

- The light color: This light is either green or red. This input is necessary for the agent to know what possible options are legal. For example the green light means that froward, left, and right are all legal options while a red light means that only None and right are legal.
- The presence of traffic from the left: Traffic from the left plays a key role in decisions for turning right on red. When there is no traffic then the agent may make a right turn on red but if there is traffic it should not.

- Oncoming traffic: This plays a role in the decision to make a left turn. If the light is green and there is oncoming traffic then the agent should not turn left while the lack of traffic allows for a left turn
- The waypoint itself: This input tells the agent where it wants to go and helps it select an action. Without this input the agent might as well act randomly.

The traffic from the right is an input that is not needed. In the real work traffic to the right is important as another agent may be doing something incorrectly but in the simulation the people to your right don't matter. (This is easy to see, when you turn left the light for the people in the right have a red and can't go so you don't have to look at them. The same applies when you are going straight. When you turn right whether people there are goring straight or not doesn't matter).

The second input that is not part of my states is the deadline. Part of the reason that I don't use it is because the decisions that are made when there is a deadline shouldn't very much. When there is a lot of time the agent should still try to get to the destination as soon as possible and when the deadline is low it should also try to follow the optimal policy. Not only does the deadline cause decisions to change it would also make the table to large, currently the table is 128 x 4 (512) states but adding the deadline into the states would make the table huge. Even if the maximum deadline is only 30 the table would grow to 3840 x 4 (15360 states). This makes the table far to large.

All the input I use as state are important because they give the agent knowledge of the environment (light, left, and oncoming) and of the model (the waypoint that tells the agent where it wants to go).

Implement Q-Learning

"What changes do you notice in the agents behavior?"

- Question 3, *Tasks 2*

Once I implement Q Learning the agent starts acting far less random. When the simulation first starts the agent act pretty randomly. When all the possible actions have the same value in the QTable then the agent just chooses a random action. As it choses actions it updates the QTable with the new reward which in turns helps it choose the action the next time. After Q learning the agent makes it to the waypoint in time most of the time. The current problem with the behavior is that sometimes the agent with make 4 of the same turns in a row (thus ending up at the same spot) before continuing to go forwards.

Enhance the driving agent

"Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does

it perform?"

- Question 4, *Tasks 2*

The first tweak that I have implemented is Optimistic Initialization. When I initialize the Q Table all the states are initialized to 4.0 rather than 0.0. This means that the agent is less likely to get stuck in local minima. This change causes the agent to more fully explore the state space leading to better agent behavior.

For my initial implementation I used reward discounting for my Q learning algorithm but this presents some problems. I included the next waypoint in my state which makes the following the exact q learning equation hard. Normally the equation is Q(s, a) ¡-(alpha) reward + gamma * the max reward from the next state. Without the next waypoint the agent cannot obtain the max reward from the next state. Not only does the lack of the waypoint make this form of Q learning impossible but the next state also includes the traffic and Light position in the next state. Due to the lack of global knowledge the state that you move into while taking an action a is unknown. This seems like a problem that would be unable to be solved because you do not know your next state. Without being able to peer into the future using the Bellman equation and discounted reward is most likely the best option.

The final tweak I did was to adjust the alpha values. My initial value was alpha = 1. In this case the alpha being one means that means that the learning rate is set to 1. This means that the old value is not used in the update because 1 - alpha (1) is zero. I tested out several alpha values and got the following results.

This table shows the following for each alpha value I tested, number of correct trials over three run where the data is collected from the last 10 trials (after 90 of training). It also shows the most recent trail where a negative reward (most likely a traffic violation) occurred. It also includes the average error from the a distance optimal route (described later in the report). Each of these metrics were averaged over 3 runs.

Alpha	Correct trials	Negative rewards	Average error
1.0	30/30	94	1.2
0.8	30/30	90	0.8
0.6	30/30	95	1.1
0.5	30/30	95	1.2
0.3	30/30	97	2.3

The metric of Negative rewards is highly variable and not a good indicator of success. As discussed later in the report the agent has some places where it doesn't have a good policy (it has not been in that state before). I collected this metric because the idea seemed good but after outputting the argmax for each state I realized that it is not a good benchmark.

This table shows that the best alpha values seems to be 0.8

"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?"

- Question 5, *Tasks 2*

In this case the optimal policy is that the agent reaches the destination within the time limit while making as few moves a possible while also taking the minimum amount of time to do so. The optimal policy should also not break any traffic laws.

The agent seems to follow find a close to optimal policy, it reaches the destination within the deadline every trial I have watched and when I was compiling the data for the table above the agent successfully reached the destination every time (in the final 10 trials after 90 trials of training). The agent also does not incur many penalties after many trials.

Due to lack of global and future knowledge it is impossible to know if it is using the absolute optimal possibility. For example if the agent is at position (0,0) and the destination is at (1, 2) then the optimal movement is 3 timesteps. It also can be done optimally in 3 moves (one to the left and 2 down or 2 down and 1 left). However the agent does not know which of these routes are really the optimal one. In some cases the light pattern could mean that going left and down would take 5 timesteps and going down then left might take 3. This kind of straying from optimality is hard to notice and even harder to fix due to imperfect knowledge of the environment.

Despite the lack of proof that the agent is acting in the policy that is optimal in time it is pretty clear that it acts in the policy that is close to optimal is distance. Comparing the differences between the optimal policy and the actual number of moves taken. This number is rather small and is below 1 for the best alpha value. This low error means that even though the agent occasionally goes off course it rarely strays to far from the optimal path.

Finally I added code to print out the argmax for each state and unfortunately there are large gaps in the agents knowledge. Many of the states that involve traffic (oncoming or from the left) have all actions set to 4.0 still. This means that that state never happened in the trials, This is the main problems with my agent, it does random actions when it encounters a state it has not encountered before.

Despite the gaps in knowledge and the non certain nature of optimality with respect to timeliness the agent works very well.