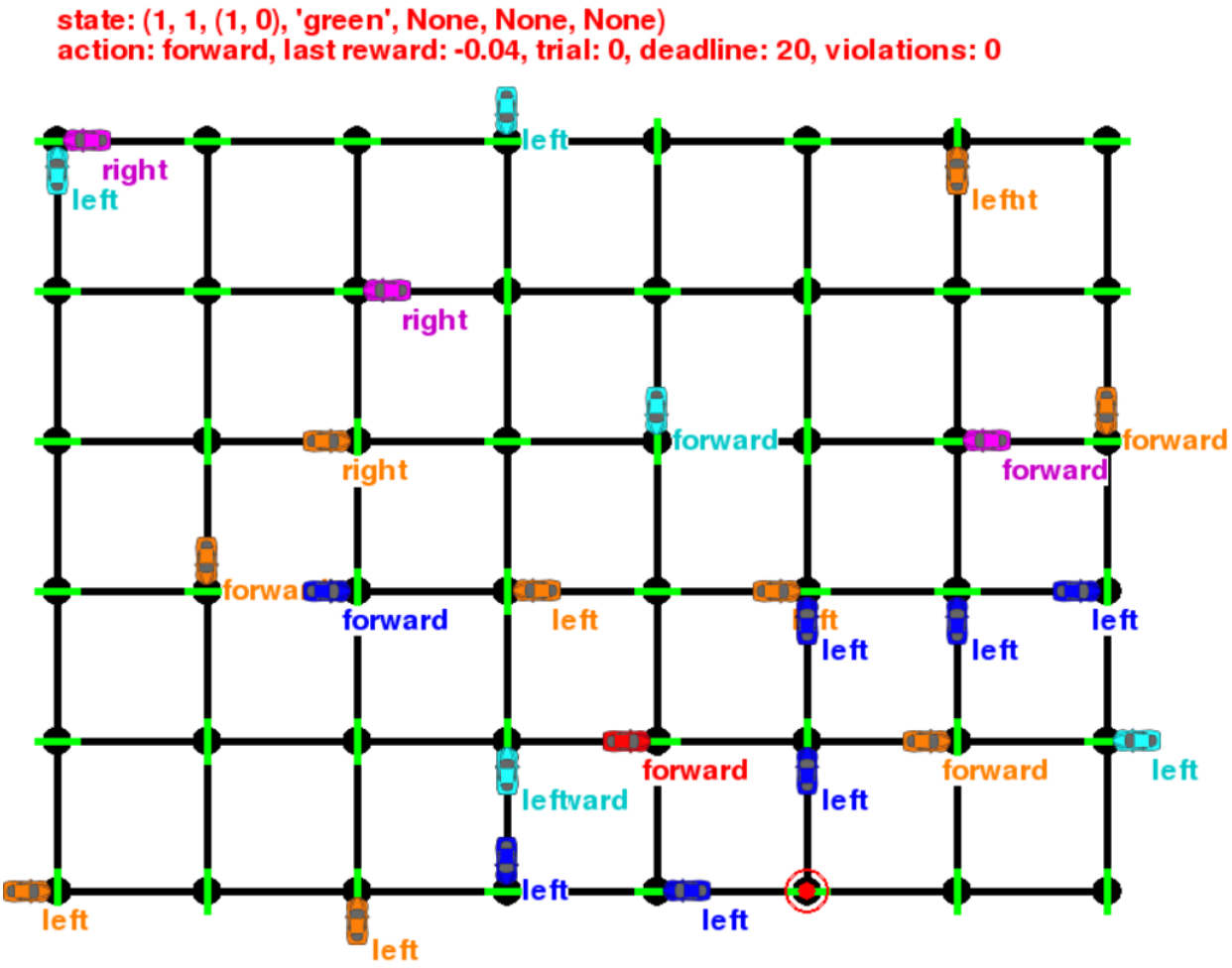
Aaron Brown
Machine Learning
Udacity
7/31/16

Udacity Self Driving Car Project



(Image 1. A red self-driving car agent trying to get to the red circle destination)

Implement a Basic Driving Agent

After setting the agent to random behaviors of moving None, forward, right and left, and setting the enforced deadline to false, it was observed that the agent would eventually make it to its goal, but not efficiently. It was also observed that the agent could never move forward, or turn left when the light wasn't green, so the agent was already following the traffic light rules. One last observation was the agent was able to wrap around the environment, it would be interesting to see if the learning behavior could later take this into account and take advantage of it.

Inform the Driving Agent

The inputs chosen for the driving agent were delta_x, and delta_y, heading, light, and surrounding car directions. The delta x and y values is just a difference between the location and

destination information and should not result in any loss of information. By using a delta distance this should just move the destination to the origin of a graph environment, also the agent should still be able to learn how to use wrapping as an advantage. It's possible to see from a wrapping environment that any absolute value of delta greater than some max will result in the agent trying to go in the opposite direction of the destination in order to try to get there faster by wrapping around the map boundary. Heading, light, and car inputs were all important in order to learn correct traffic behavior. Of course there was also the deadline value, but this could be implemented directly into the reward system by introducing a small negative reward for every move. This helps motivate the agent to get to the destination faster.

The total number of states for the smart cab for the chosen parameters are (-7,0,7) x (-5,05) =(11X15) = 165 for an 8 by 6 grid just using delta x and y values, this was simplified down from 48 x 48 = 2304 if both destination and location coordinates were used. Then there were 4 possible heading states, and 2 possible light states, and the total car direction combinations was [right, left, oncoming] x [right, left, oncoming] = 9. So at any given time there is 165x4x2x9 = 11,880 states that we need to determine an action for. This model would seem acceptable given that we need as much relevant information as possible in order to make a good decision, but at the same time for every variable we add we may need an exponential amount of more trial runs in order to capture the behavior and all the possible combinations. The delta x/y and heading are absolutely necessary for optimizing getting to the destination location. The light, and car directions are also absolutely necessary for avoiding any possible traffic violations.

Implement a Q-Learning Driving Agent

The equation for updating the Q function can be seen below in eq1. When first implementing the Q learning behavior, the first thing that was done was modifying some of the code structure in particular to how the agent was allowed to move in the environment and how it was being rewarded. At first on inspection it could be seen that the agent wasn't actually being allowed to move if it violated a traffic condition, but in our case we wanted the agent to have as free range as possible so it could fully explore. So the agent was given the ability to make any move it wanted and a high negative reward of -10 was placed for any moves that caused a violation. Before the negative rewards for violations were much smaller such as -0.5 and -1. Some other aspects of the base structure of the code were slightly changed too so that the state of the agent reflected exactly what was being seen in the GUI, before it was updating before the environment was done transitioning making it so the agent would appear to run red lights, even when it was being told by the program the light was green.

$$eq1.\ Q(s,a) \overset{\alpha}{\leftarrow} r + \gamma \max_{a'} Q(s',a')$$

The Q learner was using an alpha value that very slowly became smaller over time for each step inside a trial and the discount was set to .3. After all the implementation were made to the Q learner some extra cars were thrown in (30) and the simulation ran for 100,000 iterations. The observed behavior for the agent at first was making random choices and doing terrible but after only 10,000 iterations or so it was

observed that the agent almost always made it to the destination, and after 100,000 iterations had a very small number of traffic violations usually only 1 or 0. As the agent encountered different events and based on what action it took, the reward in essence was saved off in the agent's state mapping so it could remember it next time. To be more precise about the learner's performance, the 100,000 iterated policy was saved off and the program was ran again but this time starting with the Q function with learning turned off and asked to run 1000 times. The result was with this Q policy the agent made it to the destination 92% of the time within the time limit and had around 0.6 violations per trial. Very interestingly if learning was turned back on and repeated for 1000 times the destination time increased to 98% success rate but average violations stayed about the same at 0.6.

Improve the Q-Learning Driving Agent

Having a success rate of 98% and 0.6 violations per trial is very good for the learning agent but can we do better? To find out let's tweak some of the parameter a little bit and compare it with training the previous parameter for 1000 trials and observe which one learns faster by then doing the 1000 iteration test again with learning turned off. The first thing we would like to improve is the number of violations where right now its 60% chance of occurring, really we would like violations to be very rare. To make it so the agent is less likely to cause traffic violations we could do one of two things. The first would be to make the penalty for a violation much higher such as -100, 10 times as much as the current penalty. The second thing could be to restart the game if a violation ever occurs, this would make it so the agent could never get the +10 reward if it ever committed even one violation. Let's try both behaviors and compare the models with everything else held constant.

The base model with all the original parameters was used with training 1000 iterations, it had a 60.3% success rate of reaching the destination and on average had 3.3 violations out of 1000 iteration without learning. The model that had a -100 penalty for violations on the other hand had 47.9% success rate and 3.1 violations on average. So it would appear that the higher penalty had a slightly smaller number of violations but the success rate took a big hit. Next let's try resetting the agent anytime a single violation occurs. After just training with 1000 it was clear that the agent was having a very hard time getting to the destination only .5% success rate and violations were .999 avg. The model still seemed like it was worth exploring so it was ran with 10,000 iterations and had 8.6% success rate. When it was further ran with 100,000 iterations and tested with 1,000 iterations with learning turned off and the reset parameter put back to normal it had a success rate of 90.3% and 0.727 violations per trial. So it would appear that the base reward parameters are the best out of all the reward models explored.

$$\boldsymbol{eq2.}\ \boldsymbol{V \overset{\alpha}{\leftarrow} X}, \qquad \boldsymbol{V \leftarrow (1-\alpha)V + \alpha X}$$

$$\boldsymbol{eq3.}\ \boldsymbol{\alpha = \frac{1}{t^{1}}}$$

The next parameter to tweak are the discount and learning rates, which are currently .3 for the discount ($\boldsymbol{\gamma}$ from eq1) and a very small reduction for the learning rate as a function of time. The definition of the learning rate can be seen above from eq2, and eq3 shows how alpha is currently changing over time. It might be interesting to try an alpha that decreases faster such as 1/t, and also it might be interesting to try a higher discount of .5 and then compare to the previous results. The two

result will be tested separately to get a better idea how each contribute. First alpha was set to 1/t, after running 1000 training and 1000 tests with learning off the success rate was 41.3% with 3.4 violations which was worse than the previous alpha, because the learning was being inhibited. What if the opposite was done and alpha was stimulated by using alpha 1/t^.001? After testing it showed that 1/t^.001 had a success rate of 52% with 3.2 violations once again less than the 60.3% success.

Finally let's try modifying the discount value to .5 instead of .3. The results were 56% success rate and 3.1 violations. If the discount was decreased further from .3 to .1 then the results were 50.5% and 3.7 violations. So in conclusion it seems that the original parameters chosen are the best for fast learning by having the highest success rate and lowest violations. As a final experiment lets run the optimal policy that we found, which was the original base model, a million times (a run time of about 4 hours) for training and see what the final results are for testing on 1000 trials with learning turned off are. The results were the success rate increased from 92% at 100,000 to 93.7% and violations went from 0.6 to 0.518, if learning was turned on for the 1000 training sessions then it was 98.5% success versus 98% previously for 100,000. So from 100,000 to 1,000,000 training examples the training became about 1.7% better for success rate, and .082 better for violations on the 1,000 testing examples not using learning. If we wanted to have more than 99% success rate, and 0.1 violations then it would seem we would need to run at least 5,000,000 more times if looking at a linear trend, in actuality the problem's rate of learning is probably slowing down as more iterations are added.

Conclusions

After some further observations from running the GUI it became clear there was an error in the reward system for traffic violations that was causing problems with right turns in particular. That was corrected and right away the agent could then see how valuable right turns were. Some other changes were to the traffic violation policy, while it was seen that we don't want the penalty to be very large in the previous case it should still be bigger than the magnitude of the positive 10 reward or else if it's the last move, the agent is more likely to cause a violation getting reward 0 other than risking a small negative reward for a more probable valid move. The last change was making it so the initial action was not None but instead decided by looking at the best option. The final result was after training with 100,000 iterations and testing 1,000 the success rate was 99.1% and violations were 0.098 without learning, and 99.4% with 0.06 violations when learning was used. So in conclusion we met our goal of getting an agent that was better at reaching its goal more than 99% of the time and with less than 10% probability of having a violation, where if a violation did occur it was usually only 1. As was done previously we could continue training this successful model even more with 1,000,000 iterations and the success rate would get even better as well as the tendency for 0 violations.