Training a Smart Cab

Joe Sieber

Table of Contents

Abstract

Automated car will be required in the future if society wants to reduce automobile fatalities and increase productive time. Reinforcement learning with Q-Learning can be used to help this process. With reinforcement learning an agent, the smart cab, can learn how to navigate traffic and find the destination.

Q-Learning was successfully implemented and after training the agent learned how to quickly find the destination and travel safely. Several techniques were used to help this process. The learning rate was gradually decreased over time and the agent was allowed to explore more in the beginning.   This helped the agent converge on the optimal policy.

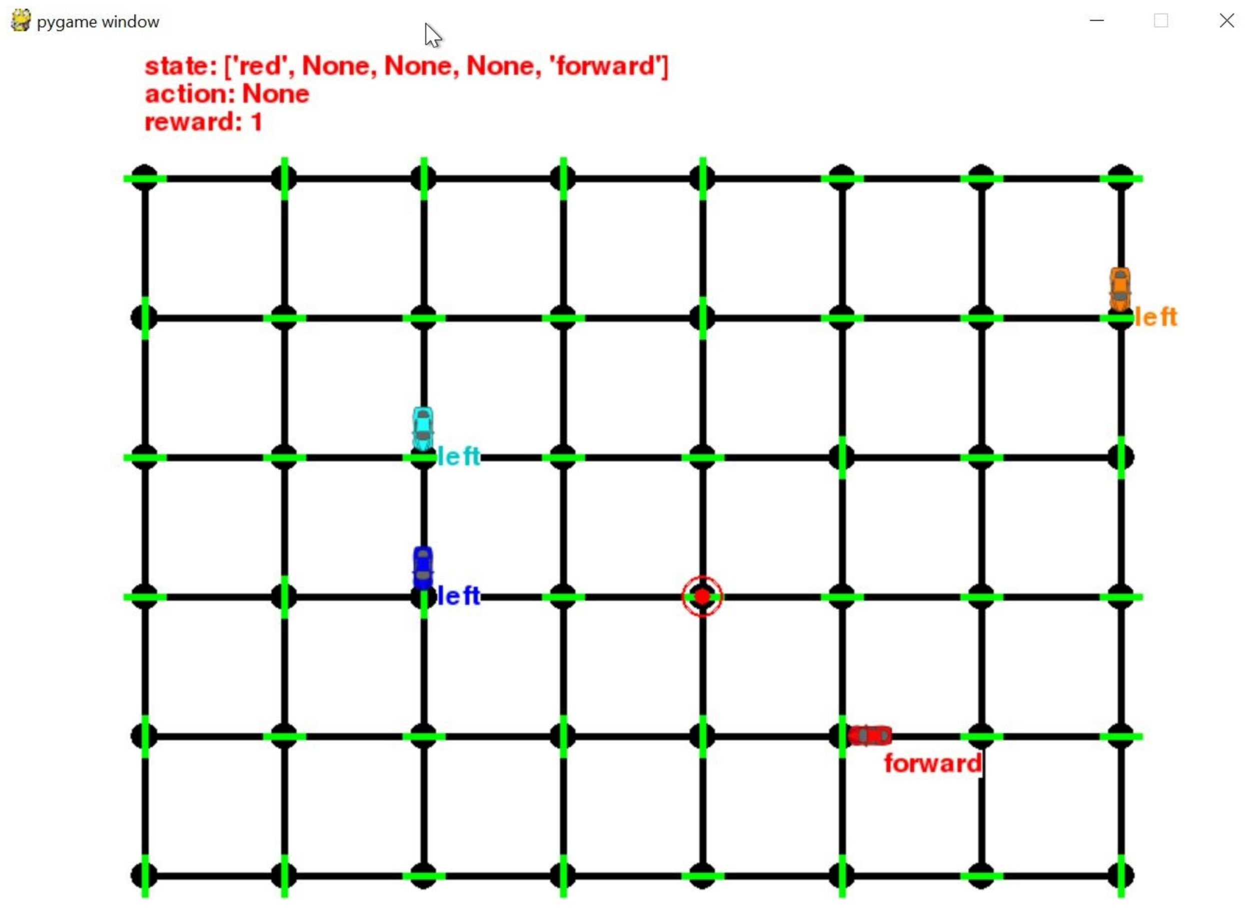
Introduction

With over 90% of all automobile accidents due to human error, there is a need to automate the automobile. This “auto auto” would need to be able to navigate the roadways safely and follow the rules of the road. For example, stopping at stop lights and obeying right-of-way rules. The benefit of this would be enormous to society, not just through the reduction of deaths, but also in freeing up people's time to do other things instead of driving.

In this project reinforcement learning with Q-Learning will be used to train the smart cab to learn the rules of how to follow the rules of the road. At every intersection, the smart cab can move forward, left, right, or make no move. Navigation is already being completed by a guidance system. The smart cab’s task is to take the navigation direction and safely make it to the destination. The inputs will be the color of the light, if there is any traffic and what direction they want to go, and navigation’s recommended direction towards the end point.

Implementing a Basic Driving Agent

To start, the program is initiated without any learning algorithm. This is done to insure that all software is loaded properly. The smart cab is set up to make random moves. The smart cab does make it to the destination but it takes a very long time and it does not follow the navigation's recommendations or the traffic rules. Figure 1 shows the basic grid world for the smart cab. The agent shown in red is the one that will be controlled by the code for this report.



```
state: ['red', None, None, None, 'forward']
action: None
reward: 1
```

Figure 1 - Grid World for Smart Cab

Identify and Update State

Justify why you picked these set of states, and how they model the agent and its environment.

There are several different scenarios that the smart cab can encounter. Table 1 shows all of the variables that the smart cab might encounter at each intersection. Multiplying all of these together we get 384 possible states. For each of these states there are 4 different actions that the smart cab can take. These are move forward, move left, move right, or not move.

Feature	Number of Inputs	Input Names
Light Color	2	Red, Green
Traffic on Left	4	None, Forward, Right, Left
Traffic on Right	4	None, Forward, Right, Left
Traffic Oncoming	4	None, Forward, Right, Left
Next Waypoint	3	Forward, Right, Left

Table 1 - Number of Possibilities for Each Feature

These states were chosen because that is all of the possible inputs that the smart cab might face. Deadline was not chosen as an input because it does not affect the how the car should act at a given intersection. If the deadline is high or low, the smart cab should still try to make the best choice to get to the destination. Based on the reward it receives the smart cab should learn how to follow the traffic rules of stopping at a red light, following right-of-way rules based on the other traffic at the intersection, and which direction to travel based on the next waypoint.

Implement Q-Learning

Q-Learning was implemented into the smart cab in order to help it learn the proper rules. Equation 1 shows the Q-Learning equation. Q(S,A) is the Q value for that state and action. R(S,A) is the reward given for taking that action under that state. Max(Q(S’,A’)) is the max possible future value if that action is taken given a new possible state. Alpha is the learning rate and it defines how much is learned at each step. Gamma is the discount rate and it determines how much emphasis is put on future rewards versus immediate rewards.

$$Q(S,A) \;=\; Q(S,A) + (1 - alpha) * (R(S,A) + gamma * max(Q(S',A')))$$

Equation 1 - Q-Learning

Alpha is initially set to 0.90 but as the agent learns more we want it to continue using what it has learned and update less. Therefore, alpha is decreased as the simulation goes on. NumTrials is the total number of trials that are going to happen, which in this case is 100. CurrentTrial is the trail that is currently being run.

$$Alpha \;= 0.9\ ((NumTrials \;-\; CurrentTrial) \,/\, NumTrials)$$

Equation 2 - Learning Rate Alpha

Since the destination for each game is random and each state that the smart cab could be is the same, gamma was set to zero. From the smart cab’s perspective, moving from one intersection to the next does not change what decision needs to be made at the particular intersection. Higher gamma values makes the agent focus more on long term planning. Due to the constantly random end placement and the smart cabs inability to get navigation guidance beyond the next move, gamma was set to zero. Gamma’s value actually does not affect the outcome of the agent's ability to learn. If it is set to zero or one the smart cab learns at the same rate because the next state that the agent will be in is unknown. The max value for that next state and action is always the same value. For example, Figure 2 shows what the smart cab is aware of at one particular state but the next state could look identitcal. The smart cab is unaware if the next move is the end goal or not.

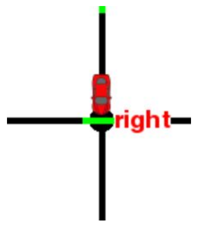


Figure 2 - Random State for Smart Cab

The smart cab also needs to explore more at the beginning of the game to get an understanding of what is expected. As the game goes on at the smart cab has learned more it should explore less. Equation 3 shows the exploration rate. This determines the probability that the agent will make a random move. In addition to the random moves being made by the agent exploration, if that state has not been seen before then the smart cab will also make a random move.

$$Exploration\ Rate\ =\ 1.2\ ((NumTrials\ -\ CurrentTrial)\ /\ NumTrials)^5$$

Equation 3 - Exploration Rate

With all of these equations implemented, the smart cab started out making several mistakes and not doing too well finding the destination. The cab would drive around making random moves for a while. As it progressed along the 100 different trials, the smart cab became very aware of its surroundings and would drive straight to the destination in the shortest possible path.

Enhance the Driving Agent

Several Alpha and Gamma values were tried at random at the start. These did not seem to help reach an optimal policy. After some research, a post on stackoverflow.com, link to article in Appendix A, mentioned that alpha should change over time. This lead to the creation of Equations 2 and 3. A linear decline in alpha and an exponential decline in the rate of exploration was tried. This allowed the agent to find the optimal policy.

After around 75 trials out of the total 100 the agent does seem to find the optimal policy. It reaches the destination in the minimum possible time and does not incur any penalties. At this point it is making very few random moves and has converged on the optimal policy.

Appendix A - References

- Google.com,. N.p., 2016. Web. 19 Feb. 2016. http://www.google.com/selfdrivingcar/images/home-where.jpg
- Stackoverflow.com,. N.p., 2016. Web. 19 Feb. 2016. https://stackoverflow.com/questions/1854659/alpha-and-gamma-parameters-in-qlearning