Implement a basic driving agent

In your report, mention what you see in the agent's behavior. Does it eventually make it to the target location?

The agent wanders around the grid with no apparent direction due to the fact that its actions are selected at random. There is no guarantee that the agent will ever make it to the target location. All directions are uniformly chosen at random, so there is no way of telling how likely it is that the agent will ever reach its destination.

Identify and update state

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

These are the states that can be immediately observed by the learning agent. The agent should pay attention to its future location, whether the light is green and allowing it to move, and if there are cars oncoming from any direction. Some states were unnecessary, however. The oncoming traffic from the right is nonessential if you pay attention to the color of the light. While other agents in the code are hardcoded to learn the right of way rules, the Learning agent will be required to also pay attention to *oncoming* traffic and potentially *left oncoming* traffic. Without acknowledging *oncoming* traffic, the agent can still decide to make a left and will incur a negative reward without having a clear explanation as to why. It could still be argued that the only other input dimensions to add to the state would be *oncoming,* and *light* traffic. That would make sure that the agent learned not to make left turns when there is *oncoming* traffic. *Left oncoming* traffic, on the other hand, is not as essential because the agent would simply be expected to make less or no right turns on red lights. It is helpful to exclude *left oncoming* because it reduces the input space that the agent needs to learn, and therefore speeds up the learning. Adding the *deadline* to the state would add far too many discrete states to explore in the limited amount of time there is to arrive at the destination. It is possible to decrease the number of states in the deadline by referencing fractions of the *deadline* time left as discrete variables.

Implement Q-Learning

What changes do you notice in the agent's behavior?

The agent starts to take the None action more as it observes a +1 for every time it does not run a red light. It is not initially encouraged to find the goal because it receives rewards for following correct rules of the road. It does not receive a huge negative reward for not finding the destination, so it makes sense that it would try to receive multiple small rewards instead of searching and making a mistake. After receiving a few +2s for following through with the *next-waypoint* it starts to take more and more *lefts, rights, and forwards.* Since these waypoints are indicated by the planner, the agent works towards the eventual goal and eventually receives a large reward that will end up contributing to its tendency to follow the *next-waypoint*.

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?

I implemented the ego- and allo- centric models in the tutorial addressed at this link:

https://studywolf.wordpress.com/2012/11/25/reinforcement-learning-q-learning-and-exploration/

Ego-centric q-learning relies solely on the present state and q-value. In other words, the ego-centric model is focused on maximizing immediate rewards. Contrary to the ego-centric model, allocentric q-learning encourages the agent to balance its understanding of the present state with that of future states. In this environment, the expectation is that the ego-centric q-table would quickly learn the right-of-way rules to maximize the immediate reward of a decision. Allocentric q-learning would be expected to focus more on following the planner as that gives a high return that would also be increased by finding the goal. Unfortunately, this did not provide sufficient improvement. The agent quickly learned each table because of the small input space. This did not have any advantages over the basic implementation of Q-learning. The basic implementation of Q-learning with limited and carefully selected states learns an optimal policy very quickly. All other aspects of Q-learning, like the learning rate alpha, gamma, and random action rate epsilon were selected after running several iterations.

I added a fractional representation of the deadline to explore whether the agent would learn to use this as an excuse to gain more rewards before getting to the destination, but it did not. Instead, I used this representation to determine the random action rate epsilon, so the agent would be encouraged to explore until the last fractions of the deadline of the trial. The agent will have a high epsilon for the first half of the time limit, and then be reduced to a very low random action rate after that.

A lower gamma (0.1) gave more sporadic results in the end analysis. Sometimes the agent would reach its destination 8 out of the last 10 trials, others it might reach the destination only 5 out of the last 10. A higher gamma (0.95) would consistently have high overall percentages of reaching the destination and only varies between 8 to 10 of the last ten trials. An analysis of the changes that occur in the q-table shows that the agent reaches a steady state quickly, but less so with a lower gamma and/or alpha compared to a high gamma and/or alpha.

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?

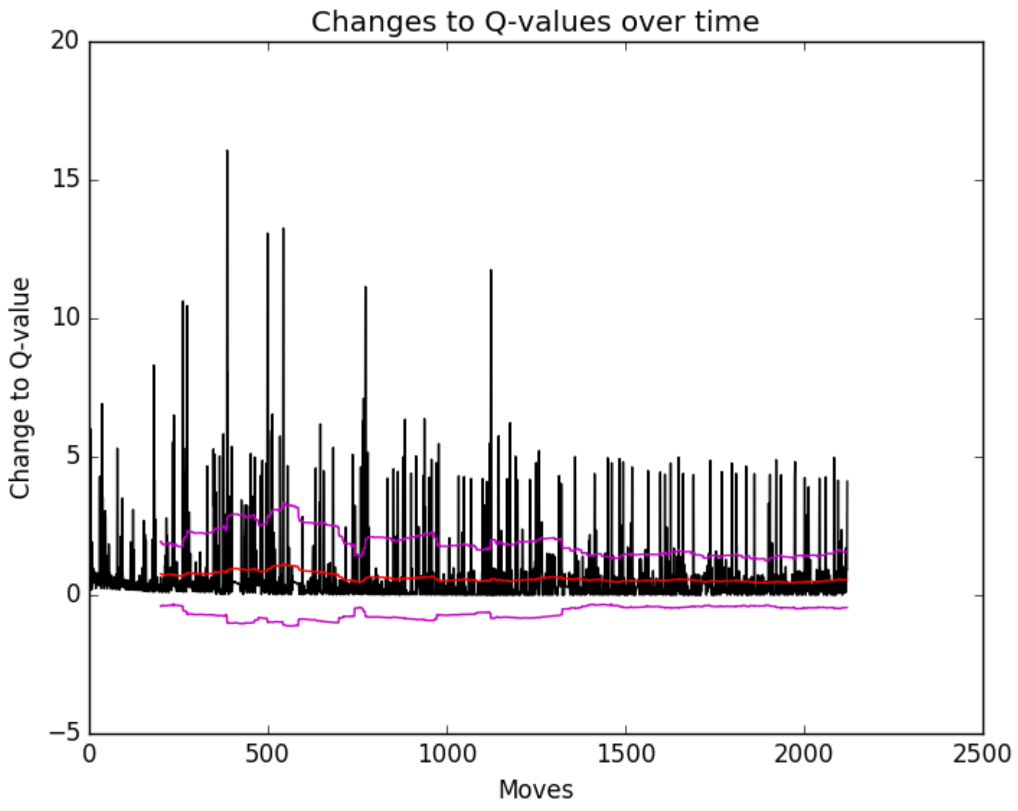
Due to randomness, it is hard to strictly state the performance of the learning agent. Every so often, the agent is programmed to take a random action that might lead to a negative reward. This can help in the learning, but hurts the overall performance. The agent quickly finds a good policy that achieves cumulative rewards per round near 30. This stays pretty stable no matter the changes made to the learning agent. It also reaches the goal in around 80% or more of the trials. In the last 50, 25, and 10 trials it tends to do better than the overall average and does better as the trials approach the last trial.

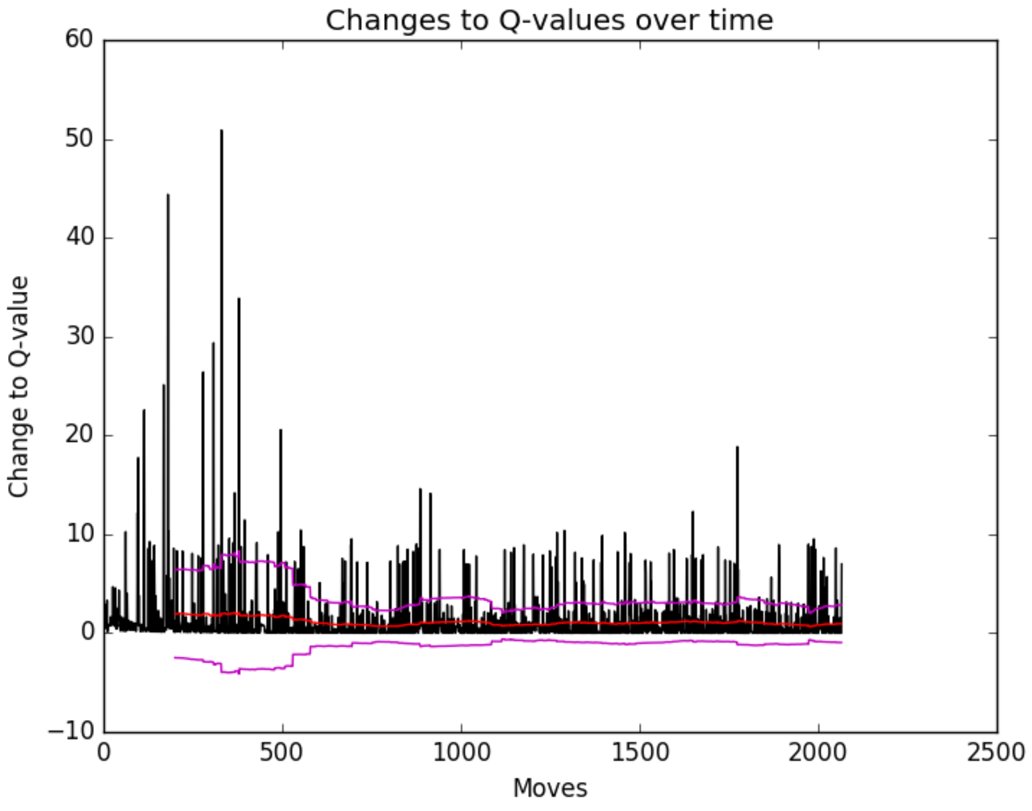
Once again, due to random actions within trials, minimum possible time does not make for a good metric to measure the efficiency of the learning agent.
All analysis was done using the output_analysis.py file.

The output analysis graph of changes over all moves made by the agent in 100 trials reveals that a semi-steady state is reached when you consider a moving average of the results. Obviously a moving average flattens out the variation in the observed data, but a trend is shown that the values start to vary less around the values towards the end of the figure below suggests that the volatility of the agents q values is decreasing. Gamma set at 0.95, and alpha set at 0.5. This learning agent completed nearly 80% of its overall runs and 9 out of the last 10 trials. Similar results were seen for an agent with Gamma=0.95 and alpha=0.3.

The second figure has high gamma and high alpha, 0.95 and 0.9 respectively. In this set of trials, the volatility decreased much more quickly. However, the number of trials completed overall decreased to 75%.

As there is a random action rate, epsilon, in every trial, there will always be penalties incurred until the random action rate is removed.


Changes to Q-values over time
Change to Q-value
Moves
20
15
10
5
0
−5
0
500
1000
1500
2000
2500


Changes to Q-values over time
Change to Q-value
Moves
60
50
40
30
20
10
0
−10
0
500
1000
1500
2000
2500