Udacity Machine Learning Nanodegree
Project Four: Training a SmartCab to Drive
Josh Casale
6/01/2016

1) 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?**
    - In order to implement a basic driving agent, I utilized random.choice over a list of the possible actions. The agent does eventually make it to the target location, but it's just due to random chance.

2) Identify and Update State

- **Identify a set of states that you think are appropriate for modeling the driving agent.**
    - To model my driving agent, I chose to use the color of the light, the presence of an oncoming car, whether or not there is a car present to the left and right of our car, of a car to the right of our car, and the next waypoint.
        - The color of the light is relevant, because traffic rules dictate that drivers must use the color of the light to coordinate our behaviour, else they are punished with fines (or in this simplified model, negative reward).
        - The presence of oncoming, right, and left cars is important because our model punishes the driving agent for colliding with other agents.
        - The next waypoint is critical for informing the behavior of the driving agent, because it inform s the agent (indirectly, through the reward function) what path to take in order to reach the goal state.
- **Justify why you picked this set of states and how they model the agent and its environment:**
    - My agent is concerned with making it to the goal state as quickly as possible, while avoiding collisions and other negative consequences that are built into the rewards system. Because the path planning is already implemented for us, this means that we simply want our agent to decide at each step whether it should attempt to move directly toward the next waypoint, or if there are obstacles that prevent this from being a feasible strategy. In order to do that, our state must capture both the waypoint and any potential obstacles, so those are the features that I chose to include in my state space.
    - With any modeling problem, we have to take the curse of dimensionality into account when choosing our model parameters: for each additional parameter we choose, we'll need about an order of magnitude more training data. Because of this, it is important to choose the parameters necessary to effectively model the problem we are trying to solve, but no more. In addition to the parameters that I chose, the following environment variables were available:
        - Deadline - The number of turns until the deadline could have been used to make the agent more or less aggressive based upon how much time remaining

it has to reach it's goal, but because of the large number of potential values for this parameter(30+, whereas each of the other chosen parameters have value-sets with cardinality less than 5), it would blow up the size of the state space. An optimization here would be to further discretize this set into ranges (0-5, 6-10, 10-15 moves remaining etc), but this would still enlarge the state space, and furthermore, in the simplified environment model, it's not clear that aggressiveness of the driver would affect the performance of the model.

- Location - The location of the agent could be used in comparison to the destination state to inform the driver of secondary waypoints, or alternate paths to the destination, but this would add a parameter of cardinality equal to that of the size of the map, (which is untenable) and as the path-planning aspect of this project is built in to the reward function, even through alternate paths might shorten the trip length, it's not clear that they would result in higher reward totals, so the Q-Learning implementation would not necessarily take this parameter into account anyway.

3) Implement Q-Learning

- **What changes did you notice in the Agent's behavior?**
  - After implementing Q-Learning to inform the agent's policy behavior, the agent mills around for a while while it is the initial stages of the learning process, but quickly begins to direct its movements toward the goal state.

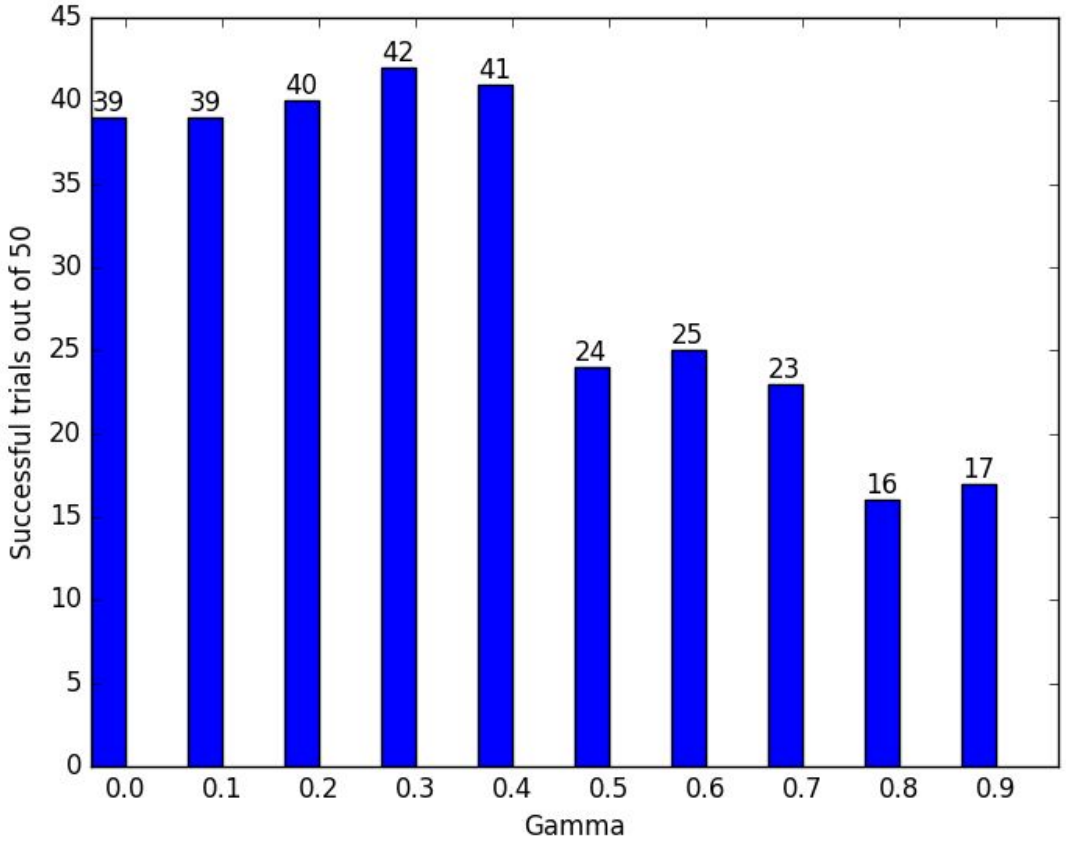
4) Enhance the Driving Agent

- **Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent.**
  - One of the fundamental problems in reinforcement learning problems is the Exploration-Exploitation dilemma. Our agent has two opposing objectives; we want to explore the state space in order to gain the necessary knowledge to inform an effective policy, but we also want to use the information that we currently have to move efficiently toward our goal. If we were to always move randomly through the state space, we would optimally explore the space, and gain knowledge, but we wouldn't move efficiently toward our goal. If we were to always take the action that our current policy dictates as most advantageous, we may fall into local optima and never reach our desired goal state. In my implementation, I address this by using a "Greedy in the limit with infinite exploration" model. My Agent takes random action initially, but as time goes on, my agent relies on its developed policy with more and more confidence, eventually taking a random action only ~6% of the time. This gives us enough state-space awareness to develop an effective policy while utilizing that policy in order to move toward our goal state.

- In a Q-Learning model, the Learning Rate informs the model as to how much weight should be placed on each piece of new information. A learning rate of 0 will cause the agent to ignore all new information, whereas a learning rate of 1 will cause the Agent to disregard all old information concerning a state each time that state is encountered. In my original implementation, I used a constant learning rate of .5, but decided to iterate on this design by decreasing the learning rate with the number of actions taken by the Agent. In my final design, the learning rate is initialized to 1, and at each step is updated to 50/(50-number of actions taken) so that the rate decays slowly, making the agent more confident in its current information as time goes on.
- In a Q-Learning model, Gamma is a factor that determines the weight that the model places on future rewards, commonly referred to as the *Discount Factor.* A discount factor of 0 will nullify the utility component provided by future actions, making the model very short-sighted, whereas a discount factor approaching 1 places a very high weight on future states. I first incorporated a static Gamma parameter of .5 in my model, but decided to iterate on this design and test out a range of Gamma parameters from 0 to .9 in increments of .1, and came to the conclusion that gamma values around .3-.4 performed best.

- **How well does it perform?**
  - As can be seen from the figure below, I tested model performance as a function of my choice of gamma by running 50 trials (with an enforced deadline), and used the number of successfully completed trips out of 50 as my metric. As gamma is increased past .4, we start to see the model's performance degrade, but with a choice of .3 for Gamma, my agent successfully finds the destination within its deadline an average of 42 out of 50 trials, (starting the first trial with a completely

untrained agent) while only encountering two penalties.



- **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?**
    - For the problem at hand, an optimal policy would inform the car to take the shortest possible path from its current state to the destination, without running any red lights and avoiding collisions. In our simplified traffic model where all intersections are four-way, the distance between two points is given by an L2 norm but there are multiple possible paths between any two points (further than one move apart) with the same L2 distance. Thus, the optimal policy would inform the agent to take the action which:
        - Deviates as little as possible from the shortest path
        - Does not collide with other agents
        - Avoids red lights when possible.
    - In our model, following the shortest path, collisions with other agents, and red-light behavior are all accounted for by the reward function, and as such, the best measure of adherence to the optimal path is the average reward per action. In addition to the calculations of average success rate visualized above, I calculated the average reward per action for each choice of gamma over ten separately-trained agent, each running 50 trials, and found that the two measures

agreed on gamma = .3 as the optimal choice. Given this choice of gamma, we see an average reward per action of 1.79; to put this into context, when the agent moves to the next waypoint on its shortest path to the destination, (according to the environment's reward function) it gains a reward of 2, when it doesn't move, it gets a reward of 1, when it makes a non-optimal, but legal, move, it gets a reward of 1/2, and when it attempts an illegal move (running a red light or crashing into another car), it is punished with a reward of -1. Finally, a successful trip gains a bonus of 10 points. The trip-bonus, as well as the stochasticity of the lights in our environment obfuscates our ability to calculate the average reward of the perfectly optimal path, but in spite of this, average reward provides us with an ordinal metric of the 'goodness of fit' of a particular model, as a higher average reward means either fewer mistakes, or a combination of more optimal moves and a shorter overall trip length. Because our agent averaged 1.79 reward per move, where the upper bound would be around 2 (slightly more because of the end-of-trip-bonus) I would argue that the agent comes close to finding the optimal path.