Smartcab project report

Question 1 - Random selection of actions

Observe what you see with the agent's behavior as it takes random actions. Does the smartcab eventually make it to the destination? Are there any other interesting observations to note?
Answer: Chosing random actions at each update, the agent is still able to reach the destination with a certain probability. Figure 1 shows the accumulated number of trials, in which a "dumbcap", disregarding all traffic rules and selecting the next action completely at random, reaches the destination within the given deadline.

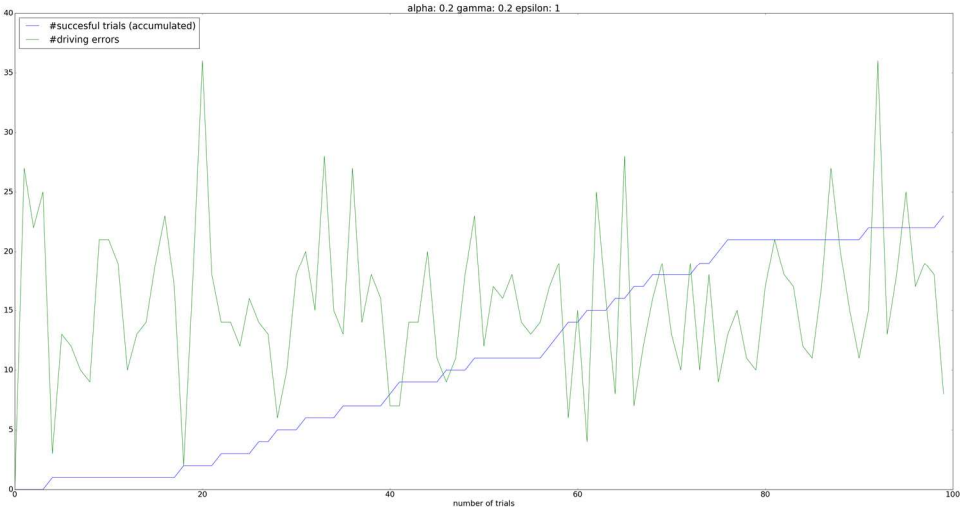

Figure 1: Agent with random action selection („dumbcab“)

Any model-based smartcab should achieve a significantly better performance than the dumbcap represented by figure 1. Specifically, the success-curve of a smartcab that learns to drive perfectly should be a line after a certain number of trials, while the number of traffic violations should be zero.

Question 2 - Inform the Driving Agent

What states have you identified that are appropriate for modeling the smartcab and environment? Why do you believe each of these states to be appropriate for this problem?
Answer: The necessary state information can be derived from the requirement definition for the agent, which can be split into three parts:

The smartcab

- has to be able to reach a given destination
- within a given deadline
- without traffic violations

To learn the traffic rules, the agent obviously needs to be able to relate the negative rewards for traffic rule violations to the relevant information about the surrounding traffic. This information can

be obtained from the *sense()* – method of the *environment* class and comprises the traffic light state and the immediately surrounding traffic to the left, the right and in front.

Figure one shows the performance of an agent using just this traffic information for q-learning:

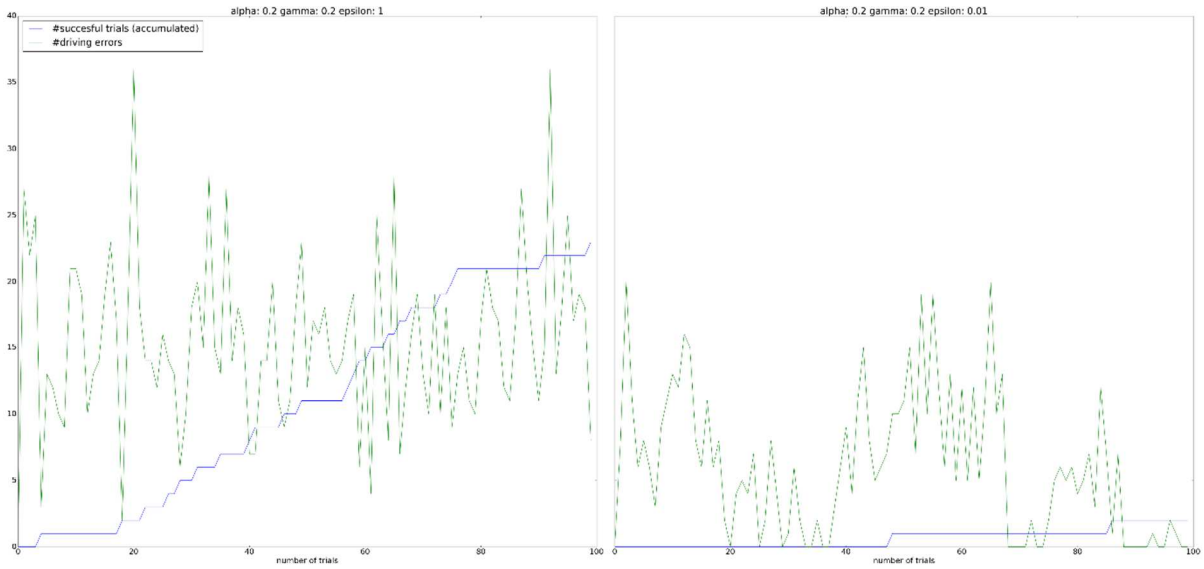

Figure 2: Q-learning-agent with traffic state (right plot) compared to dumbcab (left)

The direct comparison to the dumbcab plot on the left confirms, that the traffic information indeed helps the agent to learn the traffic rules. At the same time it is very obvious, that the agent now only focuses on learning the traffic rules and is significantly worse in getting to the destination then the dumbcab.

In order to learn moving towards the destination, the agent has to be able to relate the positive reward for reaching the destination to the action of moving into the right direction. This information can be obtained from the *next_waypoint()* method of the *RoutePlanner* class, which at every intersection points the agent to a direction which reduces the distance to the destination. With the inclusion of "next waypoint" into the state space, the performance gets significantly better:

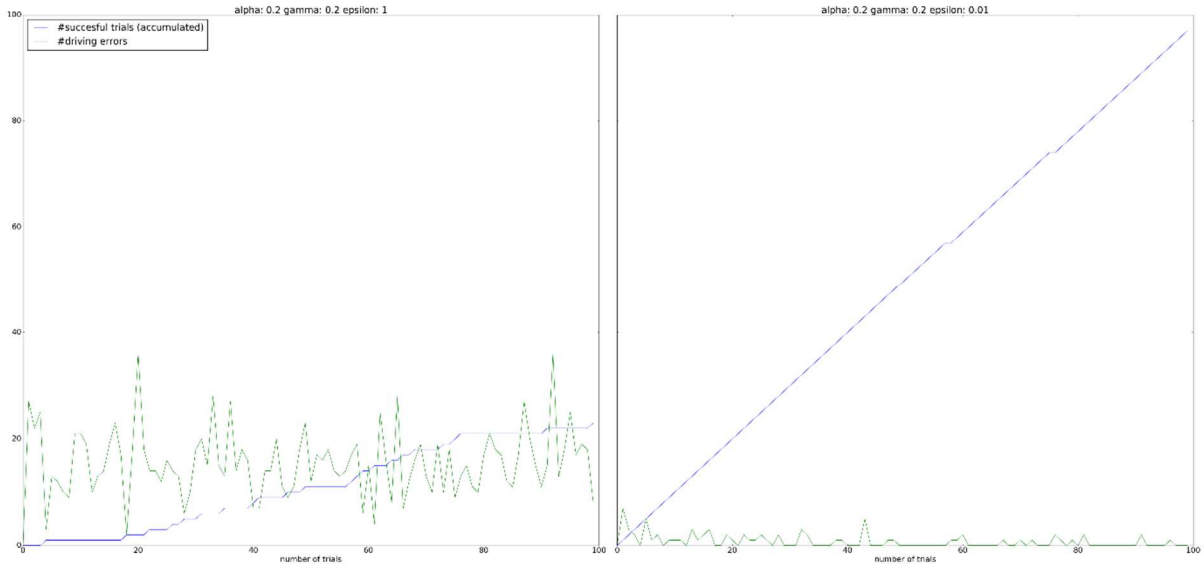

Figure 3: Agent with waypoint information (right) compared to dumbcab (left)

The direct comparison of the performance plots shows, that the agent including waypoint in its state space (represented by the right plot) performs significantly better in both performance metrics than the dumbcab and reaches the destination in almost 100% of the trials with almost no traffic violations.

Although traffic and waypoint seem to be sufficient for learning to reach the destinations reliably and with a minimum of traffic violations, there is one potential state candidate left to consider: The deadline, which can be obtained from the `get_deadline()` method of the `environment` class.

Figure 4 shows the performance of an agent with deadline added to traffic and waypoint information:

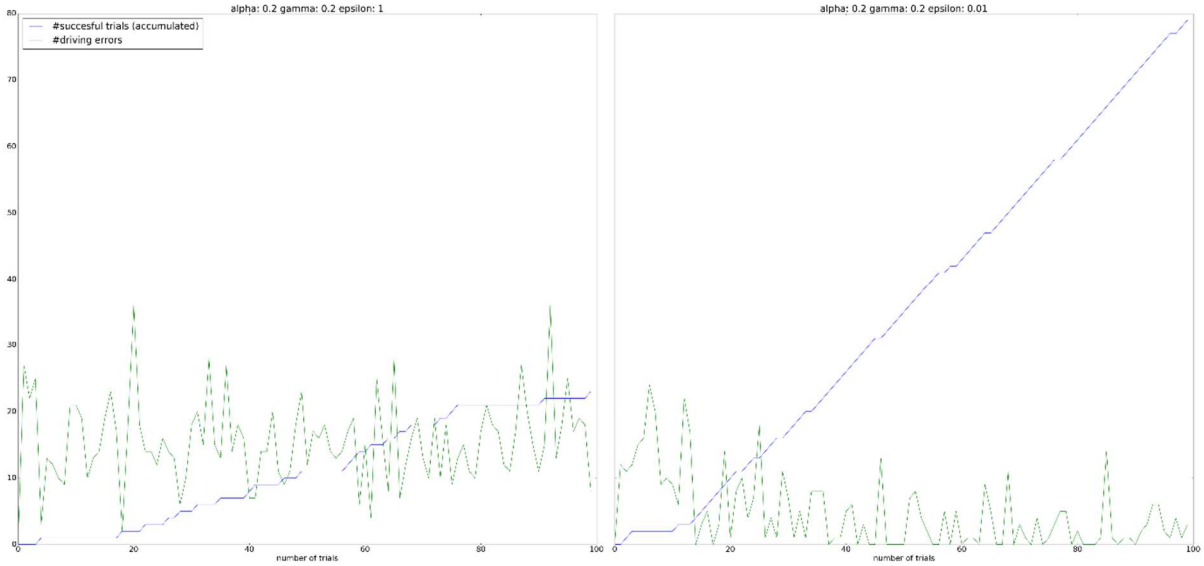

Figure 4: Agent using full state (right plot) compared to dumbcab

The addition of deadline information to the previous state space obviously decreases performance, although it is still significantly better than the performance of the dumbcab. However, when comparing this „full state“ model to the previous model with only traffic and waypoint state, one has to consider the course of dimensionality. As states are added to the state space, the number of required learning iterations increases exponentially. For a fair comparison, the full state model would therefore need considerably more training runs.

Since the state comprising traffic and waypoint information already generates nearly perfect results, there is no need to blow up the state space by adding deadline information and the attention can instead be directed to optimizing the previous model by fine tuning the q-learning parameters.

Question 3 - Implement a Q-Learning Driving Agent

What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?

Answer: The comparison of a q-learning based smartcab with an agent that just choses direction at random reveals that the model-based agent systematically learns to reach the destination within the soft deadline and with a minimum number of traffic violations. The success-curve of the model represented by the right plot of figure 3 is almost a straight line, while the mean number of driving errors goes down as the number of trials increases,

In the pygame live environment it can be observed, that the dumbcab moves around aimlessly, sometimes in circles, but still occassionally reaches the destination by accident. The q-learning agent using traffic and waypoint state information in contrast, learns to move much more systematically from an early stage on and reaches the destination in almost 100% of the trials. With

a low alpha, it doesn't get stuck in local minima, i.e. it doesn't move in circles. Only in very hard situation, when the destination is surrounded by heavy traffic, does it miss the deadline.

Question 4 - Improve the Q-Learning Driving Agent

Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?

Answer: The free parameters in q-learning can be identified as alpha, gamma and epsilon.

The learning rate alpha weighs the update values at each step and prevents the system from getting stuck in local minima.

The discount factor gamma controls the trade off between short term reward and long term gratification.

Epsilon is not used directly in the Bellman Equation, but in the selection of the next action at each update step, where it controls the exploitation-exploration trade off. More precisely, epsilon represents the probability of chosing a random action at an update step, with an epsilon value of 1 producing a completely random choice of action. In an early stage, when the q-table is still sparse or highly unstable, the agent can't rely too much on the values in the q-table and should instead freely explore the environment to gain more experience and populate the q-table. In later stages, as the q-table gets more and more stable, the choice of actions should be much more based on the values of the q-table to allow proper reinforcement of the state-action pairs. Epsilon is therefore set to decrease with $\frac{1}{n}$, where n is the number of the current trial.

The optimal parameter combinations for alpha and gamma are selected by a grid search through the parameter space.

The following figure shows the results of the grid search:

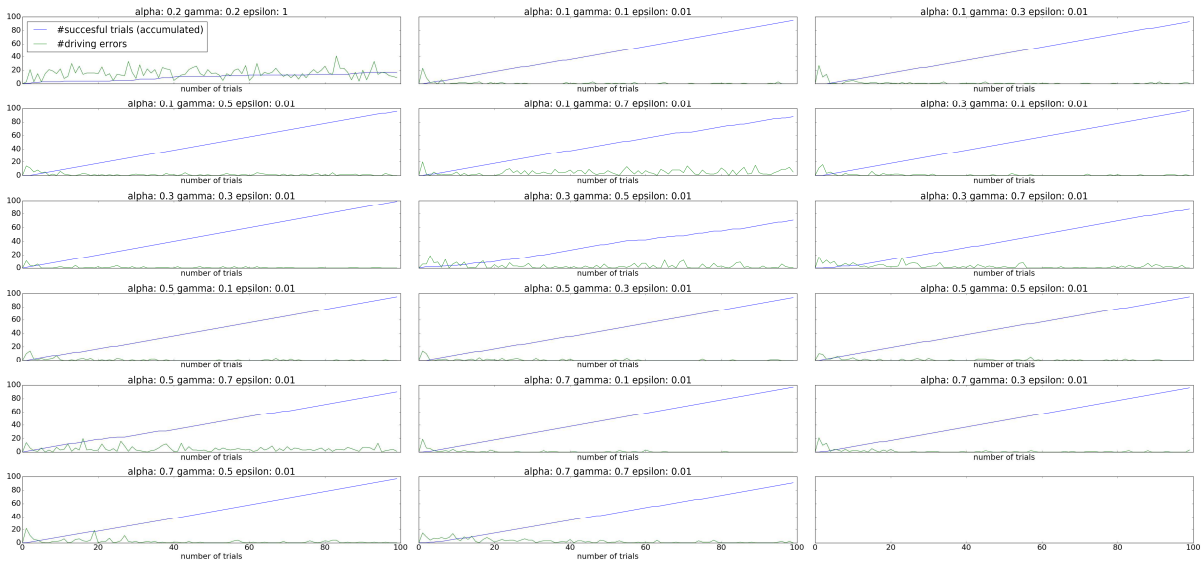

Figure 5: Q-learning with parameter finetuning

In this particular run, the best performance could be achieved with an alpha of 0.3 and a gamma of 0.3, producing a succes rate of 99/100 and zero traffic violations in the final trial.

Question 5 - Finding the optimal policy

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? How would you describe an optimal policy for this problem?

Answer:

An optimal policy for this particular problem could be formulated as follows:

If the next waypoint can be reached without traffic violations, move to the next waypoint. Otherwise, wait.

The agents above get very close to finding the optimal policy, which can be observed in the pygame live environment and is confirmed by the performance metrics provided. The best agent reaches the destination in 98% of the time, with little to no traffic violations. And it gets better over the course of 100 trials, which shows that it gradually learns what it is supposed to learn.