Project 4- Training a smartcab to drive

Implementing a Basic Driving Agent

In this project, we used *pygame* to simulate a world where multiple vehicle agents were travelling around a rectangular 6x8 grid following traffic rules, and a principal agent was assigned the task of getting from point A to point B. The goal was to use reinforcement learning to train the principal agent (also the driving agent, could be representing a smartcab) to accomplish the assigned task in an optimal fashion. In a real-world scenario, optimal fashion would imply getting to the destination in the least amount of time and avoiding any accidents along the way. In our simulation, we will equate "avoiding accidents" to "always following traffic rules". The concept of "least time" is easy to comprehend, but requires a little extra effort to compute due to traffic lights which should be followed but which also introduce stochastic delays that need to be accounted for. We will calculate the optimal duration as the *L1* distance to the destination plus 1 for every time step that the agent has taken fewer than the number of optimal steps and is at an intersection with a red light preventing it from proceeding to the next way point.

The environment provides the following information about the world (in a local sense) to the agent at each turn intersection:

- The next waypoint location relative to its current location and heading.
- The state of the traffic light at the intersection and the presence of oncoming vehicles from other directions.
- The current time left from the allotted deadline.

Random policy

To gain a sense of how the simulation works and establish a baseline against which we may compare our training algorithms, we first explored a policy where the driving agent chose a random action from the set of possible actions (*none*, *forward*, *left*, *right*) at each intersection without considering any of the information available from the environment. We turned off the deadline enforcement rule in the simulation to gather data since we didn't expect the agent to reach its destination consistently based on random choice of actions.

In a typical run we observed the driving agent wander around the grid, generally not obeying traffic rules. On several occasions, it got close to its destination, then wandered off in the wrong direction. It made illegal traffic moves on red lights and also remained stopped at a green light. All of this makes sense as the choice of what the cab does at each waypoint is random when we use a random policy.

Somewhat surprisingly, the smartcab does reach its destination within the hard limit of 100 steps approximately 2/3 of the time. This likely has to do with the dimensions of the simulations, i.e., number of intersections, other agents also moving simultaneously and number of available actions. The fact that the simulation manages grid positions to wrap around (when the agent wanders too far to the right and goes off the grid, it re-enters the grid on the far left) may also contribute to the frequency with which the agent reaches its destination.

The left panel in figure 1 below shows a plot of the percentage of failed moves and traffic violations for a series of 100 runs. The solid blue line shows all instances where the agent received a penalty (i.e., a failed move); the red circles indicate the number of traffic rule violations (penalty = -1.0) in each trip. Clearly, runs at later times do no better than ones early in the simulation and no learning is evident.

The right panel in figure 1 shows the ratio of the actual trip duration to the optimal duration. In an ideal scenario, this ratio would be 1.0. We see that with a random policy, this metric is widely distributed and its average value of 12.9 is much greater than 1 (indicating most trips were not optimal even when the agent succeeded in reaching its destination).

We will use the following three metrics as we improve on the random policy using Q-learning:

- the number of demerits the agent (this should be as low as possible, although some larger numbers may be acceptable in the early stages of the simulation)
- the number of traffic violations (this should be zero after the learning stage)
- the ratio of actual trip time to the optimal time (this should be correlated with the first measure in principle but provides a different way of visualizing the results)

Figure 1. Random Policy with no deadline enforcement

One final part of our exploration of the random policy was to enable the *enforce_deadline* flag in the simulation. This has the effect of limiting the number of steps taken in any trip to the allotted time as opposed to the hard limit. Thus there are fewer traffic violation (figure 2, left panel) and the average trip duration (right panel) is only 3.5 times the optimal time – both artifacts of the fact that enforcing the hard time rule allowed the agent to move longer

per simulation whereas with deadline enforcement, the maximum number of steps is limited to the allotted time limit. In the data for these plots, we do not account for any failed trips.

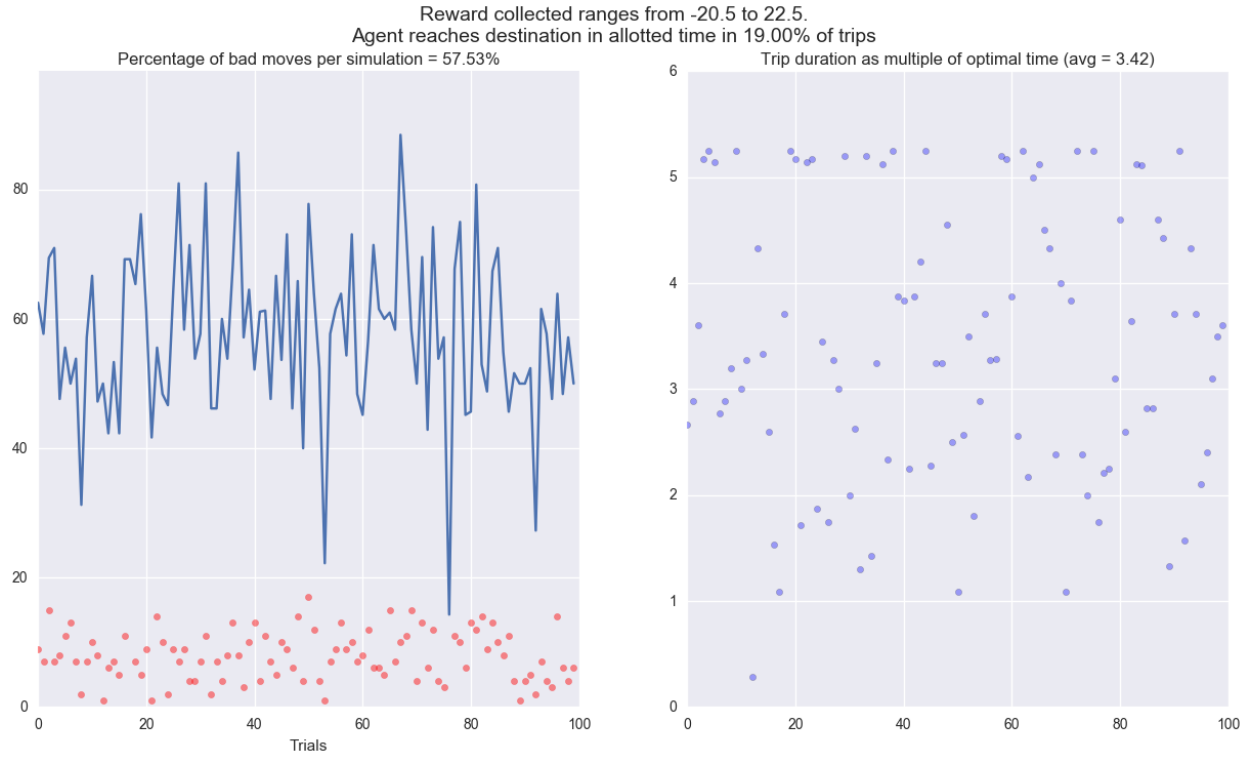


Figure 2. Random Policy with deadline enforcement

Interestingly, there were a few trips in which the ratio is *less* than 1.0. In watching the agent move around on the grid, we saw the agent make moves that would take them outside the limits of the grid and was put back at the opposite side of the grid. Since the *L1* distance calculation does not incorporate this logic, we believe the low numbers are a result of the agent reaching a destination on the other side by accidentally going around the universe.

These results are to be expected of a random policy and simply help establish a baseline for the improvements to follow.

Identifying a set of states for Q-learning

At each position, the principal agent's location (relative to destination), heading, remaining part of time allotted for the trip, signal state (red or green), next waypoint from the planner and traffic condition (oncoming, left or right) constitutes the complete set of variables that could be used to identify a state. The success of our Q-learning exercise will depend significantly on our choice of states in our model. Choosing too many states will result in learning taking unacceptably long; choosing too few will limit the quality of the model and the ability of our learned policy to discriminate between intrinsically different scenarios.

We want our agent to be able to learn the rules of traffic in effect during the simulation and we also want the agent to get to its final destination in the smallest number of steps. US traffic rules allow us to ignore traffic from the right under most circumstances which raises

the possibility of ignoring the state of traffic from the right. Using this intuition, we combine the raw inputs for signal (*red* or *green*) and traffic conditions for oncoming and left (2 directions, each with possible values *none*, *forward*, *left* or *right*) and ignore the traffic from the right. Combining these inputs with each of the three possible values for the next way point (*left*, *forward* or *right*) would give us 2x4x4x3 = 96 states. We have two additional inputs, the time remaining and the heading. Heading is not a relevant feature in the location agnostic world of the agent and we therefore ignore it. The time remaining has a large range of values (from 1 to 50 or so). However, the agent should not behave differently based on the specific amount of time remaining (e.g., 9 vs 10), but it might find it more rewarding to make different decisions when the time remaining is low vs high. We created an engineered binary feature corresponding to the remaining time being less than an arbitrary value to encode such a dependency and used that as a feature of our state space. This results in a total of 96x2 = 192 states. The arbitrary value we picked as the separator of low vs. high time remaining is 4.

For a simulation grid that is 6x8, the maximum L1 distance is 14. With typical allotted time for a random trial ranging from 25-50, an agent could expect to sample state space between 1400 and 5000 time in each run of 100 trials. Since this is on the order of 10 times the number of states (200x10=2000), we can expect that our agent will learn the optimal policy within the constraints of 100 trials. Also, the preliminary runs showed relatively little interference from other traffic so we may only end up visiting 15% of all states as the traffic conditions will mostly be *none* for the two directions we selected (so approximately 1/8 or 12.5% of available state space) and therefore get very good learning over the set of states we do visit.

We argue that rewards for following the directions of the planner should be sufficient to allow the agent to learn to choose an action that follows the planner when traffic conditions allow it and achieve the goal of collecting the most reward while avoiding "accidents". Stated another way, we expect the learned policy will place heaviest weight on following the planner, leaving the task of reaching the destination in the least amount of time up to the planner.

Implementing a Q-Learning Driving Agent

We implemented the greedy Q-learning algorithm with learning rate α and discount rate γ:

$$Q_{k+1}(s,a)=(1-\alpha)Q_k(s,a)+\alpha(r_k+\gamma\cdot\max_{a,s'}\{Q_k(s,a,s')\})$$

The values of the q-matrix were initialized to zero for a simulation of 100 trips with α=0.5 and γ=0.7. In implementing the Q-matrix, we need to know the state the agent ends up in after a move and one of the pieces of information we need to determine the state is the next waypoint from the planner. In the final move of the trip, the planner cannot provide a way point (as there is none) so we used a random choice for this datum. Also, when a state had more than one Q value equal to the maximum, we chose the action randomly from the set of actions with the maximum Q value.

The agent starts out moving around the grid randomly as it did previously, but after a short time, its movements appear more and more deliberate. One characteristic that persists through the simulation is that the agent has a small tendency to drive in a loop, usually in the clockwise direction (i.e., several consecutive rights turns). On some occasions, such loops prevent the agent from reaching its destination. Toward the end of the simulation, this tendency to loop is manifested less frequently.

Q-learning with greedy selection for learning rate (α=0.5) and discount factor (γ=0.7)

We ran 20 sets of 100 trials each with $\alpha = 0.5$ and $\gamma = 0.7$. The Q-matrix was reinitialized after each set of 100 trials and the results are shown in Figure 3. The blue line in the left panel is the average percentage of moves that incur a penalty in 20 runs for each of the 100 trials in a set. In the same panel, we also display the total number of traffic violations (rewards of -1.0) in the 20 runs using red dots. Thus a red dot with a value of 40 means there were a total of 40 violations in 20 runs (average of 2 per run). We see that the greedy choice with learning does remarkably well. The agent reaches its destination in ≈90% of its trips. It still makes many moves that incur demerits (≈12% of all moves), but the number of demerits decreases noticeably in the first 5-10% of the trips when the agent is still learning. The agent makes very few moves that violate traffic rules after the initial learning stage (less than 1 violation every 5 trials). We see relatively large values of the average of trip-time to optimal trip duration ratio (≈2) in the learning phase (right panel), which improves somewhat but has a fair amount of variability even near the end of the 100 trials. The results are clearly much better than what we observed using the random policy and the agent shows consistency but the learned policy is not yet optimal as the trips are still taking longer (> 50% longer) than the optimal time.

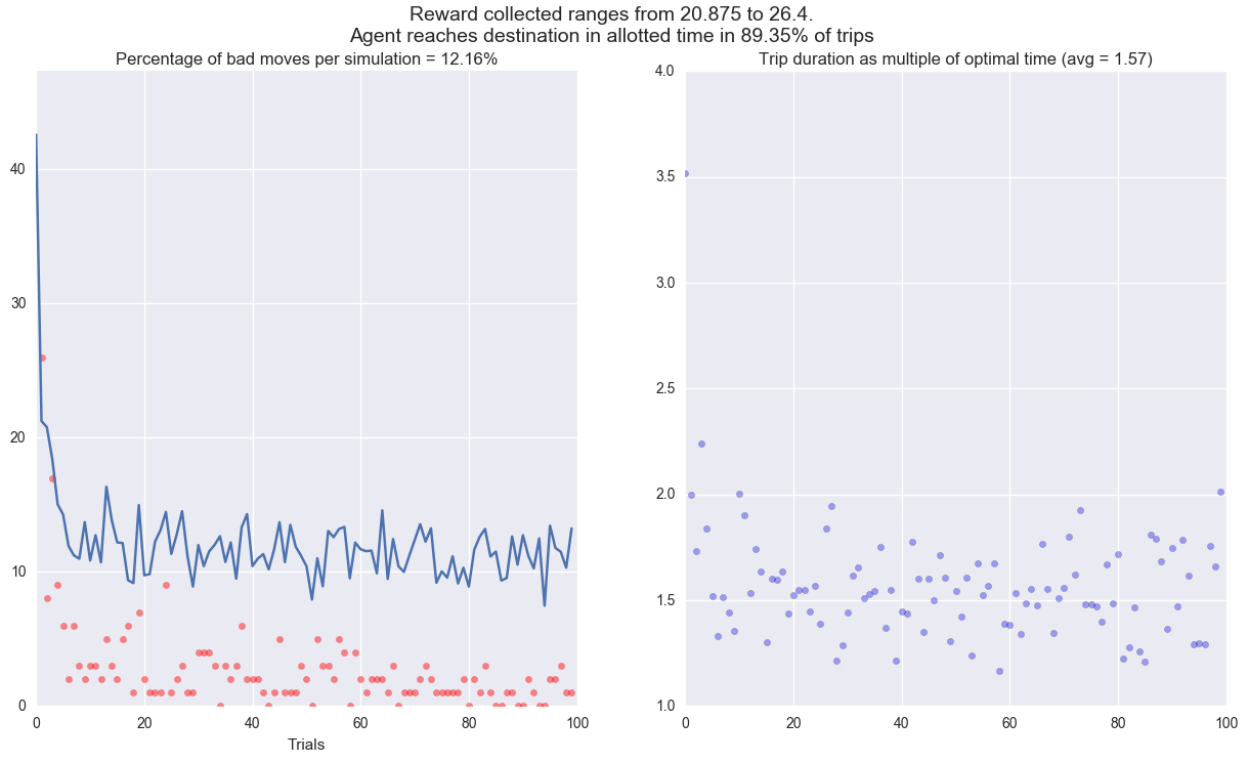


Figure 3. Greedy Action Selection with Q-learning (alpha=0.5, gamma=0.7)

Optimizing learning rate (α) and discount factor (γ)

In the previous section, we arbitrarily selected values for α and γ of 0.5 and 0.7 respectively. In order to find a combination of values for the learning rate and the discount rate, we ran a grid search of the parameter space. We ran a set of 20 repeats of

simulations of 100 trips each for all combinations of values of α and γ between 0.1 and 0.9 (total of 20x9x9x100 trips). The Q-matrix was initialized after each simulation so that subsequent simulations would not benefit from earlier ones.

The results are summarized in a series of heat-maps. In each set of maps in Figure 4-c, we show different metrics averaged over 20 trials. α values are plotted along the y-axis and γ values are plotted along the x-axis. The left-most panel includes all 100 trips for each simulation; the middle panel excludes the first 5 trips (5%, early learning) and the right-most panel excludes the first 20 trips (20%, stabilized learning). They generally show metrics improve as the simulation progresses, i.e., the agent learns "optimal" behavior after a small number of trips.

We reviewed six metrics – the total reward (Figure 4a top row), optimal reward rate (Figure 4a, bottom row), demerits (Fig 4b, top row), violations (Fig 4b, bottom) and excess trip length ratio (Fig 4c, top) and success rate (Fig 6c, bottom).

Q-learning primes the agent to select an action that should maximize total reward. As such, we would be drawn toward selecting a combination of learning ratios from the highest reward zones in 4a (top), e.g., γ = 0.9, α = 0.5. The bottom row in Figure 4a displays the percentage of runs that were optimal – where there were no demerits and the agent reached its destination within the minimal time (L1 distance plus delays due to red lights). This ratio is highest for low γ and α in the middle of its range. This suggests that considering the maximum reward will push the agent into behavior that increases the trip length to collect more rewards within the allotted time for a trip.

Figure 4a. Heatmaps of rewards and optimal reward ratio for tuning learning rate and discount rate

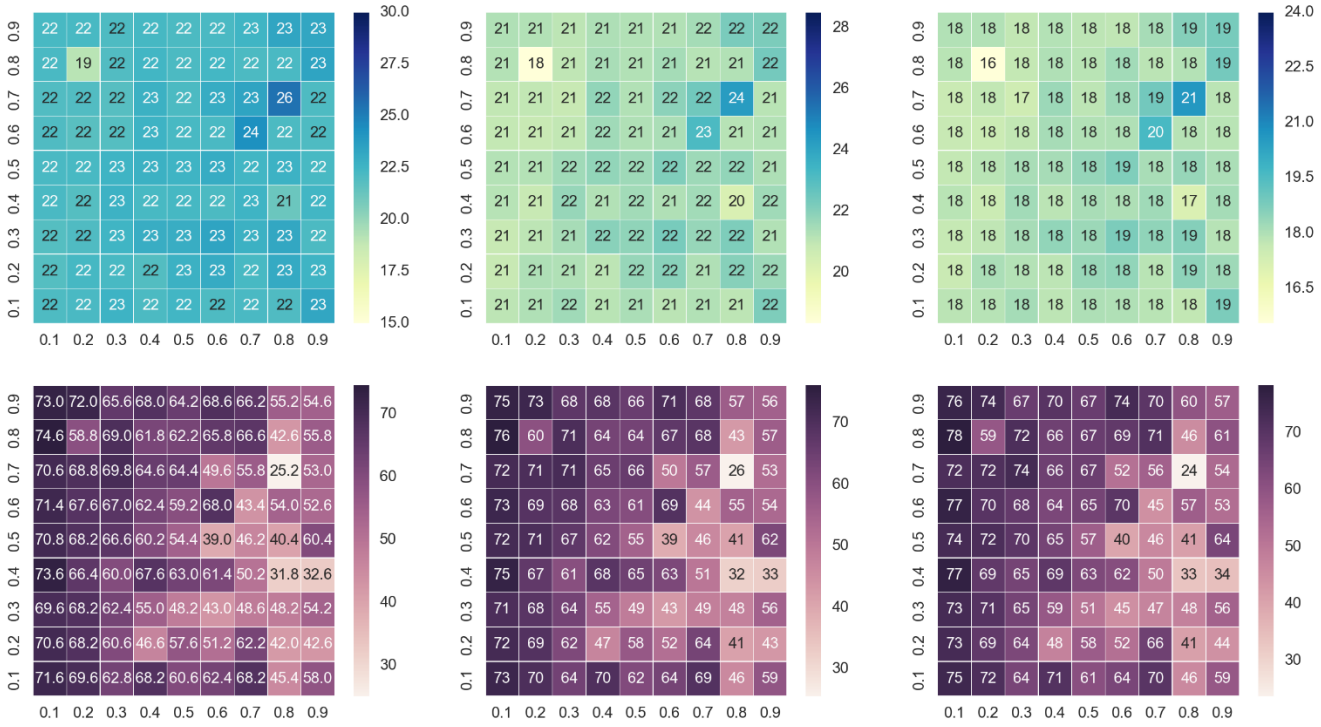


Figure 4a. Heat maps for optimizing Q-learning parameters – rewards

Figure 4b maps demerits (percentage of steps that incurred a demerit) and violations (the number of times the agent violated a traffic rule in 100 trips). For example, with α=0.6 and γ =0.6, the percentage of steps that earned a demerit goes from 42% for all 100 trips, to 25% for the last 95 trips and 20% or the last 80% of trips. Similarly, the same combination of learning parameters, the number of traffic violations go from 16 for all trips, to 10 after early learning and finally to 7 for the last 80% of the trips. Clearly, Q-learning results in fewer demerits and violations. The two maps are correlated and suggest that mid-range values of α and low value of γ would lead to the least number of demerits or violations.

Figure 4b. Heatmaps of demerits and violations ratio for tuning learning rate and discount rate

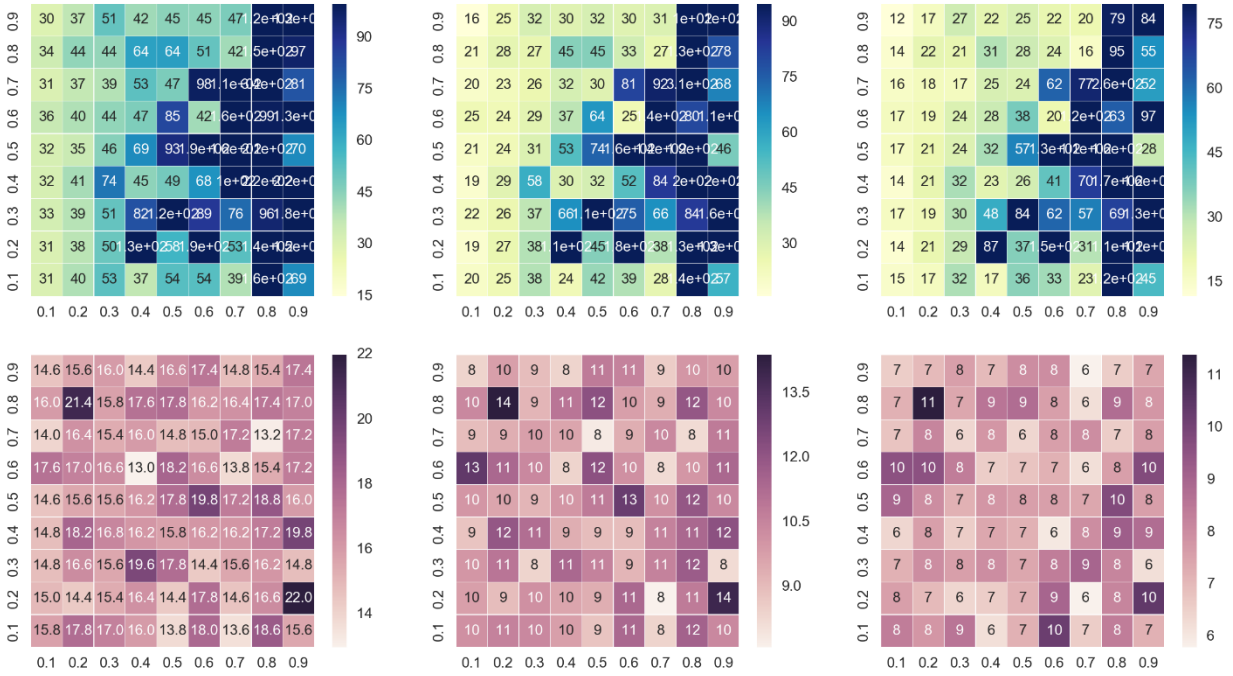


Figure 4b. Heat maps for optimizing Q-learning parameters – demerits and violations

Figure 4c displays the results for the excess trip length ratio. This metric is calculated by dividing the difference total number of steps and minimum number of steps in a trip by the minimum number of steps. The minimum number of steps were adjusted for traffic delays by adding 1 to the *L1* distance for each time the agent encountered a red light and the next way point was either *forward* or *left*. A value of 0.0 is the expected value if the agent's behavior is optimal; a value greater than 0 indicates that the agent had some demerits, such as making right turns on a red light even when the next waypoint was not '*right'*. This ratio should give us insight into agent missteps and the closer the value is to 0, the closer to optimal the agent's behavior. As might be expected this map is correlated with demerits, and indicates that low values of γ would be best to minimize the excess trip length ratio.

The final metric we consider is the success rate, displayed in Figure 4c (bottom row). The success rate is simple the percentage of times the agent reaches its destination in 100 trips. This metric also suggests a low g value for maximum success rate, but it is interesting to note that some high values of g do result in relatively high success rates.

Figure 4c. Heatmap of excess trip length ratio and success rate for tuning learning rate and discount rate

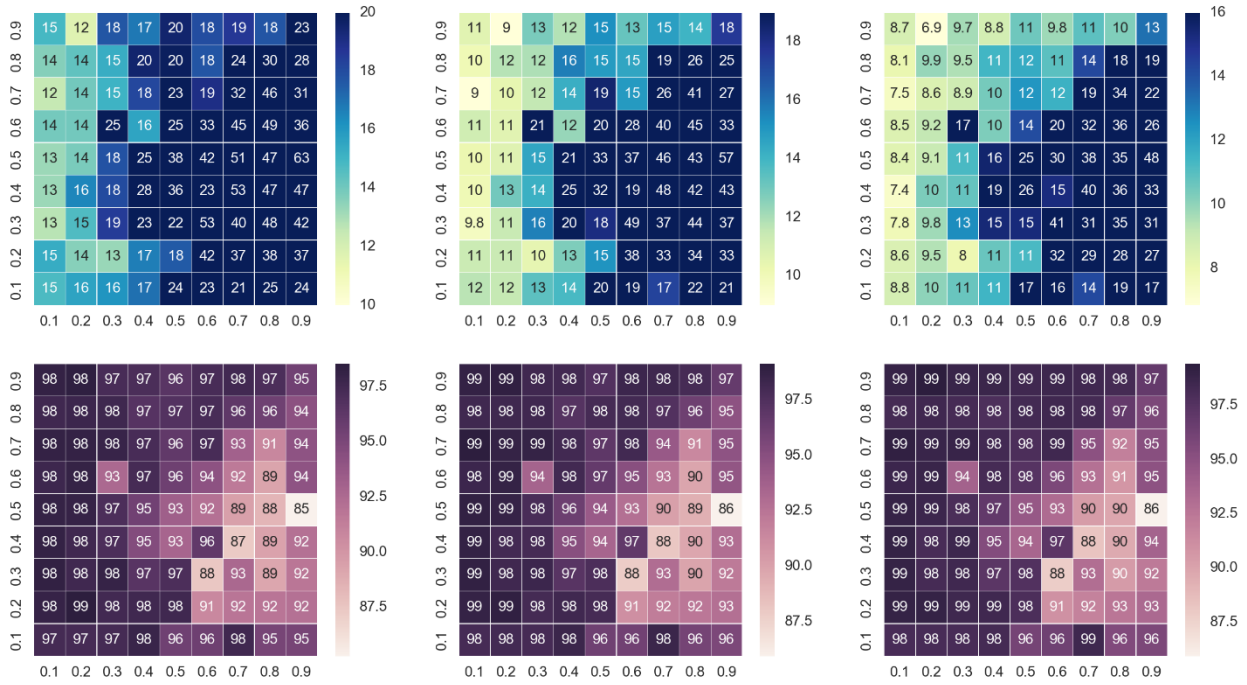


Figure 4c. Heat maps for optimizing Q-learning parameters – trip length

We chose to optimize the parameters to make the agent behave "optimally" – fewest demerits and violations, shortest trip instead of maximizing total reward. The values of α appear to have a weaker effect on the agent's performance while higher values of γ lead to poor performance. We chose α=0.4 and γ=0.1 as the best combination of learning rate and discount rate parameters and ran another another set of 20 simulations.

The results of the simulation with α = 0.4 and γ = 0.1 and shown in figure 5. The agent made it to its destination in 98% of the trips! The left panel shows that on average, the agent made a move that incurred a penalty about 2% of the time (compared to 7.5% for the earlier simulation), made moves that were traffic violations in less than 5% in all 100 trips after the initial learning phase. Also, the right panel shows that the average trip duration was higher than the optimal time (average trip was 15% longer than optimal). The variation in the trip duration is generally small indicating that the agent's behavior is consistent and close to being optimal.

We see an initial rapid drop in the percentage of bad moves followed by a much slower but noticeable decrease (improvement) over time. (except for the number of traffic violations), which would be consistent with small values of γ favoring immediate reward. This is also consistent with our implicit model, where we have generally attempted to make the agent follow the planner so the agent is not learning how to reach its destination, but to consistently follow the directives from a planner.

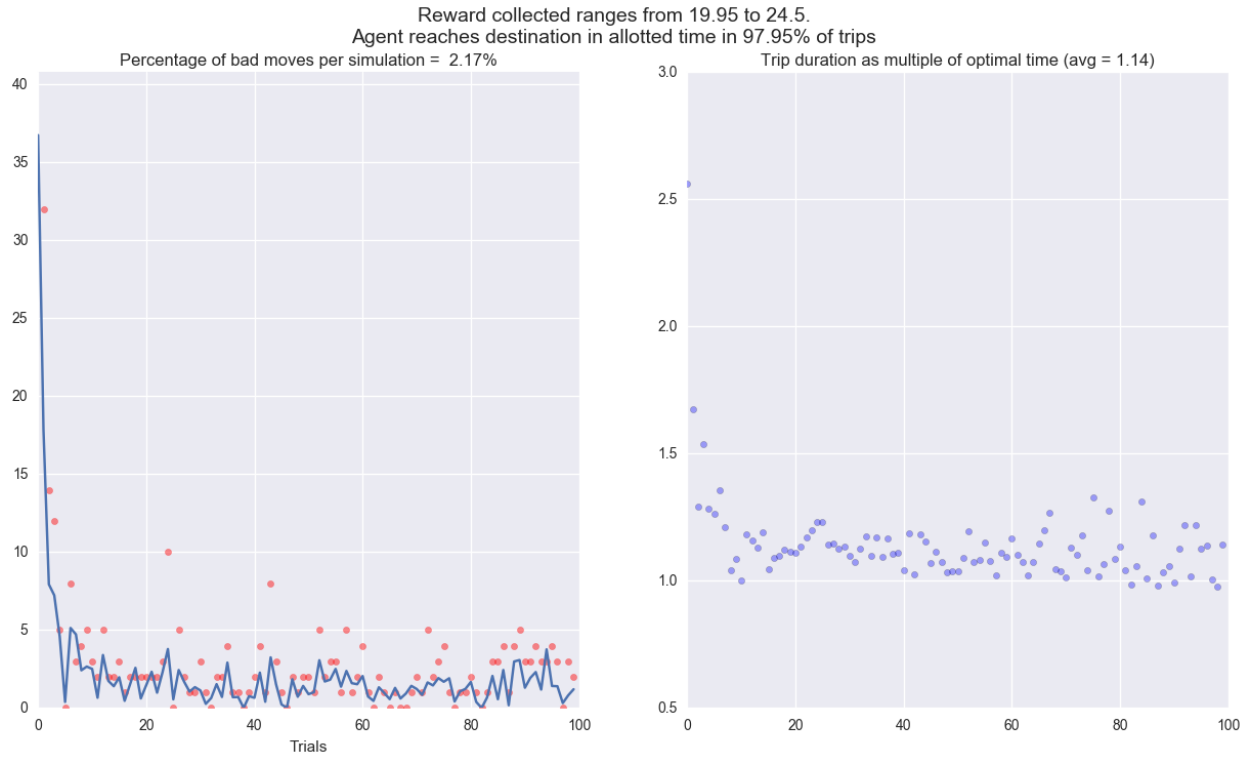


Figure 5. Greedy Action Selection with Q-learning (alpha=0.4, gamma=0.1)

Exploring the effect of using ϵ-greedy Q-learning

Research on reinforcement learning *[1]* has shown that a temporal difference learning algorithm employing purely greedy action selection results in fast learning. A variation of this that includes a probabilistic option to sometimes choose the non-greedy action (ε-greedy) can lead to slower learning but better long term rewards. The rationale for this is that greedy action selection favors early paths to high rewards, whereas ε-greedy action selection achieves a better balance of exploitation vs. exploration and allows the agent to explore other paths that may lead to a better overall policy.

We modified our policy to encourage exploration by using a Gibbs distribution for selecting actions. Stated briefly, actions are selected randomly using $e^{Q_k(s,a)/\tau} / \Sigma e^{Q_k(s,a)/\tau}$ as the weight, where $Q_k(s,a)$ is the Q-value for action *a* from state *s*, and τ is a factor called the temperature. For large τ, the weights are approximately equal and selects actions in a way that is close to purely random. At the other extreme, a small τ results in selection of the action with the highest Q-value for state *s*, i.e., the greedy selection. As the simulation progresses, we lowered the temperature based on the number of visits (*n*) to that particular state using $\tau = max(\tau_{min}, \tau_{max} \times exp(-n/\beta))$, where τ_{min} and τ_{max} are parameters corresponding to the minimum and maximum temperatures and β is a variable we can use to determine how quickly the temperature changes with the number of visits to any given state. We chose τ_{min} =0.02 and τ_{max} = 10.0 to ensure that typical Q values in our simulations would allow for reasonable probabilities. For example, for a state with values (0.0, -0.5, 0.0, 2.0) that had been visited two times (n=2), a β value of 5 gives us normalized weights (0.234, 0.217, 0.234, 0.315) which slightly favors the optimistic option, but with n=20, the weights (0.017, 0.006, 0.017, 0.96) strongly favor the greedy selection.

The results using this method are underwhelming (figure 6). With a small value of $\beta = 1$, chosen to allow the agent to pick the non-greedy option until 5 visits to a specific state, the agent shows slight improvement over the greedy action selection method over the course of 100 training examples. The agent starts out making more errors initially, but then performs about the same as the greedy algorithm on average, perhaps accumulating a few more violations as it tries to explore from states the simulation has not visited previously. Interestingly, the range of variation in trip length appears to be smaller after learning has stabilized than in the pure greedy scenario.

Figure 6. ε-Greedy Action Selection with Q-learning (alpha=0.4, gamma=0.1).

The optimized greedy Q-learning algorithm allowed our driving agent to learn a policy very close to the optimal policy very quickly. We did a "stress test" and ran the simulation with the purely greedy setting and 35 other agents to mimic a scenario with heavy traffic. Even in this case, the agent reached its destination >95% of the time, but incurred more demerits and traffic violations. We see some improvement with time -- the frequency of traffic violations gets smaller, the trip-time to optimal time ratio gets closer to 1 and more consistent.

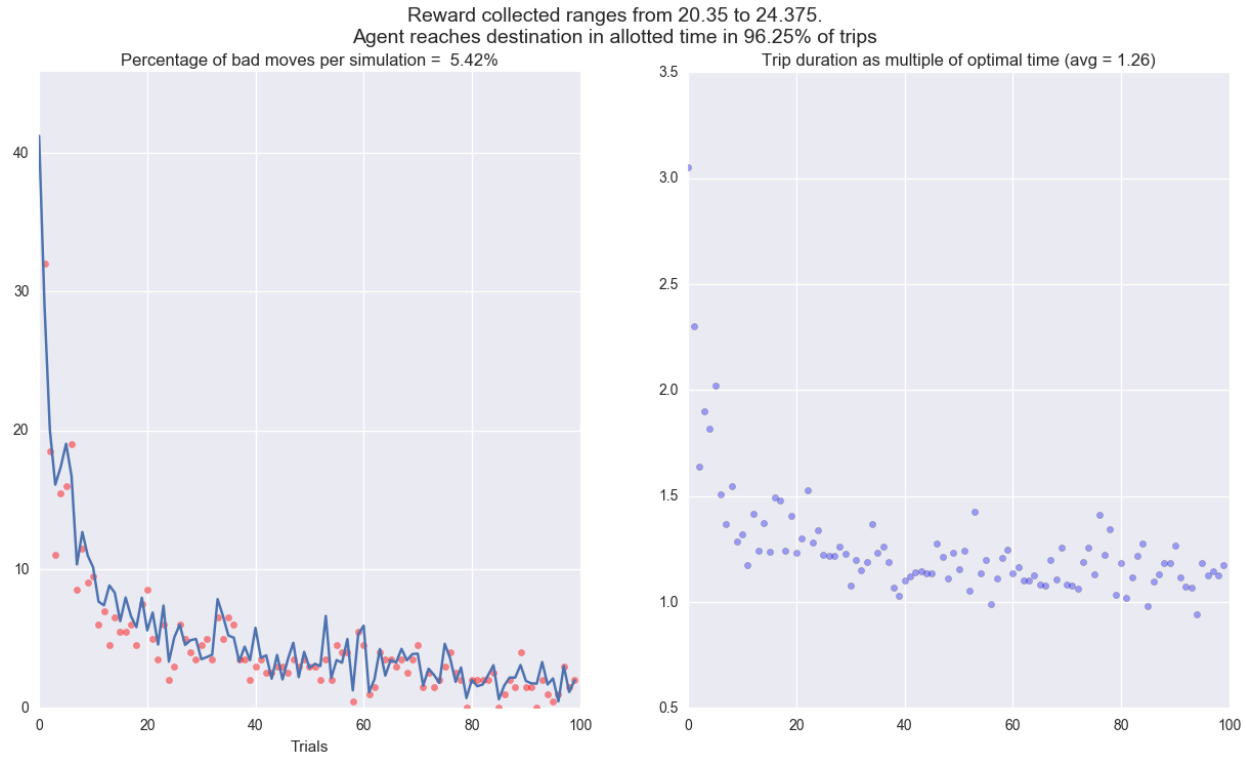
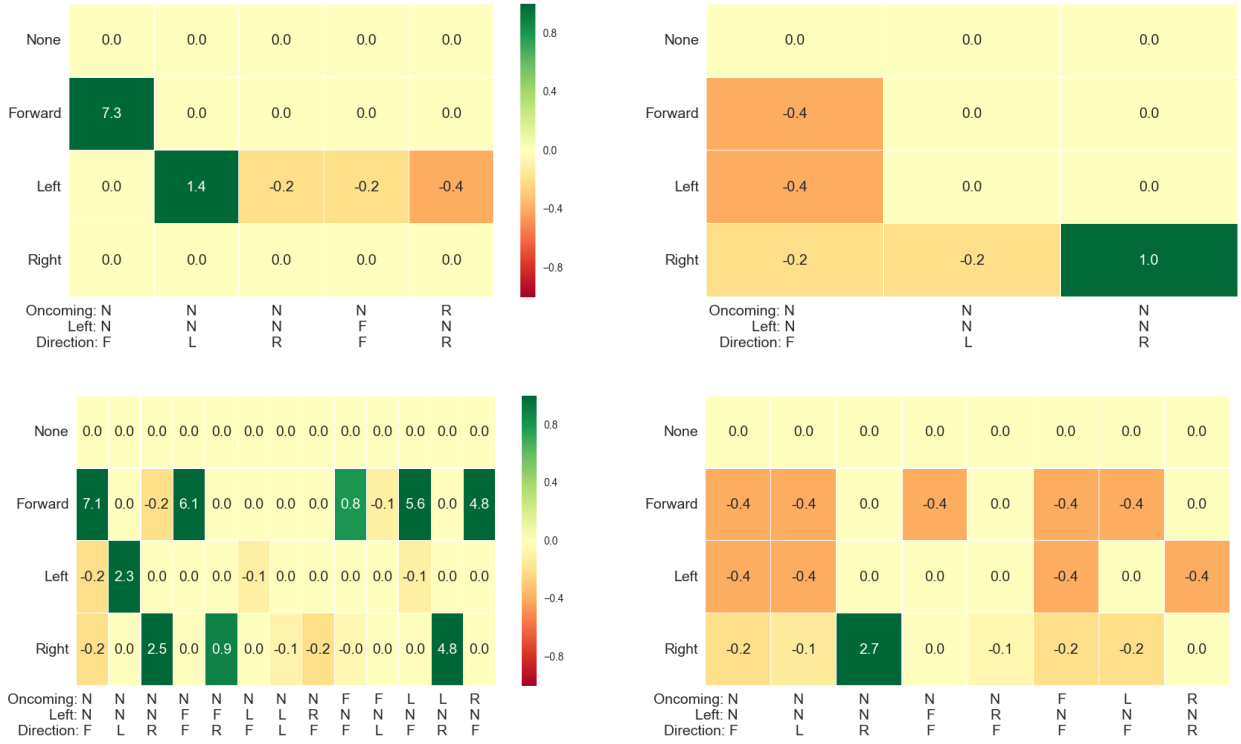


Figure 7. Greedy Action Selection with Q-learning (alpha=0.4, gamma=0.1) in heavy traffic

Greedy Q-learned policy vs Optimal Policy

The following set of figures illustrate a representative Q-matrix derived from one simulation with greedy action selection and optimized learning parameters. The left panels correspond to the traffic signal being green and the right panels correspond to the signal being red. The two rows show Q-matrix rows where the agent was in a low remaining time state (time remaining <6) state (top row) or in a high remaining time state (bottom row). The simulation was done with the default number of dummy. For each subplot, the y-axis lists the actions available to the agent: none, forward, left or right. The x-axis shows the state, which is represented by several pieces of information – signal, traffic and desired direction (way point). We display the traffic condition using a letter code for none (N), forward (F), left (L) or right (R) and the way point. The Q-matrix value is displayed in each cell, for example, in the top left subplot, the dark green cell in the first column of the second row has a Q-value of 7.3 and represents the value for moving forward (action) when the light is green, and the agent has less than 6 time steps remaining; there is no oncoming traffic (N) in any of the other two monitored directions (oncoming or left).



The Q-matrix shows our agent is well trained, especially in the high time remaining case and no other traffic. Note that not all states in our state space are shown as we have excluded any states that the agent never visited. All the greedy moves would match the rules of traffic in the US, both for when the light is green and when it is red with no traffic. The cases where there is traffic shows that the agent hasn't completed learning as the number of these instances were too small.

The Q-matrices for the binary time-remaining feature appear to be similar, suggesting that the behavior of the optimized agent isn't strongly influenced by time-remaining. One interesting observation is that there are fewer visits and a smaller extent of state-explored for the low time-remaining case, presumably because our optimized agent typically completed each trip with time to spare, therefore rarely visiting the low time-remaining region of state space.

The single factor that influenced the results reported here is our choice of what to optimize. As we discussed earlier, there was a distinct choice between maximizing total reward for the trip and either one of maximizing success rate, minimizing demerits or trip length. In this case, maximizing reward effectively minimizes success rate. We argued that maximizing success rate was the optimal choice for this project even though reinforcement learning typically selects for maximum reward.

References

1. *Reinforcement Learning: An Introduction*, Richard S. Sutton and Andrew G. Barto, *MIT Press, Cambridge, MA, 1998.*