# Smart Cab Reinforcement Training Report

## Randomly Drifting

When completely random, the cab eventually reaches its target but it is really clear there is no purpose or logic to its choice in movement. It sometimes moves closer and sometimes further away with no predictability. In some cases, the car managed to find the goal but only after a very long time.

## Choosing States

I chose to include the color of the light, the presence of other traffic in the forward, left and right directions and the next waypoint in my state model. I chose the color of the light since there appears to be a penalty for running red lights that needs to be incorporated to the model. The presence of oncoming vehicles was included to allow the model to interact with potential collisions or adversarial interactions.

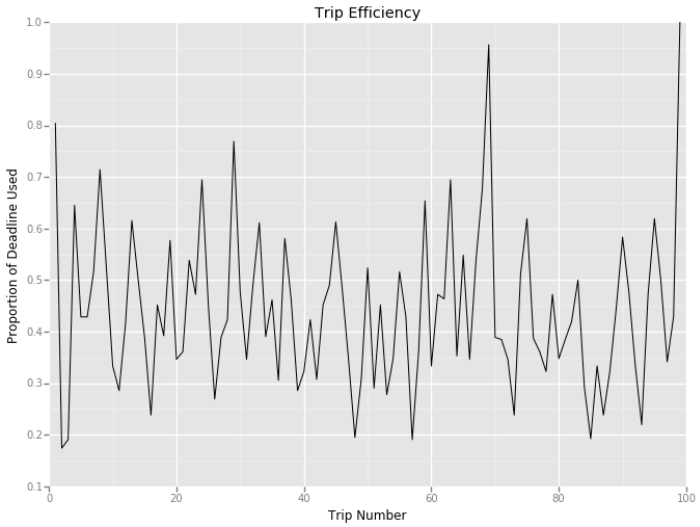
Q-Learning normally has a state for every possible map location in most of the toy problem sets I saw in research. In this case, modelling the entire map would be problematic for a number of reasons (the training time would be almost unworkable waiting for every permutation of the map in combination of all the possible lights, oncoming vehicles from various directions and the target location. Since the target changes with each run, it's not realistic to model every option and the resulting training wouldn't be able to generalize to other maps.

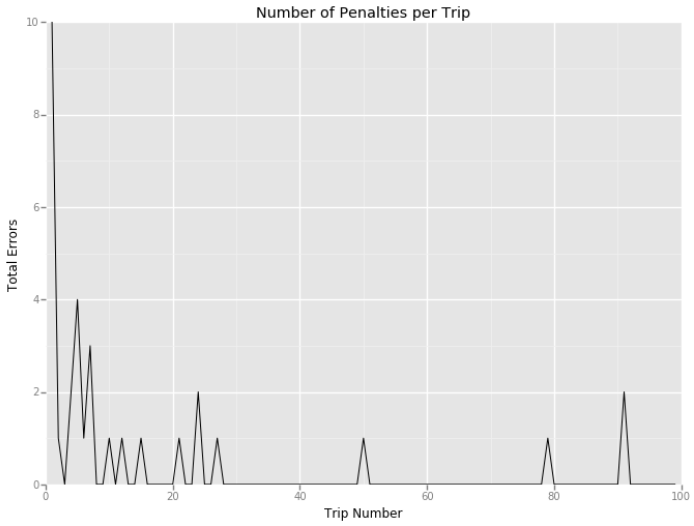
To address these issues, I chose to instead only model the next waypoint to encode spatial information relative to the goal. This greatly reduces model complexity (there are only 4 possible relative directions vs. 8x6 locations) and should allow the model to generalize rules much more quickly.

## Implementing Q-Learning

After adding the Q-Learning algorithm to our Smart Cab, the performance is just as bad at first as it explores its environment. As it begins to gain more and more experience the number of bad decisions falls off fairly dramatically and the car begins to follow a policy that seems to correspond with the actual rules of the road in the simulation.

*The chart on the left below shows the proportion of the deadline that was used in an individual trip (over a sample of 100 consecutive runs.) The car fails to reach the goal in the allotted time for the first few trips but then begins to reliably reach the goal (aside from one failure much further on.) Visualizing the number of penalties per trip (on the right) we can see a similar trend where the car quickly appears to learn to avoid most of the punished actions (although it still occasionally makes errors later).*



## Metric Choice:

Before beginning to optimize the model, I chose a few metrics to set as objective things to work towards improving. The goals of the project are to make sure the car can reach its destination, that it avoids breaking the rules (penalties) and that it reaches the goal in a quick manner by ensuring the policy is optimal.

To measure efficiency of the model, I measured the proportion of taken steps compared to the deadline (since the deadline appears to be correlated to the distance from starting). An efficient policy should ensure this metric remains as low as possible once trained.

To measure rule compliance, I looked at the absolute number of negative rewards in each trip since they represent an invalid move. I chose not to make the metric proportional to the number of moves since we are already measuring efficiency and I want to avoid broken rules absolutely (make sure they can't be diluted with a more circuitous route.)

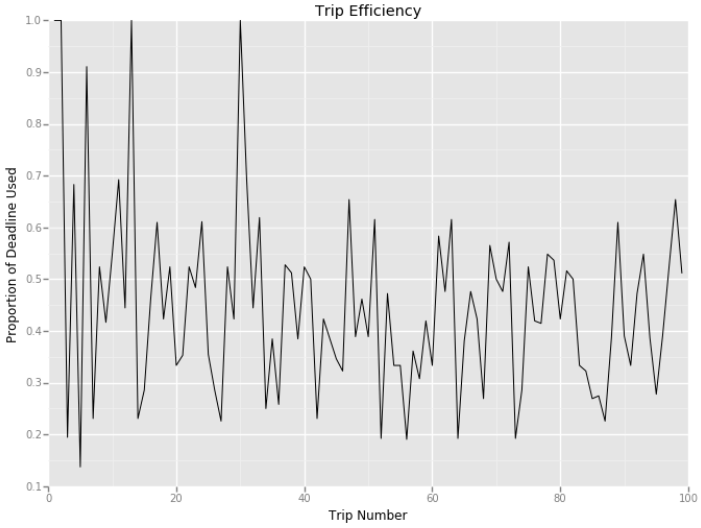
## Optimization:

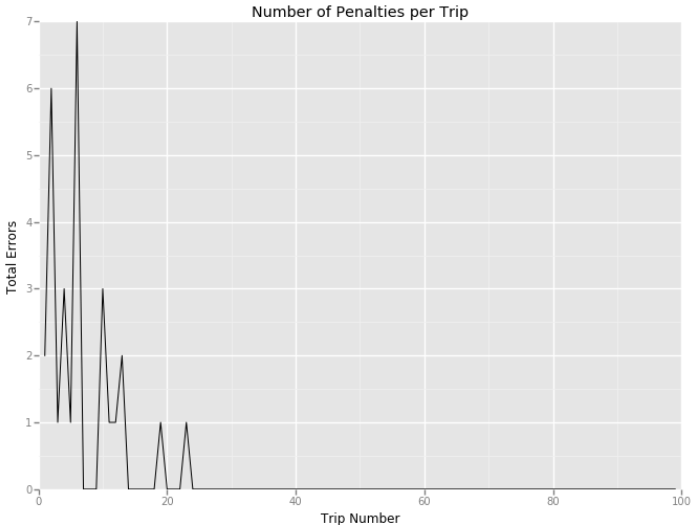
The optimization steps I took were to address the remaining errors seen in later trials and to try and improve the efficiency.

Looking at the result data for the errors in the later runs, it seems the model mostly makes mistakes when exploring (vs. using the model). There are two aspects that can drive this behavior: situations where the model still doesn't know the correct policy action from past learning and cases where the explore vs. exploit choice is set to explore.

The original model used an explore parameter that started at one and decayed linearly to near zero combined with a random number generator to determine if it should explore (reducing the odds of selecting explore as time went on.) Since it never actually reaches zero, the cab will always potentially be able to choose explore. This doesn't seem to have actually occurred in the final 50 trips and so the decay rate seems to be set too low to be effective. The five instances where it did choose to explore were all cases where the model did not have a Q value set as it had not encountered the scenario before. In all the example cases the cab was encountering a situation where the oncoming traffic (which is very sparse) was in a novel configuration.

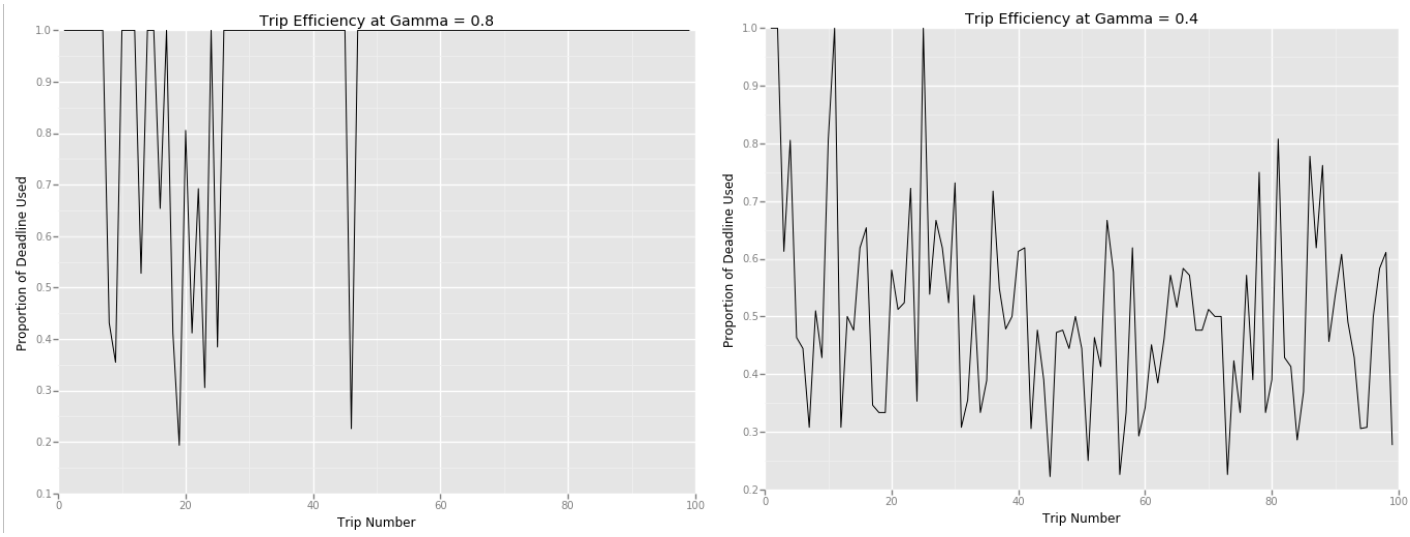
To address this, I updated the state to only include the color of the light, the next waypoint direction and whether there was a vehicle in the direction of the next waypoint. This allows for a much simpler state model with many fewer possibly states and causes the error rate to drop dramatically after the initial training period.




Adjusting the alpha (learning rate) makes a minor effect on how long errors persist with going too low (0.10) seeming to cause performance errors to be evident in later trials (although the effects appear to be minimal.) Adjusting the gamma or discount causes a much more dramatic effect on the efficiency of the model. Below approximately 0.6, the learning

rate and efficiency remain consistent but above 0.7, the cab seems to learn to take steps of 1 without attempting to reach the goals. Since there is a large element of randomness in each training series, to determine the optimal settings more completely, we would need to run a large number of repeated training/testing simulations at each parameter state to determine the statistically optimal rates but the model did show improvements.




Larger scale testing at increments of 5% for both alpha and gamma were somewhat inconclusive, possibly due to not being able to simulate enough trials to generate a true baseline measure at each of the points (to take into account both the randomness of the environment and the training). Additional tuning would need to be modeled over much larger testing environments or have a more scientific approach taken to design.

## Post Optimization:

After optimizing the State model, learning rate and future discount, the model performs more efficiently (mean 41% of deadline taken vs. 44%) and with a lower error rate (mean 0.27 penalties/move vs. 0.32) and with a much lower later error rate (no penalties in the later 50 trials vs 3). The agent appears to have correctly converged on policies very near to ideal.

Additional efforts to encode the overall direction of the goal into the state could potentially improve efficiency (e.g. if the overall target is to the upper right and turning right on red could still bring the car closer even though the next waypoint is straight ahead) but since the goal's location isn't directly available in the sensory data, in the scope of the project it might be outside the rules.

Another point of further efforts would be to revise the explore vs. exploit decision process to take into account the number of explored actions for a given state and the certainty (change compared to the last updates) to better account for the confidence in the Q metric at any given decision point. This would allow the learning to be completed more quickly with less need for exploration outside the required training period.

## An Optimal Policy:

In the simulated world, an ideal policy would be to always proceed towards the goal while obeying all traffic light rules. I'd also say avoiding oncoming traffic when proceeding would be ideal but since there isn't currently a penalty for the near misses, I can't expect the smart car to learn to avoid them. By following the next direction indicator, the route tends to be minimized as efficiently as possible.

Watching the model after training, it seems to obey these tenants by staying stopped at reds, proceeding towards the goal direction and even managed to learn that turning right on red is acceptable but not left which speeds up arrival. The one opportunity the car didn't learn was when the goal was both forward and to the right (but the next direction was forward) that it should turn right at a red to avoid losing a turn waiting. Given this would have involved encoding data

that wasn't available to us directly into the State data and the traffic lights are random, it's not clear this would have been legal or more efficient in larger scale testing but would have been an interesting additional feature.

## References:


Using named tuples for storing Q-Values
**jjinking** in the discussion forum https://discussions.udacity.com/t/guidance-please-on-states/44187/4

An amazing practical step through that really got me through to a working model:
https://www-s.acm.illinois.edu/sigart/docs/QLearning.pdf

Another great explanation piece:
http://artint.info/html/ArtInt_265.html