Smartcab Project Report

A Note on Performance Evaluation

Two criteria for performance evaluation are considered - **Accuracy** and **Fraction of Invalid Moves**.

Accuracy ¶

Accuracy is defined as the fraction of times the learning agent is able to reach the destination within specified deadline. It is calculated both for the entire 100-trials period, as well as for the last 30-trials period. If some agent takes longer time to train than others, the latter could be a fairer measure of its performance after training

Fraction of Invalid Moves

In the context of driving, any traffic violation is highly undesirable. Therefore, for every run of 100-trials, we track the fraction of invalid moves to the total number of moves. For every type of learning agent that we consider, we make 5 runs of 100-trials, take the mean of time series obtained from each run and plot the resulting time-series. In addition to having a high accuracy, a good learning agent must have this measure reach a fairly low value towards the end of 100-trials period.

Random 'Learning' Agent

Random 'Learning' Agent, in fact, does not learn and at each step, chooses a random action from the set of all the possible actions:

```
random.choice(valid_actions)
```

QUESTION: 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?

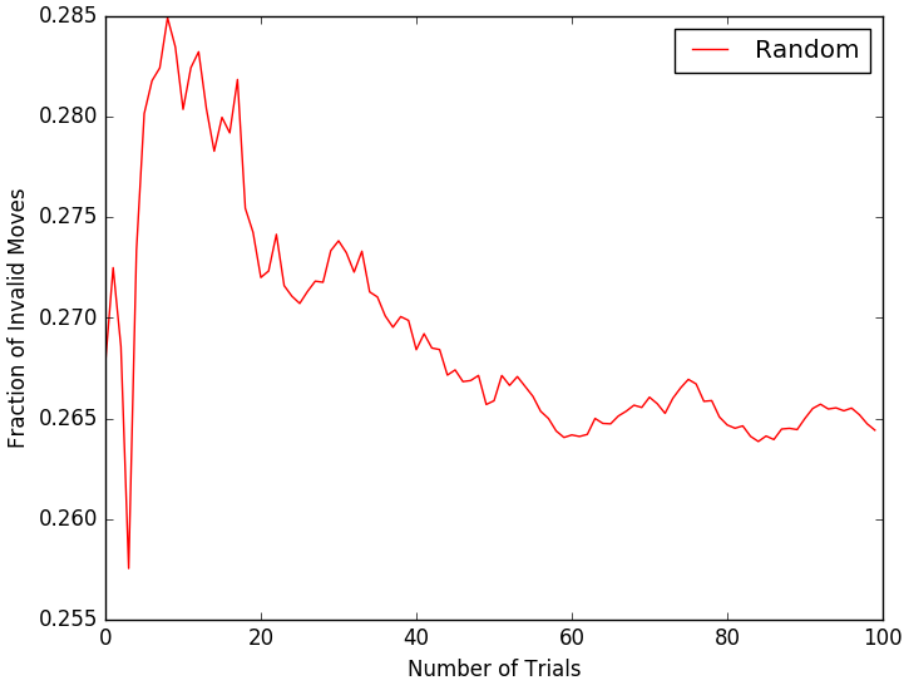
We consider two cases - one where the deadline is enforced and the other where it is not.

`enforce_deadline = True`

When the deadline is enforced, the learning agent reaches the destination in only around 20% of the trials. This figure for accuracy shows no improvement in the last 30 trials confirming that there is no learning going on here.

Accuracy_random_enfDeadline-T	Accuracy_random_enfDeadline-T_last30
0.23	0.2
0.2	0.17
0.21	0.2
0.18	0.17
0.27	0.2

`Fraction of invalid moves` stays above 0.25 throughout the run of 100-trials, dangerously high. We will take this as the base case against which any learning agent must perform better to be really considered a learning agent.

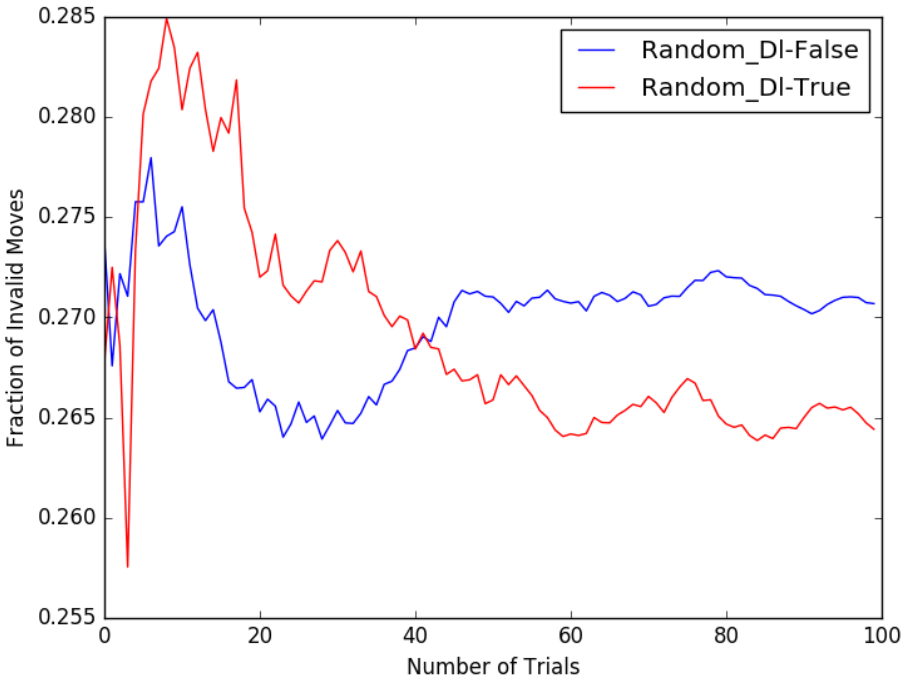


`enforce_deadline = False`

When the deadline is not enforced, the learning agent reaches the destination in around 60% of the trials. This jump in accuracy is because of the fact that the set of possible positions on a grid is finite, and given sufficient time, a random walk starting from any point on the grid can reach any other point on the grid. The reason why accuracy is not 100% then is that even when `enforce_deadline = False`, there is another `hard_time_limit` that is always imposed, which implies that our random learning agent gets additional 100 moves to reach the destination, not infinite time.

Acc_rand_enfDeadline-F	Acc_rand_enfDeadline-F_last30
0.59	0.57
0.63	0.73
0.65	0.7
0.68	0.67
0.69	0.63

Despite reaching destination more frequently, there is no qualitative change in the behavior of the random learning agent when the deadline is not enforced. This is clear from the following plot of `Fraction of Invalid Moves` for both the cases.



Inform the Driving Agent

QUESTION: 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?

The states that I think are appropriate for modeling the smartcab and environment are traffic light, traffic from oncoming, left and right directions, and next waypoint given by the `RoutePlanner`. These are encoded in a tuple stored in the variable `state`:

```
state = (light, oncoming, left, right, next_waypoint)
```

The first four states are related to traffic rules and the information regarding them is important for the smartcab to avoid breaking the traffic rules. The fifth state provides the information about what path to follow to reach the destination, therefore indispensable for the smartcab to reach the destination before the deadline.

Another state that might be of importance is the time before deadline expires. But adding this state would multiply the state space by a factor of around 50, which would neccessitate much more than 100 trials to train the smartcab. Moreover, the utility of this information to smartcab seems marginal at best - one situation where it might be of some use is when the cab is stuck at red light while following next waypoint, it might make a valid move to the left or right if the deadline is running out. But such a move which is in conflict with the next waypoint, even if valid, is penalised by the rewards system. It might also happen that the agent learns to ignore traffic rules to get to destination because the bonus reward of 10 would compensate for the negative reward of -1 for traffic violation. Therefore, I've decided not to include even a binary version of this information (deadline approaching, deadline not approaching), but it could certainly be useful and we might look at it if our current state doesn't result in good model performance.

OPTIONAL: How many states in total exist for the smartcab in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?

Among the states that we have identified, light can take two values, and the rest four can take a maximum of three values each. Therefore, the size of our state-space is bounded by `2*(3^4) = 162`. Now, assuming an average deadline of `30` for each trial, `50` trials would give around `1500` moves in the above state-space of size of around `150`, which seems reasonable for a good learning agent to learn and make informed decisions. This is because transitions in the above state-space would form a finite state irreducible aperiodic chain, for which every state would be visited at least once if the number of transitions is large enough. How large is large enough is a difficult question, but the number of transitions being `10` times the number of states seems reasonable.

Basic Q-learning Agent

Basic Q-learning agent is implemented with parameters:

- learning rate, $\alpha = 0.2$
- discount factor, $\gamma = 0.9$
- exploration rate, $\epsilon = 0.1$

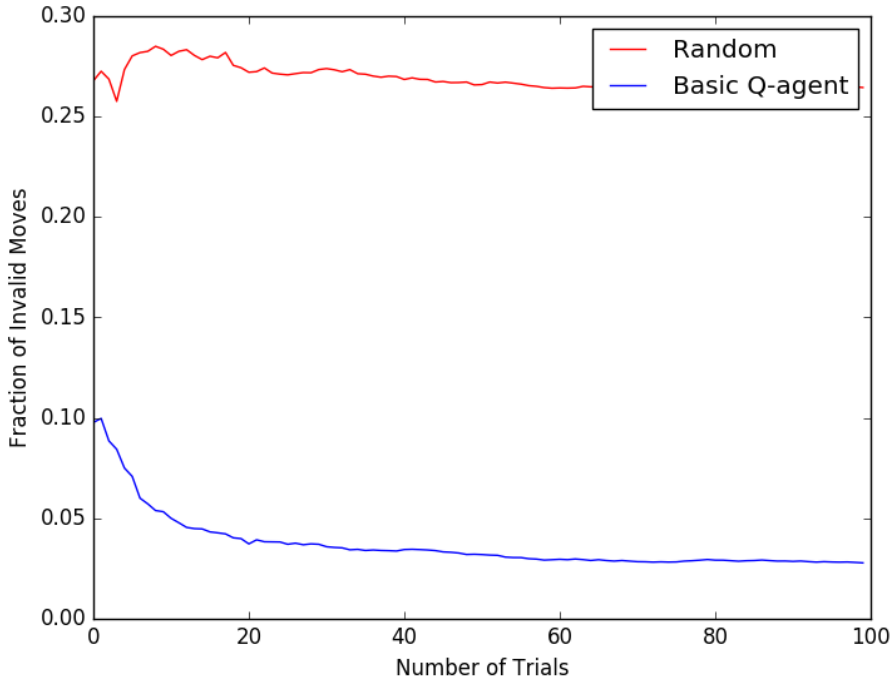
Best action is chosen using `chooseAction(state)` function, while the q-table is updated using `learn(state1, action1, reward, state2)` function. In the basic agent, q-values are initialized to be `0` (in `getQ(state, action)` function), but this is another parameter of the model which could be tuned to improve performance.

QUESTION: 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?

In what follows, we will consider random agent with `enforce_deadline=True` for comparison. Compared to random agent, the basic Q-learning agent performs markedly better, with an accuracy of around `0.85` in most trials. There is no significant difference in the accuracy for all 100 trials and that for the last 30. This indicates that the agent learns enough to reach the destination within the first few trials. This pattern (or lack thereof) will be observed for all the modified learning agents subsequently considered.

Accuracy_qmodel1_eps0.1	Accuracy_qmodel1_eps0.1_last30
0.91	0.83
0.92	0.93
0.89	0.87
0.84	0.87
0.93	0.93

The relatively quick learning is further confirmed in the following plot of `Fraction of Invalid Moves`, which goes down and stabilises at around `0.03` within only a few number of trials.



The basic Q-learning agent performs quite well on both our measure of performance evaluation, but there is definitely room for improvement. Next, we tune some of the model parameters to obtain better performance.

Tuning Model Parameters

QUESTION: 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?

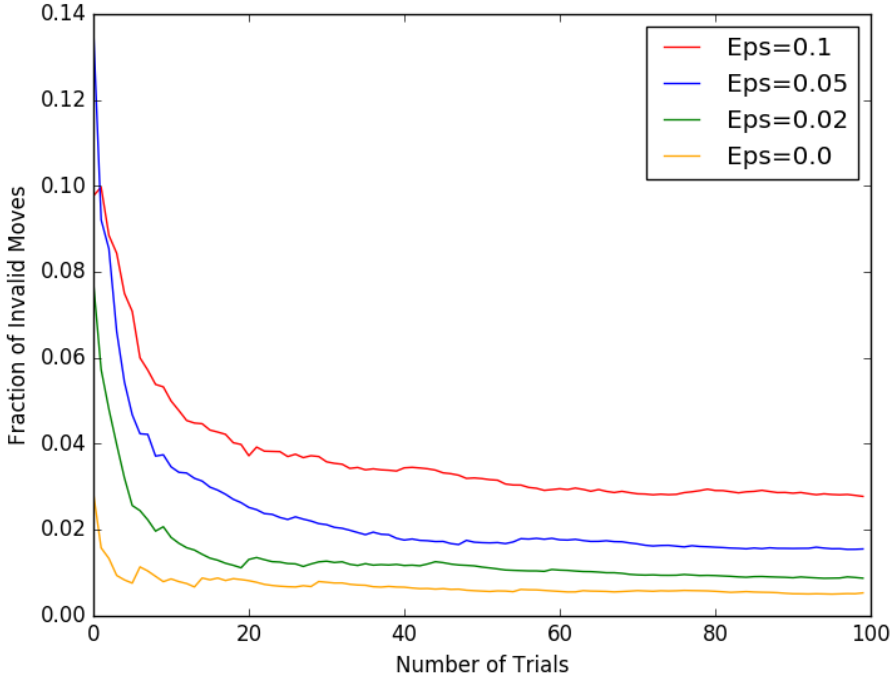
Exploration Rate, ϵ

Exploration Rate of `0.1` means that `10%` of the agent's actions will be chosen randomly even upto the end. If we could lower this rate without significantly impacting the speed of learning, we could definitely gain some performance improvement. We test this hypothesis for $\epsilon = 0.05, 0.02, 0$, respectively. The measurements are documented in the tables and the plot below.

Accuracy_qmodel1_eps0.05	Accuracy_qmodel1_eps0.05_last30
0.89	0.87
0.91	0.87
0.94	0.93
0.92	0.97
0.88	0.83

Accuracy_qmodel1_eps0.02	Accuracy_qmodel1_eps0.02_last30
0.99	0.97
0.97	1
0.97	0.93
0.95	0.87
0.96	0.9

Accuracy_qmodel1_eps0.0	Accuracy_qmodel1_eps0.0_last30
0.98	1
0.99	1
1	1
0.99	1
1	1



Quite remarkably, we don't see any Exploration-Exploitation trade-off here. As we decrease ϵ, both our measures of performance evaluation improve. For $\epsilon = 0$, overall accuracy for 100 trials is consistently above 98%, and accuracy for the last 30 trials is consistently 100%! Even the `Fraction of Invalid Moves` goes down to around `0.005` in this case and pretty quickly as well. This apparent lack of Exploration-Exploitation trade-off may be because of two reasons:

- Random initialization of starting point and destination at the beginning of each trial ensures that every state is visited.
- For every state, the optimal action is probably unique - one that does not violate traffic rules and aligns with next waypoint. Therefore, there is no great risk of being stuck at a local optima.

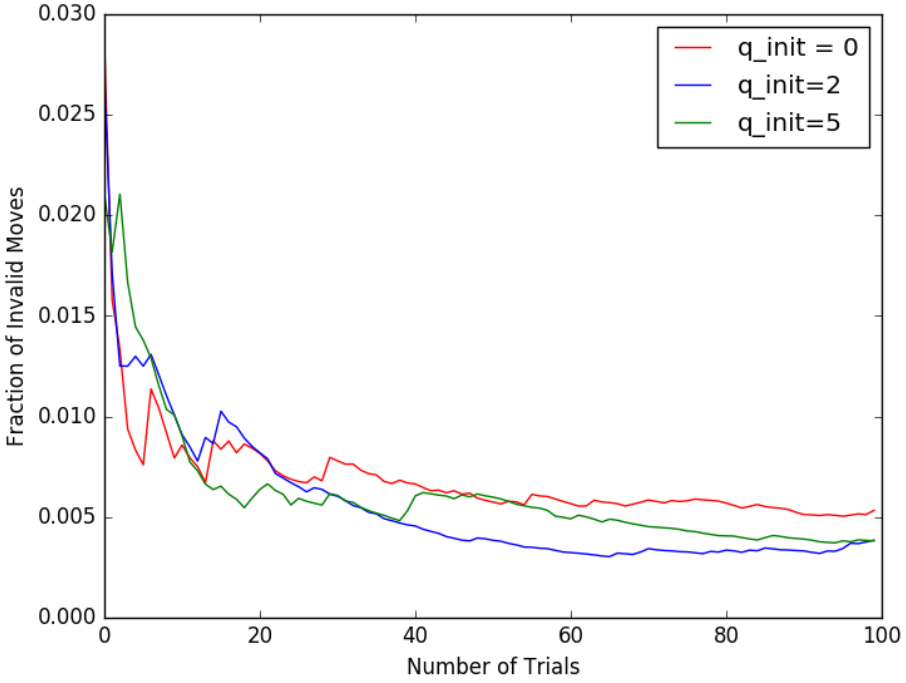
The basic Q-learning agent with exploitation rate, ϵ set to 0 gives pretty much optimal results. Below, we try some more experiments to see if atleast one of our performance measures could be further improved.

Initialization of q

The second reason that we gave above for the apparent lack of Exploration-Exploitation trade-off may not be entirely true. As we discussed earlier, there might be two optima for some states - when traffic light is red, an agent might choose not to follow the next waypoint given by the RoutePlanner, but instead make a valid move in other direction which might help him reach the destination faster. In the analysis so far, we had initialized the q values of all (state, action) pairs to 0, which might cause an agent to be stuck at a globally-suboptimal local optima for some states. In this section, we will keep the exploitation rate, ϵ to be 0, and play around with initialization of q. Specifically, we will consider two cases - one where all q values are initialized to 2 and the second where all are initialized to 5. In the code, this is done by modifying the function: `getQ(state, action)`. Measurements are documented below.

Accuracy_qmodel1_eps0.0_q-initial2	Accuracy_qmodel1_eps0.0_q-initial2_last30
0.97	1
0.99	1
0.98	0.97
1	1
0.97	0.97

Accuracy_qmodel1_eps0.0_q-initial5	Accuracy_qmodel1_eps0.0_q-initial5_last30
0.98	1
0.99	1
0.98	1
0.98	1
0.98	1



As evident from the above figures, differences in performance measures for different initializations of q are insignificant and could very well be statistical artifacts. At best, we can say that there is a marginal trade-off between `accuracy` and `fraction of invalid moves` for q initializations of 0 and 2, but we don't have high confidence in this conclusion.

Decreasing Exploration Rate, ϵ

We try one last experiment - decreasing exploration rate as the trials go on, according to the function:

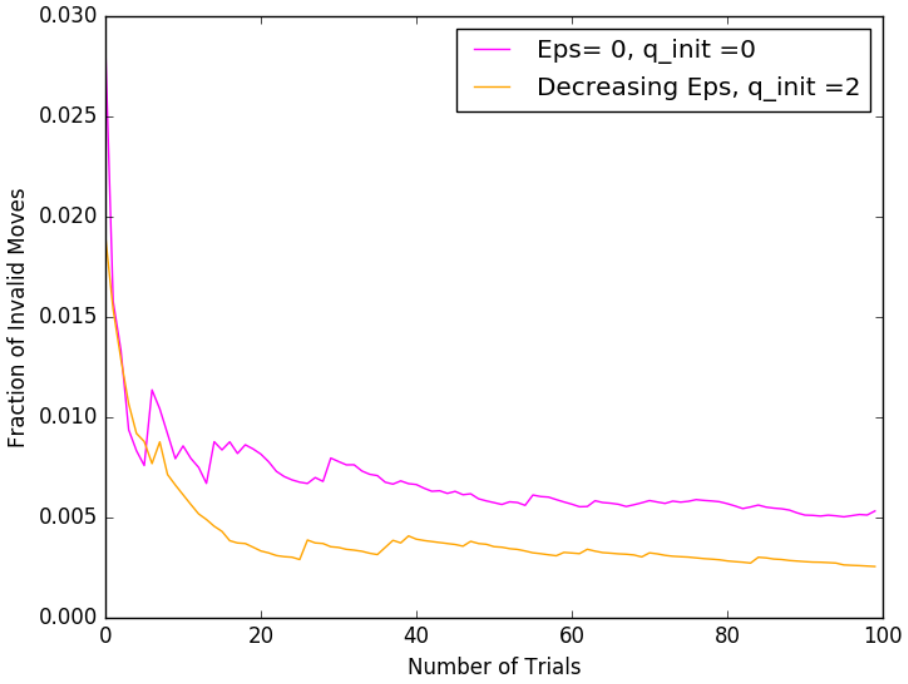
$$\epsilon_{t+1} = \frac{\epsilon_t}{t+1}$$

In the code, this is done by adding the following line in the `run` function of `Simulator` class in the file `simulator.py`:

```
self.env.primary_agent.epsilon /= trial+1
```

For this experiment, we will initialize ϵ to 0.1 and q values to 2. Measurements are documented below.

DecEps_q-init2	DecEps_q-init2_last30
0.98	1
0.96	0.93
0.99	1
0.99	1
0.97	1



Except one run, accuracy for the last 30 trials in this experiment is `100%`, as was the case for fixed $\epsilon = 0$. However,`Fraction of Invalid Moves` seems to be distinctly lower in this case, as low as `0.0025`.

QUESTION: 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?

According to the performance measures we have adopted, an optimal policy would help the agent reach the destination within deadline, especially in the last 30 trials, close to `100%` of times and keep the `fraction of invalid moves` close to `0`. Based on our experiments above, we choose the optimal agent to be with decreasing ϵ and q values initialized to 2. This agent generally reached the destination `100%` of times in the last 30 trials and `fraction of invalid moves` was as low as `0.0025` (it can't be absolutely `0` because of the invalid moves made in the training stage. Although there are other candidates for the optimal agent, we think that this one would be the most robust among those if the 'physics' of the world, that is, the traffic rules and the system of rewards and penalties, is changed because that might lead to change in the dynamics of the Exploration-Exploitation trade-off.