Udacity Machine Learning Nanodegree
Train a smartcab to drive

"In this project you will apply reinforcement learning techniques for a self-driving agent in a simplified world to aid it in effectively reaching its destinations in the allotted time. You will first investigate the environment the agent operates in by constructing a very basic driving implementation. Once your agent is successful at operating within the environment, you will then identify each possible state the agent can be in when considering such things as traffic lights and oncoming traffic at each intersection. With states identified, you will then implement a Q-Learning algorithm for the self-driving agent to guide the agent towards its destination within the allotted time. Finally, you will improve upon the Q-Learning algorithm to find the best configuration of learning and exploration factors to ensure the self-driving agent is reaching its destinations with consistently positive results."

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?*

Random Actions

	Sim 1	*Sim 2*	*Sim 3*	*Sim 4*	*Sim 5*	*Sim 6*	*Sim 7*	*Sim 8*	*Sim 9*	*Sim 10*	*Success Percentage*
Deadline = False	72	71	53	63	64	68	69	68	68	76	67.2
Deadline = True	11	23	11	19	25	16	24	21	16	19	15.5

Table 1 Total number of successes in each simulation of 100 trials.

For ten tests where the agent takes random actions, it reaches the destination on average 67% of the time. In the trials in which it succeeds, it exceeds the deadline the majority of the time. When the deadline is enforced, the average success rate of 10 test runs is approximately 18%.

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 waypoints, as defined in the problem are one intersection away in the direction of the destination. This is important information for the smartcab to take as input. Without considering the other inputs, moving consistently away from the destination is never a good thing for the smartcab to do as it will never reach the destination.

The traffic light information, that is, the state of the traffic light at the intersection helps the smartcab determine whether it can behave legally in the environment. This information is vital to teach the agent how to obey traffic rules.

Finally, the state of other cars at the intersection is information necessary to help the agent avoid accidental collision and behave safely in a real world environment. All three of these inputs, Oncoming, Left and Right are necessary to help the agent learn what states it should avoid getting into.

The deadline, while important is more of a convenience for the passengers than an absolutely necessary input for training the smartcab. It is far more important for the agent to learn how to avoid accidents, obey traffic laws and take the quickest route to the destination than to take the deadline into account when learning.

The final set of states, therefore is the combination of waypoints, traffic lights and other traffic on the road (oncoming, left and right).

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?*

There are 384 states in this environment using the inputs defined above. Q-learning helps the agent learn by adjusting the q-values at each state based on the reward, learning rate, discount rate and future q-values. The larger the state space, the less likely it will be that the agent can visit all of the states during training and adjust the q-values to approximate the rules of driving. Thought it might be possible to reduce this number further, the smartcab's behaviour would suffer as a result.

As this agent is being trained over 100 trials, I believe 384 states to be the minimum amount necessary to achieve optimal results in this environment. In a different environment, however, there could be more possible states, for example, the speed limit or driving in sub-optimal conditions like a snowstorm, that necessarily merit inclusion in the state space of the Q-table. In such a case, much more than 100 trials may be necessary to achieve convergence.

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?*

Q-Learning Results

Trials	*Sim 1*	*Sim 2*	*Sim 3*	*Sim 4*	*Sim 5*	*Sim 6*	*Sim 7*	*Sim 8*	*Sim 9*	*Sim 10*	*Success Percentage*
1 – 100	64	54	66	63	62	63	55	61	55	58	60.1
1 - 50	25	15	23	21	20	20	17	21	17	18	39.4
50 - 100	40	40	44	42	42	43	39	41	38	41	82
90 - 100	10	10	10	10	10	10	10	10	8	10	98

Table 2 Total number of successes in each simulation, by number of trials.

With a basic Q-learning equation implemented, rather than always choosing random actions, the agent chooses actions based on the value of epsilon and the Q-values in the Q-table. In this initial case, the exploration rate is decayed linearly while a constant learning rate and a constant discount factor are used. The deadline is also set to True for all simulations.

Compared with the case where every action was randomly chosen and the deadline was also set to True, the average success rate increased from 15.5% to 61.9%. This alone, however is not a good metric for successful learning as the initial simulations can be viewed as training for the smartcab. The average success rate in the final 10 simulations of the Q-learning model is 98%. Without any additional

parameter tuning, the smartcab is already getting to the destination 98% of the time. This massive improvement in performance over the random action model is the expected result.

Throughout the process of Q-learning, the Q-table is updated to represent the best action the smartcab can take in a specific state. In the very early trials, Epsilon, the exploration rate, is approximately 1. This prioritizes exploration of possible actions as there are no relevant Q-values in the Q-table yet. At this stage, Epsilon is then decayed linearly so that after 50 trials, the agent will begin to prefer exploitation of the q-values it has learned. The results of the simulations above confirm that the agent has done exactly this. In the first 50 trials, the smartcab reached the destination on average 39% of the time, whereas in the final 50 trials, the smartcab reached the destination 82% of the time on average. Somewhat surprisingly, the smartcab has a remarkable success rate of 98% with equal values of alpha and gamma.

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?*

The parameters of the Q-learning algorithm are the learning rate and the discount factor. Another parameter that could merit tuning is the tradeoff between exploration and exploitation, represented by epsilon.

The learning rate determines to what extent new information overrides old information. A factor of zero means the agent learns nothing while a factor of 1 makes the agent consider only the most recent information. The discount factor determines the importance of future rewards. With a factor of 0, the agent will only consider current rewards. With a factor of 1, the agent will strive to attain a long term reward.

The following table shows the average success rates out of ten simulations for each combination of parameters in the final ten trials.

Alpha/Gamma Tuning

α/γ	***0***	***0.25***	***0.5***	***0.75***	***1***
0	0.22	0.21	0.23	0.25	0.21
0.25	1	0.97	0.92	0.93	0.36
0.5	1	1	0.97	0.93	0.18
0.75	1	0.97	0.96	0.96	0.11
1	1	0.99	0.99	0.94	0.25

Table 3 Left: α, Top: γ

When the discount factor is 0, the agent appears to perform the best, with the cab reaching its destination every time in the final ten trials for every non-zero value of alpha. It also performs best for the combination alpha = 0.5, gamma = 0.25. As a gamma of zero makes the agent only consider the current rewards, however, a non-zero value of gamma is preferred. A more exhaustive search of parameter values can be found in the log files accompanying this report.

The final parameters are alpha = 0.5 and gamma = 0.25. The results of 10 simulations are shown in the table below.

Alpha/Gamma Tuning Results

Simulation	*Total Wins*	*Final 10 Wins*	*Last failed trip*
1	65	10	85
2	66	10	63
3	54	10	77
4	65	10	66
5	64	10	61
6	64	9	91
7	73	10	80
8	66	10	89
9	69	10	70
10	74	10	77

Table 4 $\alpha = 0.5$, $\gamma = 0.25$

The average success of this agent in the final ten trials is 99%, that is, the agent makes it to the destination on time 99% of the time. The average success of the agent over all the trials, including the final ten, is 66%. This driving agent appears to converge between the 60^{th} and 80^{th} trial most of the time. This could be improved by changing epsilon's rate of decay. As it currently uses a linear rate of decay, the agent favours exploration with decreasing probability for the first half of the simulation, and exploitation for the second half with increasing probability. When epsilon is set to decay exponentially, the total success rate for this agent is reported below.

Epsilon Tuning Results

Trial	*Total Successful Trips*
1	99
2	98
3	97
4	99
5	99
6	97
7	98
8	98
9	97
10	98

Table 5 Total Successful Trips achieved with exponential decay of ε.

The parameters for the final driving agent are:

$\alpha = 0.5$
$\gamma = 0.25$
$\varepsilon = Ce^{(-kt)}$, $k > 0$

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?*

Though the driving agent does not perform perfectly, it comes close to finding an optimal policy within the 100 trial limit. It appears to learn the rules of the road defined in this problem, i.e. don't run red lights and avoid oncoming traffic. Observation of the rewards obtained by actions taken in the final 10 trials, however, reveals that the agent occasionally ignores red lights. This could be due in part to a relatively low discount factor, leaving the agent open to prioritizing a large future reward over immediate rewards. This could be a problem in a real world scenario and more parameter tuning would be required.

The optimal policy, if safety is the greatest priority, would value traffic signals more than the deadline and always yield to oncoming traffic. Minimizing the opportunities for road accidents and potential death should always come before meeting a deadline. More specifically, the policy would exactly replicate the rules of driving.

The final driving agent gets to its destination at least 97% of the time, but it does not always obey the red light. Of particular note is this learning update obtained in trial 99 of one simulation:

LearningAgent.update(): deadline = 19, inputs = {'light': 'red', 'oncoming': None, 'right': 'right', 'left': None}, action = forward, reward = -1.0

The agent ignored both the red light and another driver on the road. For this action, it achieved only a small negative reward, but could have endangered lives in a real situation. Though the net reward for that trial was positive, the agent did not learn in the previous 98 trials the rule for obeying traffic lights.

In the real world there are even more factors to consider, such as the existence of pedestrians and how to yield to them, the legal speed limit, and the navigation of one way streets. Many more simulations would be necessary before the smartcab could learn an optimal policy.