Project 4 : Training a SmartCab

Yoonyoung Cho

May 21 2016

1 Introduction

In this project, I was tasked to develop a Q-learning implementation of an agent that drives to a destination within deadline, without violating traffic laws. The agent was assumed to lack the understanding of its current position and the destination with respect to the global coordinates, and the sensor only provided insight as to the immediate situation of the neighboring traffic and the traffic light. This highly reduced, abstract model of driving, albeit idealistic and naive, provides a testing ground for experimenting with the Q-learning algorithm.

2 Basic Driving Agent

I implemented the basic driving agent by creating a generic learning agent with exploration rate (ϵ) parameters. By setting this to a constant 1.0, the agent responded in a completely arbitrary manner to the state input. This agent, naturally, provides insight as to the baseline *worst* performance of the agent: that is, a learned agent ought to perform better.

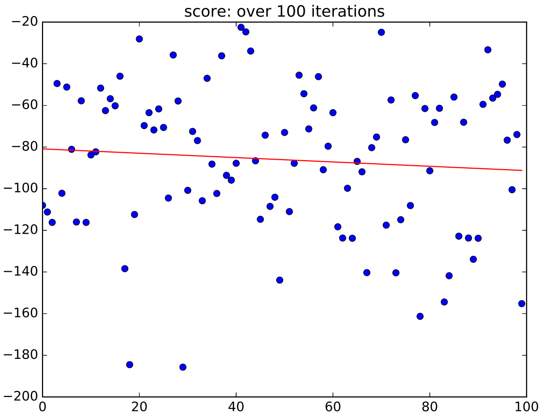
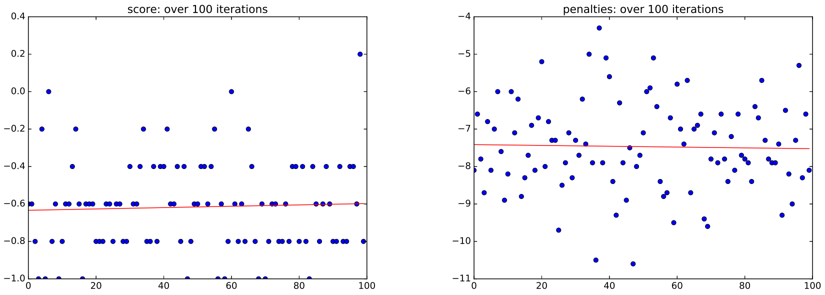


Figure 1: Reward over 100 trials for a random agent, with unlimited time for each trial.

With unlimited time, the agent, on average, ended up with the net reward of -86.06 by the time it got to the destination, which not only indicates that it violated the traffic laws quite often, but also that it reached the destination very lately. Considering that the violation of the traffic law only entails a penalty of -1.0, the net reward of -86.06 is equivalent to at least as many as 86 violations, potentially with a lot more – considering that positive reward is granted for staying put (1.0), driving correctly (2.0), driving legally (0.5) and reaching the destination within deadline (10.0).



(a) training score

(b) incurred penalty

Figure 2: The performance of the agent over time, with enforced deadlines, smoothed over 10 repeated trials: it is apparent that the agent does not progress.

In order to compare with other agents, appearing later in the report, I also tested the agent with enforced deadlines. Naturally the agent didn't perform well – in that it made to the destination for less than a third of the time. In the figure, and the metrics for the following figures as well, 1 indicates reaching the destination and -1 indicates the failure. The average score of -0.6, in this case, indicates that the agent reached the destination in about 30% of the runs; the average penalty is around -7.5 per each trial, which corresponds to approximately 8 illegal actions per each drive.

3 State Definition

My state definition was composed of the *traffic light*, and the status of cars *oncoming, left, and right* to my intersection, as well as my *next waypoint.* These are the minimum, unprocessed features required to form a reasonable response as to the subsequent action: the current state of the road and which way to go next, both of which are directly related to whether the action would incur a penalty or a reward. I didn't include the deadlines as an attribute to the state, because – while it may influence the decision of the agent – it would significantly increase the size of the state space with trivial improvements.

4 Q-Learning Implementation

I implemented Q-Learning with a basic ϵ-greedy learning agent, with the following *arbitrary* parameters:

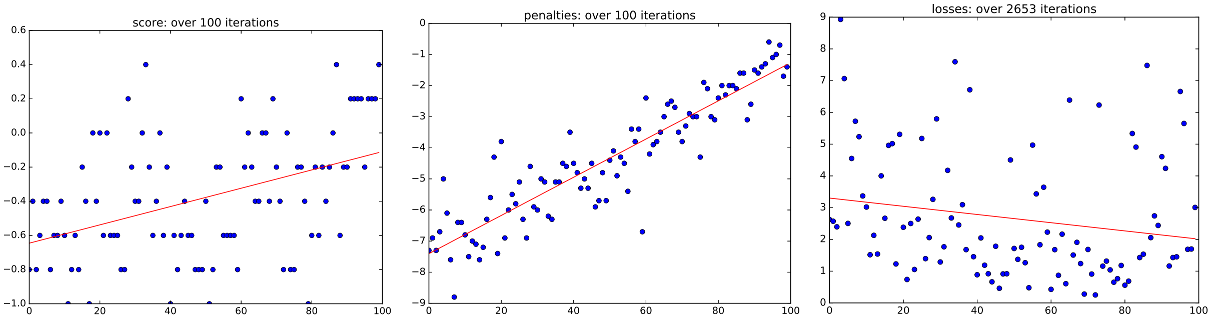
Field	Value
ϵ_{start}	1.0
ϵ_{end}	0.05
ϵ_{anneal}	linear
α_{start}	0.8
α_{end}	0.1
α_{anneal}	linear
γ	0.8

Table 1: Q-Learning Parameters; ϵ corresponds to the exploration probability in ϵ-greedy learning; α is the learning rate, and γ is the discount factor for future rewards.

Here, the α and ϵ were linearly annealed to balance exploration and exploitation: initially, the agent learns quickly and explores a large breadth of states. As the agents accumulates experience, the agent gradually tends to exploit its knowledge in selecting its actions. This enhances convergence and decreases the instability.

Moreover, I initialized the Q-table with 10.0 as the starting value; because I initialized the table with a value that is larger, in most cases, than the learned value, the unseen state-action would be more likely to be selected by the agent than the already-seen pairs. This *optimistic initialization* effectively encourages exploration.

As expected, though, the agent under these parameters performed quite poorly.



(a) training score

(b) incurred penalty

(c) Q-learning loss

Figure 3: The score of the agent over time, smoothed over 10 repeated trials. The agent improves, but not to a great extent.)

As seen above, the agent is unable to reliably reach the destination, even after training over 100 trials; it only improves up to the point in which it reaches destination about half of the time. Qualitatively, the agent is prone to engage in cyclic behavior, rather than waiting until the traffic lights change; this, among other factors, introduces inefficiency in its travel, which – in turn – reduces its overall capacity to reach the final destination.

The almost linear reduction in penalty is noteworthy. This trend is probably, at least partly, due to the linear ϵ-annealing, which decreases the exploration probability in direct proportion to how many trials the agent has undergone.

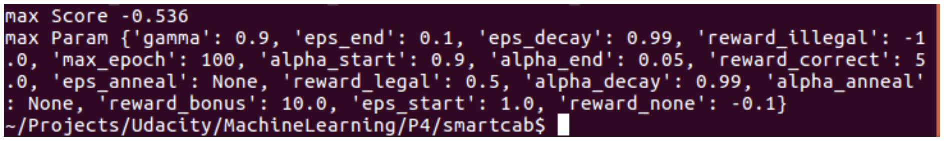
The agent definitely comes to learn about the environment, as the decrease in loss indicates, but never quite learns the best policy. For reference, here, and onwards, the loss function is the standard loss function $(0.5 * RMS(\Delta\vec{Y}))$.

5 Enhancing the Driving Agent

5.1 Grid Search

In order to enhance the performance of the agent, I performed a series of brute-force grid-searches to find the best hyperparameters.

In the initial attempt, I thought of multiple arbitrary values for each parameters, which ended up with more than 10,000 net combinations. This search didn't prove to be very useful due to the heavy computational cost, which greatly limited the number of repetitions to perform (for smoothing the data) – and the fact that I had to determine some number of 'reasonable' values for each parameter.



```
max Score -0.536
max Param {'gamma': 0.9, 'eps_end': 0.1, 'eps_decay': 0.99, 'reward_illegal': -1
.0, 'max_epoch': 100, 'alpha_start': 0.9, 'alpha_end': 0.05, 'reward_correct': 5
.0, 'eps_anneal': None, 'reward_legal': 0.5, 'alpha_decay': 0.99, 'alpha_anneal'
: None, 'reward_bonus': 10.0, 'eps_start': 1.0, 'reward_none': -0.1}
~/Projects/Udacity/MachineLearning/P4/smartcab$
```

Figure 4: Results from the initial grid search.

In the search, the best parameter I found yielded only an average of -.536 in the run, which indicates reaching the destination only a quarter of the time.

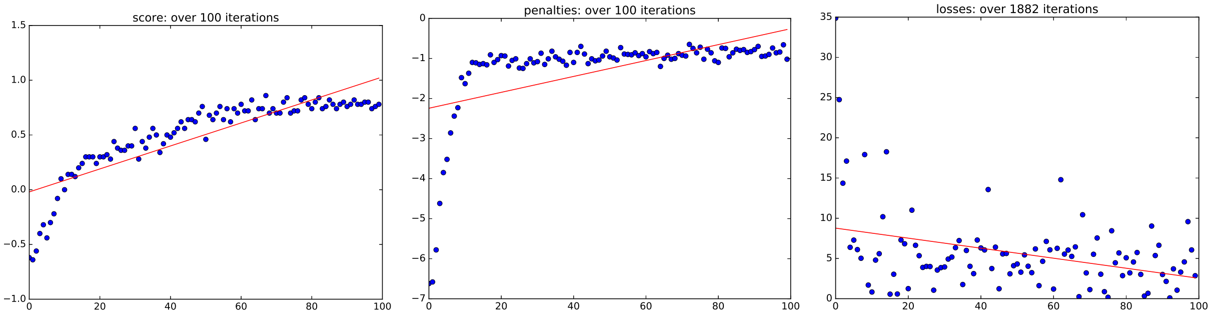
Instead, I decided to search for a smaller pool of combinations that would allow better insight into how the parameters influence the agent's learning.First, I searched for the best learning rate constant, ranging from 0.1 to 0.9, with intervals of 0.1. This, however, didn't yield a good enough agent, possibly due to faulty assumptions about the value of γ.

Thence, I turned to find the best α - γ - ϵ combinations, the representative values for each ranging from 0.1 to 0.9 with intervals of 0.2 and with two annealing options (linear / constant). After that, I introduced slight variations in the found parameter to see how well it performed: the following table describes the thus determined parameters.

Field	Value
ϵ_{start}	0.3
ϵ_{end}	0.05
ϵ_{anneal}	linear
α_{start}	0.3
α_{end}	0.05
α_{anneal}	linear
γ	0.1

Table 2: Q-Learning Parameters; ϵ corresponds to the exploration probability in ϵ-greedy learning; α is the learning rate, and γ is the discount factor for future rewards.

5.2 Performance



(a) training score

(b) incurred penalty

(c) Q-learning loss

Figure 5: The score of the agent over time, smoothed over 100 repeated trials. The agent reaches optimal policy.

The agent trained with the final parameters yielded an average score of .4996, which indicates that it reaches the destination 75% of the time over its training duration. At the end, it reaches .86, which corresponds to reaching the destination 93% of the time. This result was observed consistently, which allows for the conclusion that these parameters lead to convergence to the optimal policy, or at least the best within the constraints of assumptions: even with slight variations of the parameters, the end results were equivalent.

The agent is not completely penalty-free, in that it accumulates an average penalty of -1.26 over each trial, reduced to -.65 by the end (which is less than one violation of traffic laws per drive). This, however, is with the minimum exploration probability of 5%, which may account for the error. Even then, this is a significant improvement to the random agent, which collects an average penalty of -7.5.

6 Conclusion

6.1 Parameter Analysis

In order to get better insight into the system, it may be of benefit to analyze the hyperparameters.

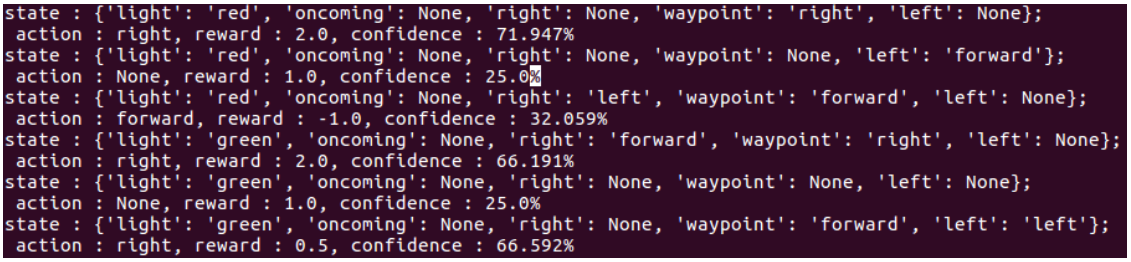
The low gamma values indicate that the temporal discount factor must be high: in order words, rewards in the future are not as meritricious as the immediate ones. I believe that this is due to the fact that there is an enforced time limit in the simulation environment, which rewards reaching the destination before the deadline occurs. Moreover, in the system, the lack of knowledge as to the deadline, or the ultimate waypoint, largely reduces the problem from the global context to a very localized, greedy problem, a trait that necessarily drives γ to be quite small.

The low starting epsilon value is probably due to the fact that the agent must exploit quickly in order to perform well on average in the 100 trials – and that it is possible the learn quickly, given that the system is quite simple. Moreover, due to the *optimistic initialization* of the Q-table, the extra exploration – provided by the high initial ϵ value – need not be as aggressive.

The alpha values could range quite a bit without impacting the final result. I believe that this is due to the fact that the agent learns quickly even for low alpha values, and the environment is relatively stable for high alpha values such that not too many oscillations would occur before converging.

Finally, the annealed result performed better than constant alpha-epsilon values, since it takes the exploration-exploitation dichotomy into account in the simulation.

6.2 Policy



```
state : {'light': 'red', 'oncoming': None, 'right': None, 'waypoint': 'right', 'left': None};
 action : right, reward : 2.0, confidence : 71.947%
state : {'light': 'red', 'oncoming': None, 'right': None, 'waypoint': None, 'left': 'forward'};
 action : None, reward : 1.0, confidence : 25.0%
state : {'light': 'red', 'oncoming': None, 'right': 'left', 'waypoint': 'forward', 'left': None};
 action : forward, reward : -1.0, confidence : 32.059%
state : {'light': 'green', 'oncoming': None, 'right': 'forward', 'waypoint': 'right', 'left': None};
 action : right, reward : 2.0, confidence : 66.191%
state : {'light': 'green', 'oncoming': None, 'right': None, 'waypoint': None, 'left': None};
 action : None, reward : 1.0, confidence : 25.0%
state : {'light': 'green', 'oncoming': None, 'right': None, 'waypoint': 'forward', 'left': 'left'};
 action : right, reward : 0.5, confidence : 66.592%
```

Figure 6: An excerpt of the policy learned by the agent.

The above figure is an excerpt of the policy learned by the agent after 100 trials. As a result of the short run, the agent didn't observe the entire state-space; rather, it has only experienced 47, perhaps some of the most common, states. In the figure, the confidence was measured by evaluating the utility of the four actions with the softmax function.

An optimal policy in this scenario, naturally, would be one that heads to the waypoint if possible – given the traffic situation – and simply waits otherwise, or even take a detour. This is the policy that a human driver would follow when navigating to the destination.

As seen, the resultant policy yields one that is somewhat close to the optimal policy, especially for actions with high confidence. For instance, the agent elects to head right at 71.9% confidence when the waypoint is *right* and, evaluating the traffic states, it is able to go right without penalty.

The policy is, as seen, faulty: the agent attempts to go forward despite the red light – though with a lower confidence. However, it is also important to bear in mind that the agent may not have learned the *fastest*, or the most accurate, policy due to the fact it is able to accumulate more reward, in the trial, through stalling. As seen also in the last policy in Fig. 6, the agent does not go forward despite the fact that the waypoint was forward and the action was feasible.

When, during the simulation, I observed the behavior of the agent (the last 10 trials), in most scenarios the agent heads directly towards the destination, and was able to reach the final destination. However, in one case, I observed cyclic behavior. This may either be due to the random selection of actions that, by chance, directed the agent away from the waypoint, or the effect of the phenomenon observed above, which is that even in the policy the agent would tend to stall, possibly because the penalty for performing a legal, yet incorrect, move is not significantly worse than simply staying put – or possibly just to collect more reward, taking advantage of the fact that, often, more time than necessary is provided to reach the destination.

6.3 Wrapping Up

In this project, I demonstrated how Q-Learning could be used to iteratively update its evaluation of the state-action pair to estimate its utility – the agent found a policy that nearly guaranteed reaching the destination within the deadline.

Considering that the maximum size of the state-space was 512 at most, and often only about a tenth of the possible state space was visited in the 100 trials, the simulated environment is very concise and idealistic. In most practical situations, using a q-table proves to be too naive to store the state space, especially with the curse of dimensionality.

However, this project served as an apt introduction as to how Q-Learning operates, and provided insight as to how the hyperparameters influence the learning process and are influenced by the character of the system.