Training a Smart Cab

You can find all the relevant code for running this PyGame application and the smartcab navigating through it using Q-learning algorithm here.

Installation

Python 2 Mac users

1. Install dependencies `brew install sdl sdl_ttf sdl_image sdl_mixer portmidi`
2. Install PyGame through Conda `conda install -c https://conda.anaconda.org/quasiben pygame`

Python 3 Mac userss

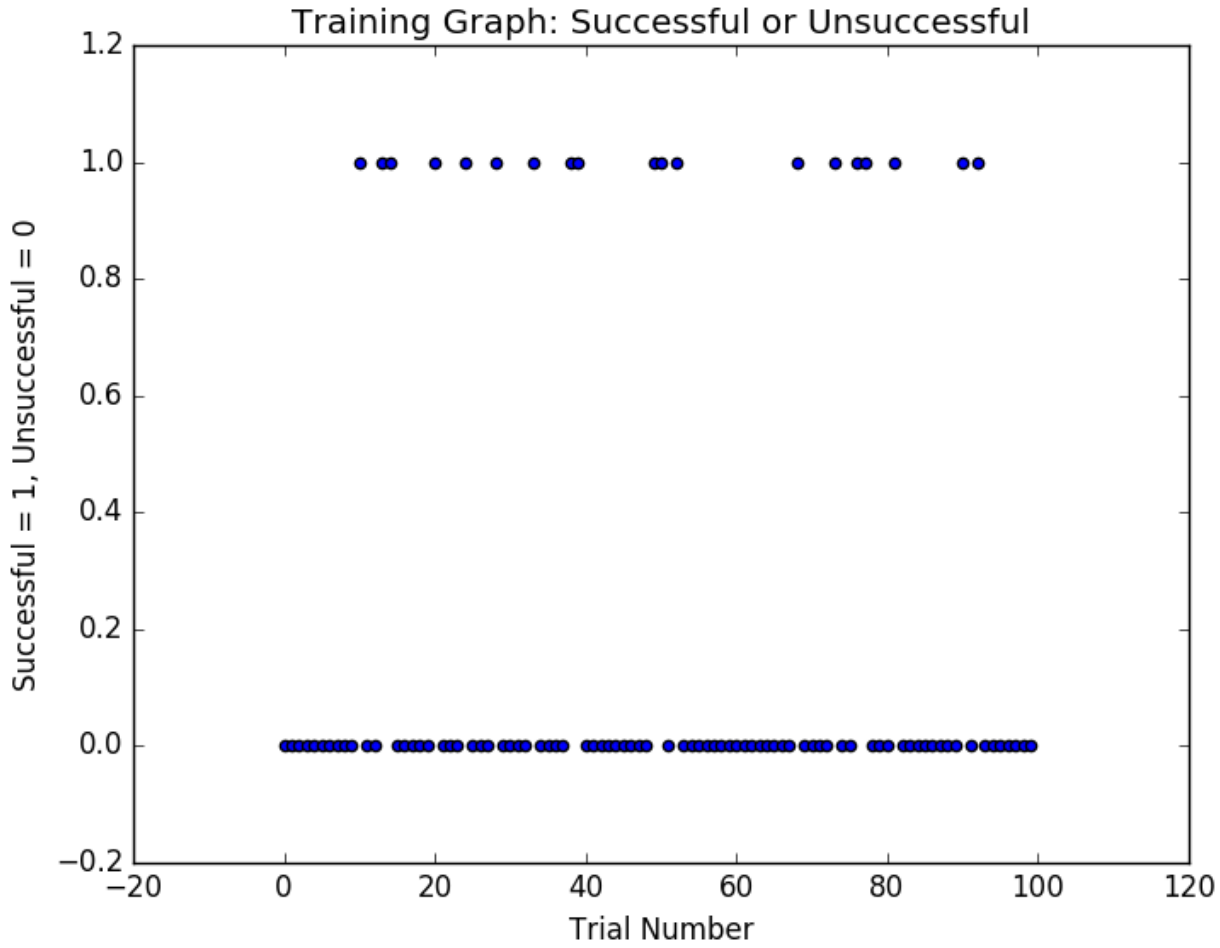
1. Create environment for Python 2.7 `conda create -n py27 python=2.x ipykernel`
2. Activate source `source activate py27`
3. Install dependencies `brew install sdl sdl_ttf sdl_image sdl_mixer portmidi`
4. Install PyGame through Conda `conda install -c https://conda.anaconda.org/quasiben pygame`

Implement a Basic Driving Agent

Code

There is only a change of code for `action` to choose an action randomly through the list `valid_actions = [None, 'forward', 'left', 'right']`.
https://gist.github.com/ritchieng/3a0a813e507f0e0ac68ebc9644fd71ac

Graph of successes and failures



Observations

- If you look at the graph, the car only made it to the end 19 times out of 100 times (19% success rate) based on the following conditions:
  - Successful: within deadline and rewards above 10. And unsuccessful would be the "else" of successful as seen in the code.
- The car does eventually reach there with a few caveats:
  1. It took a long time.
  2. It clashed with other cars.
  3. It made illegal moves (not obeying the traffic lights).
- Once in awhile, the car does reach the destination before the time is up. It reached 19 times to be precise.
- We will be comparing this metric to our improved smart cab when we learn using the q-learning algorithm.

Inform the Driving Agent

Appropriate States

- I have created tuples for the states as they are hashable.
- State

    - State means the environment that the smart cab encounters at every intersection.
    - This could be the:
        - Status of traffic lights (green or red)
        - Status of traffic at that intersection
        - Deadline
        - Next waypoint.
    - We have to decide what inputs you old like to include to define the state at the intersections.
- Relevant inputs:
    - Status of traffic lights: red and green.
        - This input is relevant as we need to obey the rules to reach our destination.
    - Next waypoint
        - This is relevant too because it concerns where the car should go.
    - Status of traffic at that intersection
        - This is a situation where there are few cars. This may not be so relevant but it helps in reducing penalties and increasing our cumulative success rate.
- Irrelevant inputs:
    - Deadline
        - I would normally include this.
        - However, there seem to be no rewards for reaching early. Only rewards for obeying traffic rules.
        - Hence, it is not worthwhile to include Deadline as an input as this would drastically increase the number of states we need to train on.

Number of States

- As such, we would have states that factor in next_waypoint (Left, Right and Forward), light (red or green), incoming traffic (None, 'forward', 'left', 'right')
    - This would result in 3 (next_waypoint) x 2 (light) x 4 (incoming traffic) = 24 states
- 24 states per position seem like a reasonable number given that the goal of Q-Learning is to learn and make informed decisions about each state.
    - This is because we have to understand that we face an exploration-exploitation dilemma here that is a fundamental trade-off in reinforcement learning.
    - This seems like a good balance of exploration and exploiting 6 states.

Code

https://gist.github.com/ritchieng/905f12bf65265331f0e051541379c767

Implement a Q-Learning Driving Agent

Estimating Q from Transitions: Q-learning Equation

- Personal notes:

Estimating Q from Transitions: Q-learning Equation

- $$\hat{Q}(s,a) \leftarrow_{\alpha_t} r + \gamma \max_{a'} \hat{Q}(s',a')$$

- **Intuition**: imagine if we've an estimate of the Q function.
  - We will update by taking the state and action and moving a bit.
  - We have the reward and discount the maximum of the utility of the next state.
- This represents the **utility function**.
  - $$r + \gamma \max_{a'} \hat{Q}(s',a')$$
- This represents the **utility of the next state**.
  - $$\max_{a'} \hat{Q}(s',a')$$
- α_t is the **learning rate**.
  - $V \leftarrow_{\alpha_t} X$
  - $V \leftarrow (1-\alpha_t)V + \alpha X$
    - When $\alpha = 0$, you would have no learning at all where your new value is your original value, V=V.
    - When $\alpha = 1$, you would have absolute learning where you totally forget your previous value, V leaving your new value V = X.

Parameters Initiated

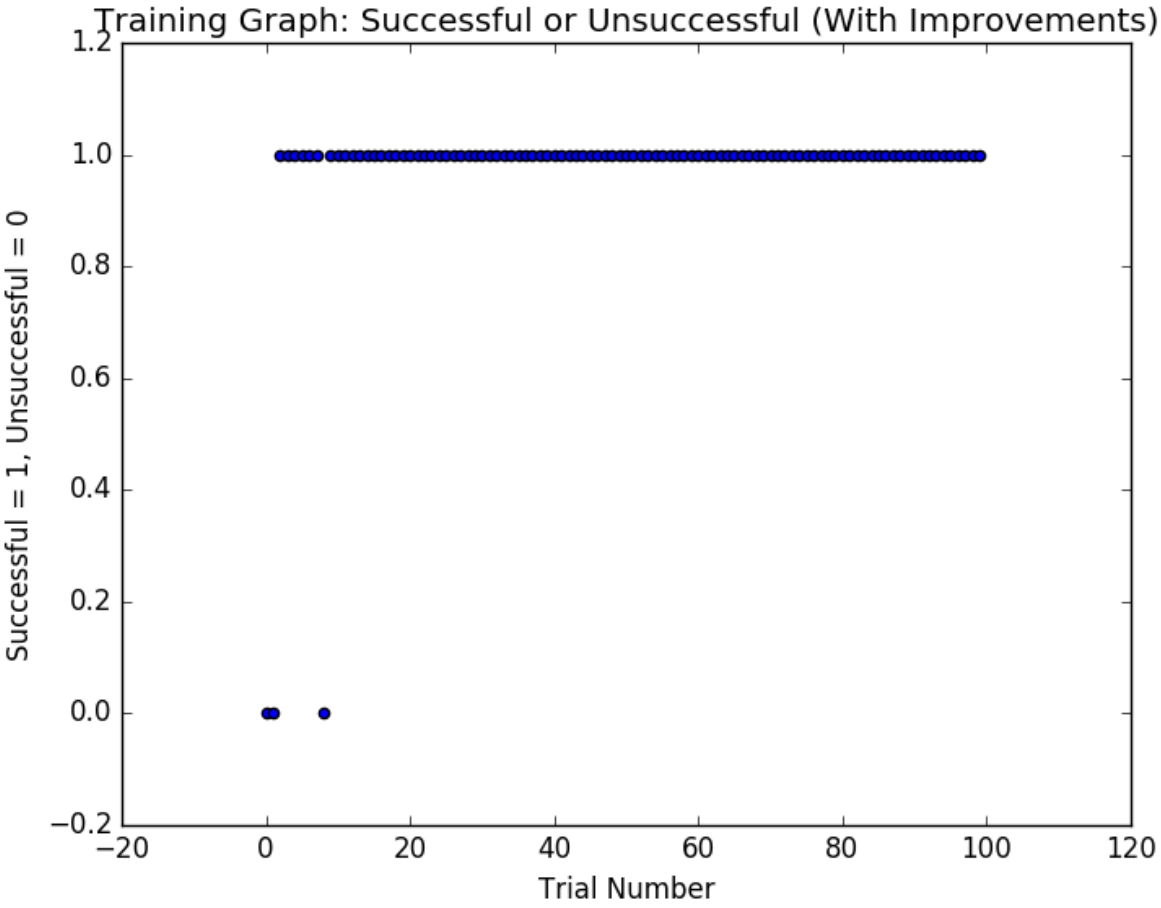
- Alpha (learning rate), is arbitrarily set at 0.3.
- Gamma (discount rate), is arbitrarily set at 0.3.
- Epsilon (randomness probability), is arbitrarily set at 10 such that it is 10%.
  - This is done by randomizing the values of p from 0 to 100.
  - And if p < epsilon, the smart cab would take a random action.
- Q initial values set at 4
  - Although we will suffer more initial penalties trying out every action.
  - This will matter more only when scaling up this reinforcement learning problem to include more dummy agents.
  - This technique is called optimistic initialization.

Trial 100 results

https://gist.github.com/ritchieng/02a2dd735ff4b13dccfeb45fb4e07fe3

Results

- The smart cab reaches the destination more frequently.
- Moreover, as you can see, we've achieved a success rate of 97% with a random assignment of parameters

- 

- And if you look at the results of the 100 trails, we're having fewer violations of traffic rules compared to without learning.

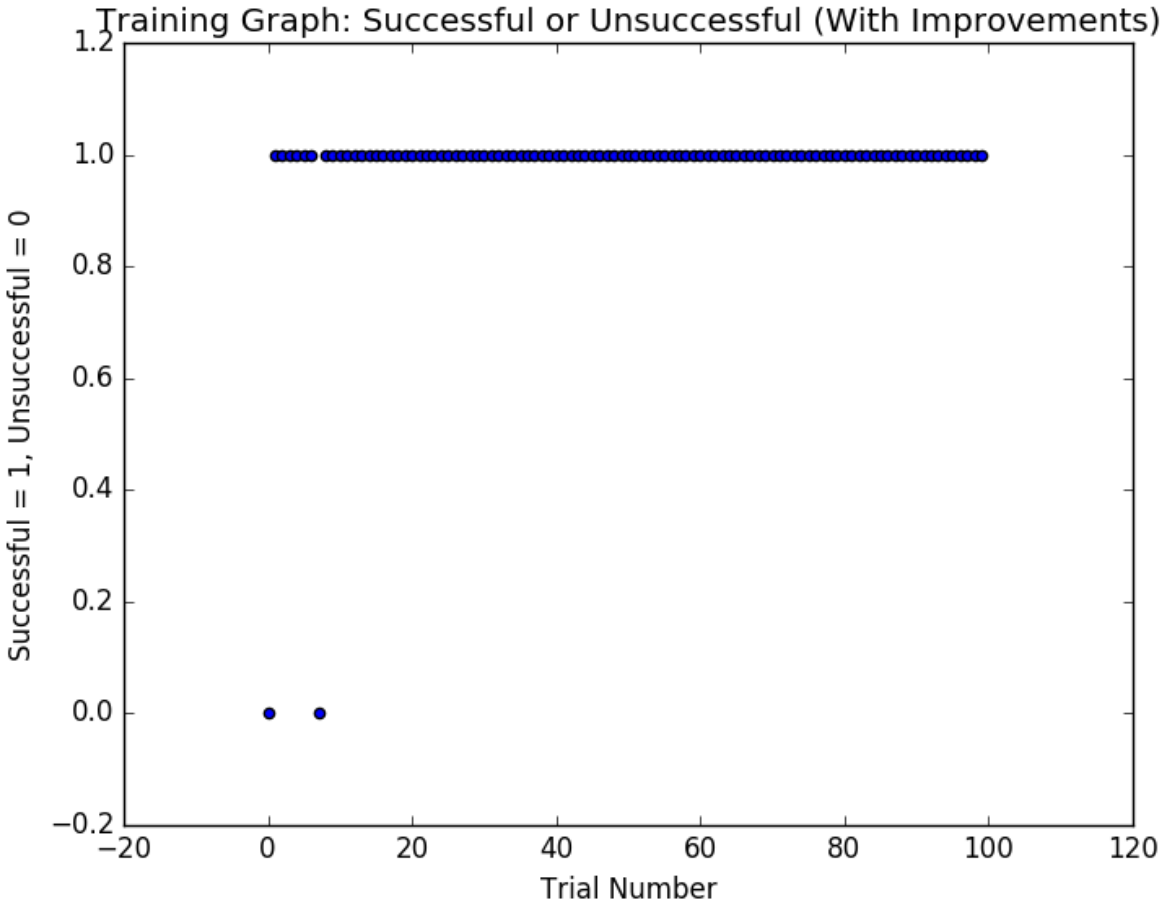
Code

https://gist.github.com/ritchieng/a43b7c188f083bb731efc2e78bd2ec4f

Improve the Q-Learning Driving Agent

Parameter Tuning

- With trial and error, it seems the following parameters allow the agent to perform best.
  - Alpha (learning rate): 0.8
  - Gamma (discount rate): 0.2
  - Initial Q = 4
  - Success rate: 98%

- 

Results of Final Trial

https://gist.github.com/ritchieng/c6f75bad8426f013310e5598a322391a

Results of Prior Trials

https://gist.github.com/ritchieng/6702086652c592a54ebc6e686f0beed5

Optimal Policy

- The agent does reach to the final absorbing states in the minimum possible time while incurring minimum penalties.
- The optimal policy would be:
  - Minimum possible time.
  - Obey all traffic rules.
  - No clashes with other cars.
- In this project, there are two issues resulting from the rewards' system. It benefits the agent to:
  - Disobey traffic when the time is running out to reach the destination.
    - This can be seen from the trials' summary above where when there is a red light, the car might still make a move instead of remaining stationary.
    - This would cause accidents and this is undesirable in the real world.
  - Going in circles when there is a lot of time left to have more rewards before reaching the absorbing state.

- This can be observed how the cab takes a particular long time initially when there is a lot of time. Thereby explaining the high number of moves in the trials' summary above.
- This would cause the smart cab to increase the cab fare in the real world and also delay the trip for the passenger, both of which are undesirable in the real world.