report_pdf.md

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.js

Trial 0: Reached Destination, Exceeded Time Limit

https://gist.github.com/ritchieng/3debad4a7c75b64514ebb8d4c0790dc4.js

Trial 1: Reached Destination, Within Time Limit

https://gist.github.com/ritchieng/4f0948e859640ed4a38a08b47a83c7ed.js

Observations

- If you can see on the first trial, Trial 0, the car took more than double the time of the given deadline of 25 moves.
- The car does eventually reach there with a few caveats:
    i. It took a long time.
    ii. It clashed with other cars.
    iii. It made illegal moves (not obeying the traffic lights).
- Once in awhile, the car does reach the destination before the time is up.

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.
- Irrelevant inputs:
  - Status of traffic at that intersection
    - This is a situation where there are few cars.
    - As such, the values for 'oncoming', 'left', and 'right' should almost always be 0.
    - We can leave this input out to simplify our learning process.
  - 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) and light (red or green)
  - This would result in 3 (next_waypoint) x 2 (light) = 6 states
- 6 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.js

Implement a Q-Learning Driving Agent

Estimating Q from Transitions: Q-learning Equation

- Personal notes:

Estimating Q from Transitions: Q-learning Equation

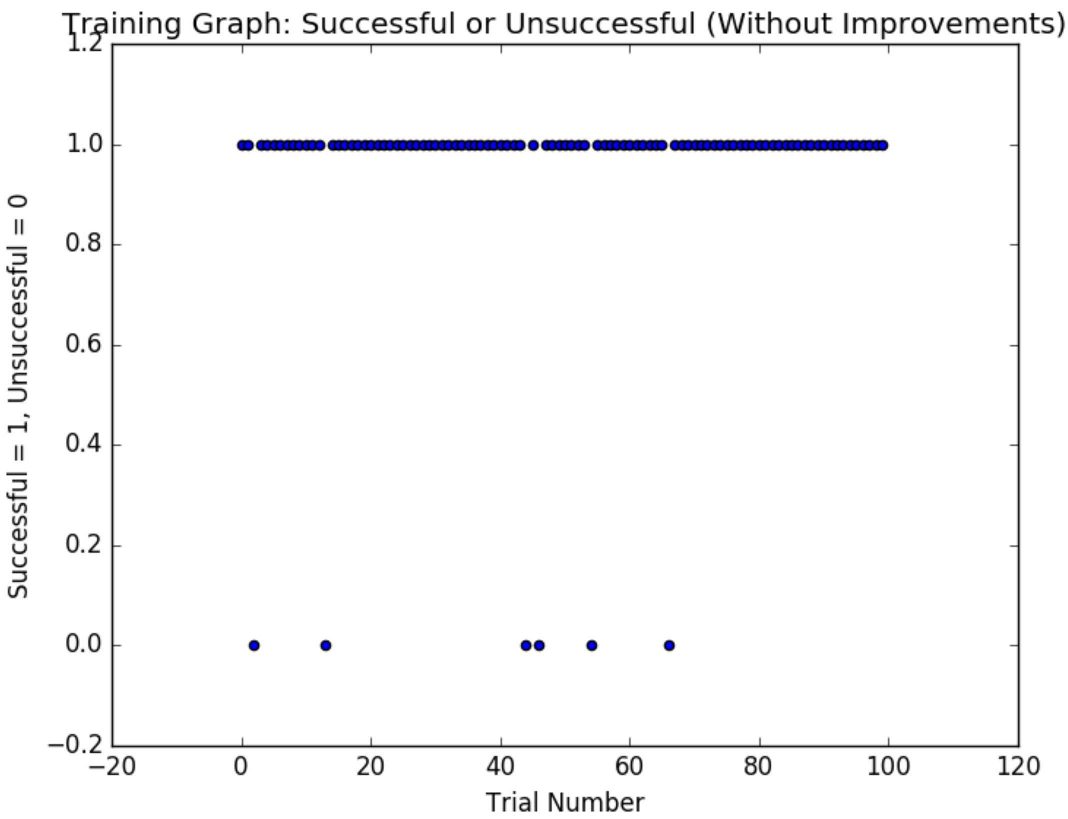
- $$\hat{Q}(s,a) \leftarrow_{\alpha_t} r + \gamma \max_{a'} \hat{Q}(s',a')$$
- **Intuition**: imagine if we've an estimate of the Q function.
  - We will update by taking the state and action and moving a bit.
  - We have the reward and discount the maximum of the utility of the next state.
- This represents 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 0.5.
- Gamma (discount rate), is aribitrarily set at 0.5.
- Epsilon (randomness probability), is arbitrarily set such that it is 1%.
- Q initial values set at 1

Results

- The smart cab reaches the destination more frequently.
- Also, the smart cab takes fewer moves to reach to the destination.
  - The final trial took only 8 steps.
- Moreover, as you can see, we've achieved a success rate of 94% with a random assignment of parameters



Trial 100 results

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

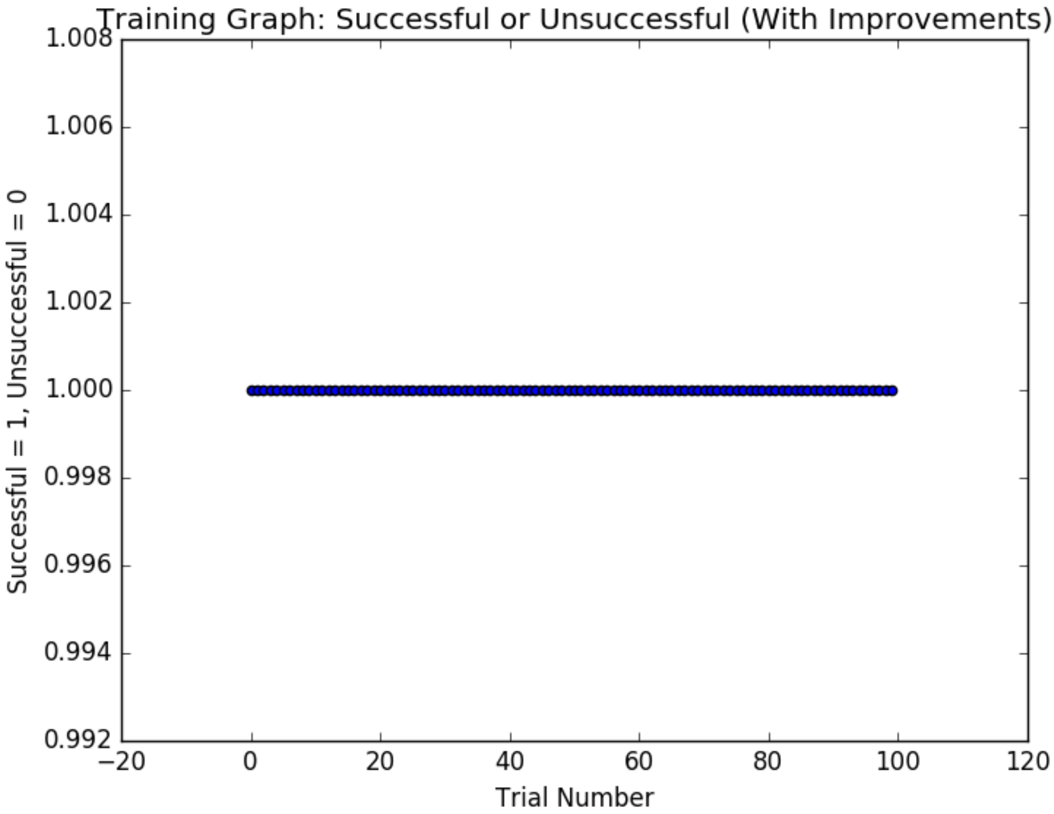
Code

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

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): 0.3
    - Gamma (discount rate): 0.3
    - Initial Q = 1
    - Success rate: 100%

- 

Final trial results

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

Optimal Policy

- The agent does reach to the final absorbing states in the minimum possible time while incurring minimum penalties.
- Yet, it still does violate some traffic rules.
- The optimal policy would be:
  - Minimum possible time.
  - Obey all traffic rules.
  - No clashes with other cars.