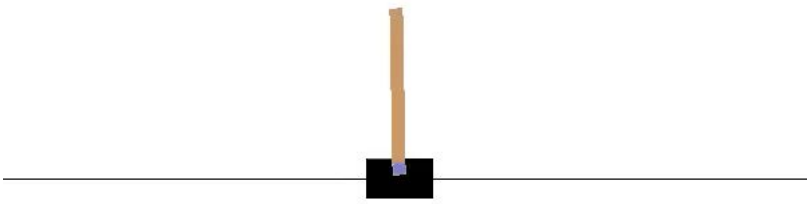
project[3] - Reinforcement Learning

due 12/11/2018 midnight

Work to be done individually, group discussion is allowed

Objectives:

1. To implement a reinforcement learning algorithm that can learn a policy for a given task based on task-based rewards
2. To take a continuous environment and discretize it so that it is suitable for a reinforcement learning task



This is the **CartPole** task. The idea here is to balance this pole using a one-dimensional robot (it can only move left and right). The robot's **state** has 4 components:

- x: the location of the robot (0 is the center, -2.4 is the leftmost part of the board, 2.4 is the rightmost part of the board)
- xdot: the velocity of the robot (technically, this can go from -inf to inf)
- theta: the angle that the pole is at (0 is straight up, -12 degrees or fewer means that the pole falls to the left, 12 degrees or more means that the pole falls to the right)
- thetadot: the change in angle per second.

The robot can choose between two **actions**:

- 0: move left
- 1: move right

Success is balancing for 200 ticks, failure is if the stick falls more than 12 degrees from the median, or if the robot moves more than 2.4 meters from the center.

Your first task is to make a robot learn this task using reinforcement learning (Q-learning).

OpenAI Gym:

You do not have to implement the problem domain yourself, there is a resource called openAI gym which has a set of common training examples. Gym can be installed with the following command:

```
> sudo pip install gym
```

After running the provided command, you may also be asked to install some additional packages for the video encoding. You'll see an error message with instructions to follow.

State Discretization:

We will discretize the space in order to simplify the reinforcement learning algorithm. We could do this a number of ways, but for this I adapted a student project that was pretty simple:

- x: (one bucket for < -.08, one for -.08 < x < .08, one for > .08)
- xdot: (one bucket for <-.5, one for -.5 < xdot < .5, one for > .5)
- theta: divided into six buckets, separated by: -6deg, -1deg, 0, 1deg, 6deg
- thetadot divided into three buckets, separated by: -50deg/s, 50deg/s

These state components are combined into a single integer state (0 ... 161). This is done for you in the function *discretize_state* in the provided code.

Example Code:

can be found in the github repo for the class, '4-rl'

Your Task (part 1):

You need to implement the q-learning part of this task (right now the example code will not do any learning). You need to implement the following equation from the lecture slides:

Q-values can be learned directly from reward feedback

$$Q(a,i) \leftarrow Q(a,i) + \alpha(R(i) + \gamma * max_a Q(a',j) - Q(a,i))$$

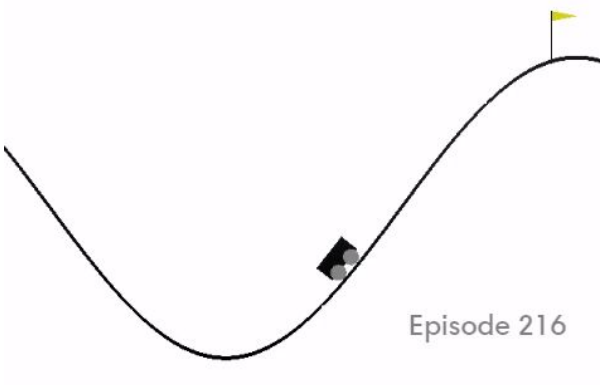
The magic happens (or right now does not happen) on line 137. In this case, the discretized state (s) would be i in this equation, and the next discretized state (sprime) would be j. Reward is stored the variable reward, and the learning rate (α) is in the variable alpha, which is set on line 112. The predicted value of the next state, maxa'Q(a',j), is already computed and stored in the variable predicted_value.

You'll need to implement the equation on line 137 and set the alpha and gamma values (lines 112 and 113, respectively). I'd try stepping the alpha values up and down by factors of .1 in both directions and the gamma values by .01 in both directions to see the effect it has on the learning. You should only have to write one line of code and adjust some values.

The program will train over 50001 episodes and play a video of each 1000th training.

This is easy on the code side, but will allow you to experiment with the various factors of reinforcement learning.

Your Task (part 2):



Now that you've implemented q-learning for one task, you will move to the mountain car task. Instead of 2 actions (left, right), this task has three (left, null, right). The task also has different **state** variables (only 2)

- x: the location of the robot (-1.2 is the left, -.45 is approximately the valley, 0.6 is the rightmost part of the board, 0.5 is the location of the flag)
- xdot: the velocity of the robot (this can go from -0.7 to 0.7)

This will require you to change the number of states and the size of the state variable in a few places:

```
Q = np.zeros([161, env.action_space.n])

discretize_state(state[0], state[1], state[2], state[3])

if tick < 199:
     print "fail ", tick
```

You'll also have to change the discretize_state function to match the new state.

An example video of how this training can proceed is here:

https://youtu.be/81F1tE21BXA

Code should be submitted using the submit script:

```
first, download the submit script:

$ cd ~
$ wget http://www.cse.unr.edu/~newellz2/submit
$ chmod +x submit

then, from your project directory:

$ ~/submit

this will submit all files from the current directory
```

Your turned in directory should have the following files:

```
.
+--   cart.py (the RL python script for part 1)
+--   car.py (the RL python script for part 2)
```

References for python and numpy:

http://cs231n.github.io/python-numpy-tutorial/