1. Policy Gradient Implementation

Starter code is provided here (pg-startercode.py). We have provided a full, working implementation, which works on MDPs with discrete action spaces.

Each iteration, it collects a batch of trajectories. It computes the advantage at every timestep, and concatenates together the observations, actions, and advantages from all timesteps. Then it symbolically constructs the following objective:

$$\sum_n \log \pi(a_t|s_t,\theta)\hat{A_t}$$

and then differentiates it (using Theano) to get the policy gradient estimator.

Here, the policy is parameterized as a neural network with one hidden layer, so the parameters θ are the weights and biases of this neural network.

This code uses a time-dependent baseline, which computes the average return at the t^{th} timestep from the batch of trajectories.

$$\hat{a_t} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots - b_t$$

You can try various things:

1. Experiment with different parameter settings, to get faster or more reliable convergence (See `config` in `REINFORCEAgent` constructor).
2. Try measuring the performance of the policy by using the most likely action, instead of the random action. You should see a big boost in performance.
3. See if you can get this code working on `Acrobot-v0`. It usually helps to make the episodes longer. To do that, modify the line where the agent is constructed:

```
agent = REINFORCEAgent(env.observation_space, env.action_space,
    episode_max_length=env.spec.timestep_limit)
```

4. Generalize it to work with continuous action spaces, by outputting the mean and standard deviation of a Gaussian distribution