Advanced Methods in ML 2017 - Exercise 4

1. In this exercise you will implement and train a reinforcment learning agent for solving Open AI's Lunar Lander environment. The agent will learn how to behave in the environment by updating its parameters via stochastic *policy gradient* ascent.
   Task Explanation:

   - OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. The gym open-source library is a collection of test problems called "environments" that you can use to explore reinforcement learning algorithms. These environments have a shared interface, allowing you to write general algorithms.
   - In the Lunar Lander environment, the agent is a small spacecraft which flies in a two dimensional world. The spacecraft should use its three engines (left, main,right) to land gracefully at the landing pad, which is located at coordinates (0,0). Four discrete actions are available: do nothing, fire left engine, fire main engine, fire right engine. State vector is 8-dimensional, where the first two numbers are the spacecraft's coordinates, and the rest of the vector is various other physical properties. See link above to learn more details about this environment (dynamics, scoring scheme, etc).

   Algorithm Specification:
   The agent to be implemented in this exercise will be a neural network, consisting of two hidden layers with $tanh$ activations. At each step, the input to the network will be the agent's observation of the environment, and the output of the network will be the probabilities from which actions will be sampled.
   More formally, let $\boldsymbol{x} \in \mathbb{R}^8$ be the observation at a given step. The network computes the following:

$$\boldsymbol{h}_1 = tanh(W_1\boldsymbol{x} + \boldsymbol{b}_1) \tag{1}$$

$$\boldsymbol{h}_2 = tanh(W_2\boldsymbol{h}_1 + \boldsymbol{b}_2) \tag{2}$$

$$\boldsymbol{y} = softmax(W_3\boldsymbol{h}_2 + \boldsymbol{b}_3) \tag{3}$$

   Where $\{W_i\}_{i=1}^3$ and $\{\boldsymbol{b}_i\}_{i=1}^3$ are trainable parameters. The dimensions are $W_i \in \mathbb{R}^{n_i, n_{i-1}}$, where n_i is number of neurons in layer i and $n_0 = 8$ (namely the dimension of the input x) and $n_3 = 4$ (since there are four actions and this is the output layer). And $\boldsymbol{b}_i \in \mathbb{R}^{n_i}$. The output $\boldsymbol{y}$ are the probabilities of each action given the state $\boldsymbol{x}$, namely the policy π (note that $\boldsymbol{y}$ is a normalized distribution by construction). For training the network, empirically estimate the gradient of its cumulative rewards then use gradient ascent using any SGD-based optimizer.
   For approximating the gradient, use the *policy gradient theorem*:

$$\nabla_\theta L(\theta) = \mathbb{E}\Big[\sum_{t=0}^{T}\Big(\nabla_\theta \log \pi(A_t|S_t)\sum_{k=t}^{T} r(A_k, S_k)\Big)\Big] \tag{4}$$

   Implementation Guidelines:

   - Go through the Open AI gym tutorial.

- Your agent should be implemented with TensorFlow. For installing it and Open AI gym on nova, view the installation instructions on the course website.
- You can use any optimizer you want, but ADAM should work ok.
- You can use any number of hidden units you want, but using roughly 15 per layer should work ok.
- You can use any batch size and stopping criteria you want, but a batch size of 10 and a stopping criteria of 30K episodes should work ok.
- Your code should finish running at $<$ 12 hours at the worst case.

Submission Guidlines:

- Add your IDs and the path to your codes folder in the following Google Form: https://tinyurl.com/y796vtfu
- Your code should be run using the command 'python lunar_lander.py', and saved as a cPickle file named 'ws.p', containing your agent's parameters.
- Your code should output parameters under which the spacecraft reasonably lands on the landing pad. The average reward of your spacecraft should be at least 180.
- Make sure your code's folder has the proper read permissions.
- Your code should be *readable* and *well-documented*.

Hints and Tips:

- Training can take a long time, so for basic testings of your implementation you can use the CartPole task, which can be solved quickly.
- Agent performance can vary and oscillate through training, so if $\theta_0, ..., \theta_t$ is the series of parameters the learning algorithm went through, you might want at test time to use θ_i for the i which had best perfomance at train time, instead of using θ_t.
- If training performance oscillates too much, consider adding additional step size decreases to your optimizer. TensorFlow has built-in methods for this.
- You are of course free to use the skeleton and tester code at the course website.