PSL432 Theoretical Physiology
Department of Physiology
University of Toronto

ASSIGNMENT 3

Assigned 20 March 2018
Due 3 April 2018

This assignment is worth 35% of your final mark in the course. Please do all parts of both questions. For each question you will submit computer code as m-files. Please email these m-files to douglas.tweed@utoronto.ca and alexandre.guet.mccreight@mail.utoronto.ca by 11:59 p.m. on 3 Apr. Your code will be graded partly on concision and clarity, so keep it brief and orderly. And to avoid confusion, please don't write any new functions with the same names as posted functions, e.g. don't write your own version of forward_relu and call it that; instead, when you write a function, give it a new name.

1. Model-free Learning

Here you will use the deep deterministic policy gradient (DDPG) method to learn to control a two-joint arm. To make the task easier, the arm's dynamics are slightly altered from those of Assignment 1; in continuous time they are now

$$\begin{aligned}\boldsymbol{a} &= \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{\Gamma}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} \\ &= \begin{bmatrix} \psi_1 + 2\psi_2 & \psi_2 + \psi_3 \\ \psi_2 + \psi_3 & \psi_3 \end{bmatrix}\ddot{\boldsymbol{q}} + \psi_2 \sin(q_2)\begin{bmatrix} -\dot{q}_2 & -(\dot{q}_1 + \dot{q}_2) \\ \dot{q}_1 & 0 \end{bmatrix}\dot{\boldsymbol{q}}\end{aligned}$$

The differences are that the $\cos(q_2)$ terms have been omitted, and the constants ψ_1, ψ_2, and ψ_3 are *100 times smaller* than before: $\psi_1 = 0.02$, $\psi_2 = 0.0065$, $\psi_3 = 0.0056$. (This change in $\boldsymbol{\psi}$ means we are measuring $\boldsymbol{a}$ in units of hundreds-of-spikes per second rather than spikes per second, which means the $\boldsymbol{a}$'s will now be a hundred times smaller than before, and therefore about the same size as the $\boldsymbol{s}$'s. Having variables roughly the same size speeds up learning.)

You will implement these state dynamics in a function called yoursurname_arm.m that takes `s` and `a` as inputs, and puts out the rate of change of the state, i.e. you will Euler-update the arm in this way:

```
s = s + Dt*yoursurname_arm(s, a);
```

where `Dt` = 0.01.

You should write your dynamics function so it can handle `s` and `a` when they are *minibatches*. If possible, write it so it handles minibatches without using any loops (e.g. without processing each column of the minibatch individually, one after the other, in a for-loop); your code will be neater and much faster if you can do that.

The function must also prevent the joints going outside their motion ranges, defined by `q_min = [-1.5; 0]` and `q_max = [1.5; 2.5]` as in Assignment 1.

Use *global variables* to pass needed information (other than the inputs, `s` and `a`) to your dynamics function, i.e. write

```
global Dt psi q_min q_max
```

as the first line (after the title) in yoursurname_arm.m, and right after `clear variables; clc;` in your main m-file. Then if you define those variables in your main file, they will also be available to your dynamics function, without your having to pass them as inputs.

The cost-rate for this question will be quadratic, $c = \boldsymbol{q}^T\boldsymbol{q}$. You will compute it using a function `c(s, a)`, defined inside your main m-file, and also written to handle *minibatch* `s`'s and `a`'s.

To make it easier to compare your results with mine and with other students', please do not create your own test set or policy network `mu`, but instead download the file A3_infile.mat from the course website and load it into your main m-file, using code something like this:

```
folder = 'C:\\PSL432\\';  % folder with A3_infile
infile = sprintf('%sA3_infile.mat', folder);
load(infile);
```

As soon as you load that file, your program will have access to an initial policy network `mu` and a matrix `s_test` containing initial states for the 100 examples in your test set.

Please *do not* use tanh to bound the actions, but instead make $\boldsymbol{a}$ equal the output of the network `mu`.

Run 50 rollouts in all, each one a minibatch of 100 examples (different from those in `s_test`), and each with a duration of 0.3, i.e. 30 times `Dt`. Before rollouts 1, 10, 20, 30, 40, and 50, run your current policy on the downloaded test set of 100 examples, and print the mean cost over those 100 movements. You don't need to plot anything.

If your state-dynamics function and cost-rate are correct then the initial, mean cost over the test set, right before the first rollout, should be 0.5465, to 4 decimal places. If you program DDPG correctly, the mean cost should fall below 0.544 in at least one of the 5 tests after that. (I got these results using `eta_mu = 3e-8`, `eta_Q = 1e-4`, `tau = 3e-5`, `a_sd = 0.1`, and a 4-layer `Q_est` network with 400 neurons in each of the hidden layers.)

Name your variables as in these pages, the notes, and the sample code, e.g. `q`, `psi`, `M`, `GAMMA`, `s`, `a`, `Q_est`, `Q_tgt`, `mu_tgt`, `eta_mu`, `eta_Q`, `tau`, etc.

Submit m-files called Yoursurname_A3_Q1 and yoursurname_arm.

2. Model-based Learning

Here you will write a modified version of the Lagrange policy gradient (LPG) method to learn to control the same two-joint arm as in Question 1, though with a slightly different cost-rate. Please compute the state dynamics using the same function, yoursurname_arm.m, that you used in Question 1.

The cost-rate will be $c = \boldsymbol{q}^T\boldsymbol{q} + 0.001\,\boldsymbol{a}^T\boldsymbol{a}$, i.e. the agent still cares mainly about getting $\boldsymbol{q}$ to $\boldsymbol{0}$, but now it also cares a little about keeping $\boldsymbol{a}$ small, to save energy. Again you will compute the cost-rate using a function `c(s, a)`, defined inside your main m-file and written to handle minibatch `s`'s and `a`'s.

To simplify things, you will modify LPG so that it does not have to learn the cost-rate c (or c'), but knows the exact formula for c and its gradient from the outset. So everywhere in the rollouts where the posted file Lagrange_policy_gradient.m uses the network `c_prime_est` or its gradient, you will instead use the exact cost-rate function `c(s, a)` and its gradient.

As in Question 1, you will load the file A3_infile.mat. As soon as you load that file, your program will have access to an initial policy network `mu` and the matrix `s_test`, as in Question 1. And it will also have a network `f_est` which has already learned to estimate the state dynamics. That is, *your program should not learn* `f_est` *at all, but should simply use the one provided by* A3_infile.mat. This way, your code will run faster and it will be easier to compare your results with mine and other students'.

If you use Lagrange_policy_gradient.m as your template, be sure to remove extraneous code from your m-file, e.g. code that is used to train `c_prime_est` and `f_est`. Also note that Lagrange_policy_gradient.m and the pseudocode in the notes deal with a case where c is a function of $\boldsymbol{s}$ alone, not $\boldsymbol{a}$, while here it depends on $\boldsymbol{a}$ as well. Further, the pseudocode and posted code have $\boldsymbol{a}$ bounded by tanh whereas here, as in Question 1, you should *not* bound the actions, but instead make $\boldsymbol{a}$ equal the output of the network `mu`. Your code should differ from the posted code to take these facts into account.

Run 100 rollouts in all, each one a minibatch of 100 examples (different from those in `s_test`), and each with a duration of 0.3, i.e. 30 times `Dt`. Before rollouts 1, 10, 20, 30, 40, and 50, run your current policy on the downloaded test set of 100 examples, and print the mean cost over those 100 movements. You don't need to plot anything.

If your state-dynamics function and cost-rate are correct then the initial, mean cost over the test set, right before the first rollout, should again be 0.5465, to 4 decimal places. If you program LPG correctly, the mean cost should fall below 0.52 in at least one of the 10 tests after that. (I got these results using `eta_mu = 3e-6`.)

Submit an m-file called Yoursurname_A3_Q2.