PSL432 Theoretical Physiology
Department of Physiology
University of Toronto

ASSIGNMENT 3

Assigned 19 March 2024
Due 2 April 2024

This assignment is worth 35% of your final mark in the course. Please email your submission, in the form of an m-file, to douglas.tweed@utoronto.ca and ankit.roy@mail.utoronto.ca by 11:59 p.m. on 2 April. Your code will be graded partly on concision and clarity, so keep it brief and orderly.

Here you'll use a variant of the deep deterministic policy gradient algorithm to learn to control a two-joint arm. The arm will have the same dynamics as in Assignment 1 except that the time step Δt will now be 0.1 s. Whereas my posted code file DDPG.m assumes that Δt = 1 (and therefore doesn't mention `Dt` at all), you will introduce a `Dt` variable and write your code so it works with a `Dt = 0.1`.

You'll implement the state dynamics in a function called `f` which you'll place at the bottom of your m-file. (To see how to define a function at the bottom of an m-file, look at DDPG.m).

Your `f` function will take the state and action, `s` and `a`, as inputs, and put out the state at the next time step, `s_next = f(s, a)`. Your `f` should handle single examples of `s` and `a`, not minibatches. It should also prevent the joints going outside their motion ranges, defined by `q_min = [-2; 0]` and `q_max = [1.5; 2.5]`, as in Assignment 1. And it should Euler-update `q` before `q_vel` and `q_acc`.

Also at the bottom of your m-file, please insert the function `test_policy` from the sample code DDPG.m, but modify it so it calls your `f` function in the appropriate place.

Use global variables to pass information other than their inputs to your `f` and `test_policy` functions, i.e. write

```
global n_q psi zeta q_min q_max n_steps Dt r
```

in your m-file right after `clear variables;` and then, inside `f` and `test_policy`, list the global variables they need (much as I did for `test_policy` in DDPG.m).

Define your reward function this way:

```
r = @(s, a) -10*(s - s_star)'*(s - s_star) - 0.01*a'*a;
```

where `s_star = [0.5; 0.5; 0; 0; 0; 0]`. When computing the return `G`, use no discount factor, or in other words set $\gamma = 1$.

To make it easier for Ankit and me to compare your results with mine and with other students', please seed the random number generator by writing `rng(4)` into your code right before you create your networks. Create all your networks using the posted function create_net.m.

In this variant of DDPG, you will *not* use a `Q_est` network. Rather, you'll decompose the action-value function Q into a combination of more-basic functions, in the hope that by exploiting its internal structure you can improve learning. This question will be an exercise in making multiple networks operate together, and particularly in figuring out what signals to backpropagate through which networks.

The decomposition of Q will work like this. First, we define the *value* function $V^\mu(s) = Q^\mu(s, \mu(s))$, i.e. $V^\mu(s)$ is simply $Q^\mu(s, a)$ when a is chosen to fit the policy μ, or in other words it's the return we'll get from now on if we're currently in state s and we choose all our actions, including the current one, using policy μ.

Next, we observe that

$$Q^\mu(s_t, a_t) = r(s_t, a_t)\,\Delta t + V^\mu(s_{t+\Delta t}) = r(s_t, a_t)\,\Delta t + V^\mu(f(s_t, a_t)),$$

i.e. $Q^\mu = V^\mu \circ f + r\,\Delta t$. Your job is to code a variant of DDPG that has no $\langle Q \rangle$ net but instead works with networks $\langle V \rangle$, $\langle f \rangle$, $\langle r \rangle$, and also target networks V^+ and f^+ (but no r^+ or Q^+).

Each of your six networks should have four layers and relu activation. The networks $\langle f \rangle$, f^+, $\langle V \rangle$, and V^+ should have 100 neurons in each hidden layer, $\langle r \rangle$ should have 50, and μ should have 250. Initialize all your networks' weights using (as the second input to create_net) the rescale vector [0; 0.5; 0.5; 0.5]. Create the networks immediately after the line `rng(4)`, and in this order: μ, $\langle f \rangle$, $\langle r \rangle$, $\langle V \rangle$, and then the target networks.

If you structure and initialize your μ network correctly, then the first five columns of `mu.W{end}`, before learning begins, should equal

```
  -0.0338    0.0159   -0.0517   -0.0349    0.0364
   0.0100    0.0020   -0.0657    0.0905    0.0163.
```

You should train $\langle r \rangle$ in the obvious way based on minibatches drawn from the buffer, using a loss function that is half the square of the error $e^r = \langle r \rangle(s\_, a\_) - r\_$ where $s\_$, $a\_$, and $r\_$ are from the buffer.

Not so obviously, you should train $\langle f \rangle$ using a loss function that is half the square of

$$e^f = \langle V \rangle(\langle f \rangle(s\_, a\_)) - \langle V \rangle(s_{\text{next}}\_),$$

i.e. $\langle f \rangle$ needn't necessarily predict the correct $s_{\text{next}}\_$, but it should predict a new state whose estimated *value* matches that of the true $s_{\text{next}}\_$ (because that's easier than predicting $s_{\text{next}}\_$).

You will have to work out how to train $\langle V \rangle$ — by finding an equation analogous to the Bellman equation but holding for $\langle V \rangle$ rather than $\langle Q \rangle$, and then rearranging your equation to define an error signal analogous to the Bellman error but for $\langle V \rangle$. You will also have to figure out how to adjust the policy network μ using the networks you have.

Use the hyperparameters `eta_mu = 3e-8`, `eta_f = 1e-4`, `eta_r = 1e-2`, `eta_V = 1e-4`, `tau = 3e-4`, `a_sd = 0.1`. And for all learning, use the adam optimizer with its standard hyperparameters.

Run 1000 rollouts in all, each with a duration of 2 s, i.e. 20 times `Dt`. Before learning begins, and then after every 100 rollouts, call `test_policy` using `s_test = [-1; 1; 0; 0; 0; 0]`, and also call batch_nse.m to compute `r_nse`, `f_nse`, and `V_nse` on the most-recent minibatch, and print those three variables alongside the `G` from `test_policy`.

If your `f` and `r` functions and your `mu` network are correct, then you should get `G = −92.8` (to one decimal place) the first time you call `test_policy`. If you program the learning correctly, you should have `G > −65` and `r_nse`, `f_nse`, and `V_nse < 0.01` by 1000 rollouts.

Please name your variables as in these pages, the notes, and the sample code, e.g. `q`, `psi`, `zeta`, `M`, `GAMMA`, `s`, `a`, `mu`, `f_est`, `f_tgt`, `r_est`, `V_est`, `V_tgt`, `eta_mu`, `eta_V`, `tau`, etc.

Submit an m-file called YOURINITIAL_YOURSURNAME_A3.