Imperial College London

November 29, 2022

Reinforcement Learning – Prof Aldo Faisal & Dr Paul Bilokon
Assignment design: Filippo Valdettaro

Lab Assignment 4: Deep Deterministic Policy Gradients

This lab assignment provides a framework for you to get started with exploring reinforcement learning in a continuous control setting. To reduce the overhead to get started, we are providing a continuous version of the CartPole environment which you are already familiar with from Lab Assignment 3 and Coursework 2.

The main code for this lab assignment is provided in the `DDPG.ipynb` notebook, while the Colab version is available at:
`https://colab.research.google.com/drive/17rOIuXF0THUjErdQ60pM8ts8maNuND6c?usp=sharing`

NOTE: with the hyperparameters provided "out of the box" DDPG takes significantly longer than DQN to train. You should expect to see visible improvements in your agent after roughly 1000 episodes (approximately just over 5 mins run on Colab with GPU enabled). After that point, episodes will last longer so running many more episodes will take considerable time. You may consider setting an early stopping condition or reduce the number of episodes if you don't want to wait for long training times.

1 Continuous Control

The CartPole class has been modified so that any continuous impulse can be applied, rather than just +1 or -1. Now, instead of passing an integer 0 or 1 as action, any real number representing the force exerted on the cartpole can be passed. Therefore, we have moved from a discrete action setting to a continuous action setting, and must move away from DQN and implement some other reinforcement learning algorithm.

1.1 Proportional control

Before starting with deep reinforcement learning, you may want to investigate whether a proportional control approach works for this environment. For example, does setting force to be proportional to angular displacement from the vertical give good results? Is this surprising?

2 Deep Deterministic Policy Gradients (DDPG)

You will have come across the Deep Deterministic Policy Gradients algorithm in lectures. We provide a template notebook that implements almost everything you need to run DDPG. The only component missing are the optimisation steps in the Actor and Critic classes. However, in order to complete this step, you will have to have a good grasp on how the whole code works, so you should try to follow how the rest of the implementation functions.

2.1 Actor and Critic optimisation

Both actor and critic have trainable model attributes. These models both share the same MLP class, but they do not share the same shapes. In addition, the actor's output is confined to be within a particular range, so that the actor can't exert an arbitrarily large impulse on the CartPole.

In the `Actor` and `Critic` classes, you should complete the `optimise_step` method. The arguments of the function should be clear on a high level from their variable names, but to find exactly what form they take (or shape in the case of tensors), you should understand where these steps are called and what these variables are in the larger context of the DDPG algorithm implementation.

You should complete these methods so that they modify the actor and critic's policy nets respectively, by updating their weights according to the DDPG algorithm.

3 Training

Once you are confident that your implementation will work as expected, train your agent. To see significant improvements with the provided hyperparameters, you will have to run the code for at least roughly 1000 episodes. It is advisable to use Colab's GPU capabilities if you don't have a GPU locally and would like to speed up the training process.

Experiment with hyperparameters and see whether you manage to improve the training speed or results of your agent. Would you consider this a "difficult" task to solve (think back to proportional control) - does the training time surprise you or is it expected?