EE619 PROJECT #3

1. Introduction

Implement an RL algorithm for continuous control—any algorithm of your choice—from scratch. For those of you who are having difficulties in deciding which algorithm to implement, we give you a list of suggested RL algorithms.

- Proximal Policy Optimization Schulman et al. [2017]
- Twin Delayed Deep Deterministic policy gradient Fujimoto et al. [2018]
- Soft Actor-Critic Haarnoja et al. [2018]

We provided a sample code that is an implementation for the REINFORCE Williams [1992] algorithm. Refer to the attached `README.md` before running the sample code.

2. Specifications

Language to Use: Python 3.8 or higher

Files: The `ee619_project3.tar.gz` archive consists of three files.

- `evaluate.py`
  The script that will be used for testing your agent's performance. DO NOT modify this file.
- `ee619/__init__.py`
  `__init__.py` file for the ee619 module.
- `ee619/agent.py`
  A sample implementation of an agent that takes a certain action at every step.

As you can see from `agent.py` the Agent class has the following two public methods:

- `act(time_step)` : This method takes the current time-step as an argument and returns the action the agent should take at this step. `time_step` is a namedtuple object with four fields: `step_type` , `reward` , `discount` , `observation` . You may access each field using the dot operator, e.g. `time_step.reward` . Note that `time_step.observation` is an OrderedDict of NumPy arrays, so you would need to convert it into a single NumPy array before feeding it to the policy network. You may use the function `flatten_and_concat` implemented in `agent.py` if you want. The following code shows the basic usage of `flatten_and_concat`.

  ```
  observation = flatten_and_concat(time_step.observation)
  logits = self.policy(torch.as_tensor(observation))
  ```

  For further information about the TimeStep object, please refer to Tassa et al. [2018].
- `load()` : This method, when called, should load all of the neural network parameters.

3. Testing

Your agent will be evaluated on the Walker domain using the `run` task provided by the Deep-Mind Control Suite Tassa et al. [2018]. Refer to `evaluate.py` for the exact details as the script will be used for evaluating your agent.
In order to run the script, you will need the Python package `dm_control` Tunyasuvunakool et al. [2020], which can be installed by running the following command.

```
pip install wheel dm_control
```

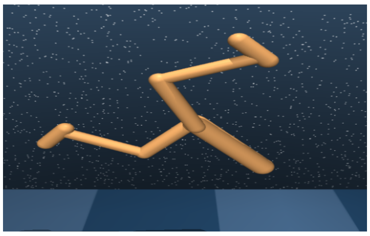

Figure 1: A screenshot of a DMControl walker domain.

4. What to Submit

You have to submit the 2 files below.

- (50 Points) `report.pdf`
  A short report that contains the following:
  - (25 Points) A detailed explanation about the algorithm you chose to implement
  - (10 Points) Plots that track the training progress of your agent, which includes but are not limited to training loss plots and average episode return plots
  - (10 Points) Your own analysis of your experimental results
  - (5 Points) Hyper-parameters for your implementation, which have to include your random seeds for training
- (50 Points) `STUDENT_ID.tar.gz` or `STUDENT_ID.zip`
  Your compressed file should files below; Do not compress files other than files below.
  - Your trained model
    - Naming & File Format: `trained_model.[saving format]`
    - For example, `trained_model.pt`
  - Your `agent.py`
    - Naming: `agent.py`
  - (Optional) Either `requirements.txt` or `environment.yml` if you needed
    - Naming: `requirements.txt` or `environment.yml`
  - (Optional) Files needed to evaluate your trained model;
    Read the instructions below carefully. We are not responsible when your trained model is not valid in the `evaluate.py` we provided.

We will evaluate your trained models over 10 random seeds, which will not be disclosed to you. We'd like to mark your trained model up to 50 points based on the evaluation performance.

Before evaluating your trained models, we will create a directory named your `STUDENT ID`. We will extract your compressed file and place the provided `evaluate.py` into that directory. Then, we will create the directory structure as shown in the example below.

```
20990001/
├── ee619/
│   ├── trained_model.pt
│   ├── agent.py
│   ├── requirements.txt
│   ├── others
│   └── __init__.py
└── evaluate.py
```

Finally, to evaluate your trained model, we will execute the following command in the `STUDENT ID` directory:

```
python evaluate.py
```

Please make sure that your trained model runs well with the `evaluate.py`, which we provided, in the directory setting. Also, make sure that the model file is inside the `ee619` directory and your `agent.py` loads the model from that file. If you load from somewhere else, e.g. from the parent directory of `ee619`, the test script may not work.

If your `agent.py` requires third-party libraries other than `dm_control` and `PyTorch` in order to run, compress a `requirements.txt` or an `environment.yml`. You can create `requirements.txt` by running

```
pip freeze > requirements.txt
```

and `environment.yml` by running

```
conda env export > environment.yml
```

You can use the following command to compress regardless of whether or not you have to provide an environment specification file:

```
tar --ignore-failed-read -zcvf project3.tar.gz \
    ee619/* requirements.txt environment.yml
```

References

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018. URL https://proceedings.mlr.press/v80/fujimoto18a.html.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL http://arxiv.org/abs/1707.06347.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018. URL http://arxiv.org/abs/1801.00690.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. URL https://www.sciencedirect.com/science/article/pii/S2665963820300099.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pages 5–32, 1992. URL https://link.springer.com/article/10.1007/BF00992696.