CMP9137M Advanced Machine Learning – Workshop Week 6

UNIVERSITY OF LINCOLN

Summary: This workshop aims to give you practical experience with deep reinforcement learning using value-based and policy-based methods. First, you will test and train agents using a well-known benchmark called "LunarLander". Then you will test and train image-based agents playing the game of "VizDoom". The materials of this workshop make use of the widely-used and open-source toolkit [Stable-Baselines3](). Last but not least, you will play the game of VizDoom yourself and will have the choice to record your game actions for training a supervised policy (your own benchmark). Please work in groups of two or three students to achieve the tasks efficiently and to discuss the outcomes.

The materials for today are in Blackboard, assessments, item 1, file `VizDoom-DRL-task2.zip`.

Task 1: **Train and evaluate LunarLander agents**

To familiarise yourself with the states, actions, rewards and task of these agents, you should read the following in parallel to the commands below, which will give you an appreciation for the non-trivial decision-making that is required: Lunar Lander - Gym Documentation (gymlibrary.dev)

Edit the file `sb_VizDoom.py` and make sure the **LunarLander-v3** environment has been chosen.

You should be able to train an agent using one of the following commands:

- `python sb_VizDoom.py train DQN`
- `python sb_VizDoom.py train A2C`
- `python sb_VizDoom.py train PPO`

You can evaluate the performance of newly trained or pre-trained agents as follows (see your files to identify seed values—your downloaded files contain a couple of pre-trained agents):

- `python sb_VizDoom.py test DQN SEED_NUMBER`
- `python sb_VizDoom.py test A2C SEED_NUMBER`
- `python sb_VizDoom.py test PPO SEED_NUMBER`

What results did you get? Record your results and compare them to gain some understanding in terms of performance against amount of training steps and training/test time, among others.

Algorithm	Avg. Reward using 10K steps	Avg. Reward using 200K steps
DQN	total_reward=[-1682.0576] avg_reward=[-84.10288] 792	total_reward=[-858.5688] avg_reward=[-42.92844] 166
A2C	total_reward=[-4351.8555] avg_reward=[-217.59277] 79	total_reward=[-602.48663] avg_reward=[-30.124332] 674
PPO	total_reward=[-12238.657] avg_reward=[-611.93286] 622	total_reward=[4664.1206] avg_reward=[233.20602] 260

Task 2: **Train and evaluate VizDoom agents**

Do the same as in task 1 but using the environment called "**VizdoomTakeCover-v0**".

What results did you get? Record your results discuss them with your peers/lecturer/demonstrator.

Algorithm	Avg. Reward using 10K steps	Avg. Reward using 400K steps
DQN		
A2C		
PPO		

UNIVERSITY OF LINCOLN

Task 3: **Train a behaviour cloning agent—via supervised learning—and evaluate its performance**

The program of this task requires you to play the role of expert demonstrator of VizDoom. For every state in the game environment, you have to provide an action by pressing the LEFT arrow or RIGHT arrow on your keyboard in the terminal screen. Only two actions are enabled for this task, but you can change them later on to a higher number of actions depending on the environment. Once an episode is over, the program will ask you to save the data of the episode or not. You want to save episodes with reasonable game scores (>700 points, as good demonstrations instead of bad ones).

First, run the following to play the game and collect data from a few/several episodes:

- `python bc_VizDoom_FromDemonstration.py train human`

Once data has been collected, you should train your behaviour cloning agent as follows:

- `python bc_SupervisedPolicy.py`

After executing the program above without errors, you can evaluate its performance as follows:

- `python bc_VizDoom_FromDemonstration.py test human`
- `python bc_VizDoom_FromDemonstration.py random`

Task 4 [homework]: **Get familiarised with the API of Stable-Baselines3**

4.1 Read the parameters (and default values) of DQN, A2C and PPO implementations. Links:

- classstable_baselines3.dqn.DQN(policy, env, learning_rate=0.0001, buffer_size=1000000, learning_starts=100, batch_size=32, tau=1.0, gamma=0.99, train_freq=4, gradient_steps=1, replay_buffer_class=None, replay_buffer_kwargs=None, optimize_memory_usage=False, target_update_interval=10000, exploration_fraction=0.1, exploration_initial_eps=1.0, exploration_final_eps=0.05, max_grad_norm=10, stats_window_size=100, tensorboard_log=None, policy_kwargs=None, verbose=0, seed=None, device='auto', _init_setup_model=True)

- classstable_baselines3.a2c.A2C(policy, env, learning_rate=0.0007, n_steps=5, gamma=0.99, gae_lambda=1.0, ent_coef=0.0, vf_coef=0.5, max_grad_norm=0.5, rms_prop_eps=1e-05, use_rms_prop=True, use_sde=False, sde_sample_freq=-1, rollout_buffer_class=None, rollout_buffer_kwargs=None, normalize_advantage=False, stats_window_size=100, tensorboard_log=None, policy_kwargs=None, verbose=0, seed=None, device='auto', _init_setup_model=True)

- classstable_baselines3.ppo.PPO(policy, env, learning_rate=0.0003, n_steps=2048, batch_size=64, n_epochs=10, gamma=0.99, gae_lambda=0.95, clip_range=0.2, clip_range_vf=None, normalize_advantage=True, ent_coef=0.0, vf_coef=0.5, max_grad_norm=0.5, use_sde=False, sde_sample_freq=-1, rollout_buffer_class=None, rollout_buffer_kwargs=None, target_kl=None, stats_window_size=100, tensorboard_log=None, policy_kwargs=None, verbose=0, seed=None, device='auto', _init_setup_model=True)

4.2 Look at the source code of each of the following: DQN, A2C, PPO.