AI Traineree

Dawid Laszuk

Nov 21, 2021

CURRENT CONTENTS:

As you may have noticed, current version of the documentation is rather modest. True. But be patient, young one, as this is only the beginning. Hopefully.

It is always a motivational boost and a driver to see that someone is using the project. Feel free to let me know if you have any questions or anything, and I'll sure try to help.

CHAPTER

ONE

GETTING STARTED

1.1 What is this?

Have you heard about DeepMind or recent advancments in the Artificial Intelligence like beating Go game, StarCraft 2 or Dota2? The AI Traineree is almost the same. Almost in the sense that it's unlikely to achieve the same results and those algorithms aren't provided (yet) but at least we use the same terminology. That's something, right?

AI Traineree is a collection of (some) Reinforcement Learning algorithms. The emphasis is on the Deep part, as in Deel Learning, but there are/will be some of more traditional algorithms. Yes, we are fully aware that there are already some excelent packages which provide similar code, however, we think we still provide some value especially in:

- **Multi agents.** The goal is to focus on multi agent environments and algorithms. It might be a bit modest right now but that's simply because we want to establish a baseline.
- **Implementation philosophy.** Many look-alike packages have the tendency to pass environment as an input to agent's instance. We consider this a big no-no. The agent lives in the environment, it lives thanks to the environment. Such distinction already makes algorithms' implementations different.

1.2 Installation

Currently the only way to install the package is to download and install it from the GitHub repository, i.e. https://github.com/laszukdawid/ai-traineree.

Assuming that this isn't your first git project, the steps are:

```
$ git clone https://github.com/laszukdawid/ai-traineree.git
$ cd ai-traineree
$ python setup.py install
```

1.3 Issues or questions

Is there something that doesn't work, or you don't know if it should, or simply have a question? The best way is to create a github issue (https://github.com/laszukdawid/ai-traineree/issues).

Public tickets are really the best way. If something isn't obvious then it means that others must have the same question. Be a friend and help them discover the answer.

In case you want some questions or offers that would like to ask in private then feel free to reach me at ai-traineree@dawid.lasz.uk .

1.4 Citing

If you found this project useful and would like to cite then we suggest the following BibTeX format.

```
@misc{ai-traineree,
    author = {Laszuk, Dawid},
    title = {AI Traineree: Reinforcement learning toolset},
    year = {2020},
    publisher = {GitHub},
    journal = {GitHub repository},
    howpublished = {\url{https://github.com/laszukdawid/ai-traineree}},
}
```

CHAPTER

TWO

EXAMPLES

2.1 Single agent

2.1.1 DQN on CartPole

This example uses the *CartPole* environment provided by the OpenAI Gym. If you don't have the *Gym* then you can install it either through `pip install gym`.

```
from ai_traineree.agents.dqn import DQNAgent
from ai_traineree.runners.env_runner import EnvRunner
from ai_traineree.tasks import GymTask

task = GymTask('CartPole-v1')
agent = DQNAgent(task.obs_space, task.action_space, n_steps=5)
env_runner = EnvRunner(task, agent)

# Learning
scores = env_runner.run(reward_goal=100, max_episodes=300, force_new=True)

# Check what we have learned by rendering
env_runner.interact_episode(render=True)
```

2.2 Multi agent

2.2.1 IQL on Prison

This example uses the Prison environment provided by the PettingZoo. The *Prison* is simple environment where all agents are independent with a simple task alternatively touch walls. To install the environment execute `pip install pettingzoo[butterfly]`.

```
from ai_traineree.multi_agent.iql import IQLAgents
from ai_traineree.runners.multiagent_env_runner import MultiAgentCycleEnvRunner
from ai_traineree.tasks import PettingZooTask
from pettingzoo.butterfly import prison_v2 as prison

env = prison.env(vector_observation=True)
task = PettingZooTask(env)
task.reset()
```

(continues on next page)

(continued from previous page)

```
config = {
    'device': 'cpu',
    'update_freq': 10,
    'batch_size': 200,
    'agent_names': env.agents,
}
agents = IQLAgents(task.obs_space, task.action_space, task.num_agents, **config)

env_runner = MultiAgentCycleEnvRunner(task, agents, max_iterations=9000, data_
↪logger=data_logger)
scores = env_runner.run(reward_goal=20, max_episodes=50, eps_decay=0.95, log_episode_
↪freq=1, force_new=True)
```

2.3 More examples

Here are only some selected examples. There are many more examples provided in the repository as individual files. There is *examples* directory or directly here https://github.com/laszukdawid/ai-traineree/tree/master/examples.

The easiest way to run them is to checkout git package and install it (see note below). Examples can be run as modules from the root directory, i.e. directory with `setup.cfg` file. To run *cart_dqn* example execute:

```
$ python -m examples.cart_dqn
```

Note: Examples use some libraries that aren't provided in the default package installation. To install all necessary packages make sure to install AI Traineree with `[examples]` conditions. If you are using *pip* to install packages then you should use `pip install -e .[examples]`.

CHAPTER

THREE

AGENTS

CHAPTER

FOUR

MULTI AGENTS

Usage of "agents" in this case could be a bit misleading. Here are entitites or algorithms that understand how to organize internal agents to get better in interacting with the environment.

The distinction between these and many individual is that some interaction between agents is assumed. It isn't a single agent that tries to do something in the environment and could consider other agents as part of the environment. Typical cases for multi agents is when they need to achieve a common goal. Consider cooperative games like not letting a ball fall on the ground, or team sports where one team tries to capture a flag and the other tries to stop them.

4.1 MADDPG

4.2 IQL

CHAPTER

FIVE

BUFFERS

5.1 Basis

This class is the abstraction for all buffers. In short, each buffer should support adding new samples and sampling from the buffer. Additional classes are required for saving or resuming the whole state. All buffers internally store data in as Experience but on sampling these are converted into torch Tensors or numpy arrays.

class ai_traineree.types.experience.**Experience**(***kwargs*)

Basic data unit to hold information.

It typically represents a one whole cycle of observation - action - reward. Data type used to store experiences in experience buffers.

5.2 Replay Buffer

The most basic buffer. Supports uniform sampling.

5.3 Replay Experience Buffer (PER)

5.4 Rollout Buffer

CHAPTER

SIX

POLICIES

CHAPTER

SEVEN

NETWORKS

Networks are dividied depending on their context. For some reason it's often to find convetion of heads and bodies, and that's why we're keeping it here. If you haven't heard of these before think about the Frankeinstain monster. Body is not a whole body but rather a body part, e.g. arms and legs. Obviously(!), they don't work by themselves so you need a head which will control them. Some heads take body parts explicitly and build the whole monstrocity and some heads are predefined to closely match suggestion in a paper. So, in general, a head is more complex and does more than a body, but for some agents a single body part, e.g. Fully connected network, is good enough.

7.1 Bodies

7.2 Heads

CHAPTER

EIGHT

ENVIRONMENT RUNNERS

8.1 Single agent

8.2 Multi agent

CHAPTER

NINE

TASKS

In short, a Task is a bit more than environment. Task takes an environment, e.g. CartPole, as an input but it also handles state transformation and reward shaping. A Task also aims to be compatible with OpenAI Gym's API. Some environments aren't compatible and so we need to make them.

CHAPTER

TEN

DEVELOPMENT

10.1 Philosophy

- Agents are independent from environment. No interaction is forced.
- All agents should have the same concise APIs.
- Modular components but simplicity over forced modularity.

10.2 Concepts

10.2.1 State vs Observation vs Features

State is an objective information about the enviroment. It is from external entity's point of view. Access to states isn't guaranteed even if one has full control over the environment.

Observation is from agent's perspective. Its domain is defined by agent's senses and values depend on agent's state, e.g. position.

Features are context dependent but generally relate to some output of a transformation. We can transform observation to a different space, e.g. projecting camera RGB image into an embedding vector, or modify values, e.g. normalize tensor.

Example: Considering basketball game as an environment. A spectator is the one who might have access to the state information. In this case, a state would consist of all players' positions, ball possession, time and score. However, decide being able to see everything they wouldn't know whether any player is feeling bad or some places on the field have draft. An agent, in this situation, is a single player. Everything that they see and know is their observation. Although they might be able to deduce position of all players, it will often happen that some players will be behind others or they will be looking in a different direction. Similar to spectator they don't know about other players stamina levels but they know theirs which also has an impact on the play. Their physical state and internal thoughts are features.

Code-wise, state is the output of enviroment. Observation is what an agent can get and deals with on input. Feature is anything that goes through any transfomrations, e.g. bodies and heads. A specific case of a feature is an action.

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

E