# TEK5040 Compulsory Assignment No 3
# Generative Adversarial Imitation Learning

November 3, 2020

## 1 Introduction

This assignment deals with Generative Adversarial Imitation Learning (GAIL). The GAIL algorithm is applied to solve the problem of *inverted pendulum*. In order to simulate the problem, open-AI gym environment `Pendulum-v0` is used. If you want to learn more details, take a look at `https://gym.openai.com/envs/Pendulum-v0/`

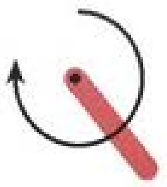

Figure 1: The problem of inverted pendulum

The goal of the problem is to swing the pendulum so that it stays upright. We have only one action parameter, namely the torque applied on the pendulum. Therefore the action is a scalar (i.e. The action vector has only one element). Note however that the action space is continuous. The system state is described by three parameters, $[\cos\theta, \sin\theta, \dot{\theta}]$ where $\theta$ is angle of the pendulum respect to a reference vertical line. We also assume that states can be observed directly, therefore the terms *observations* and *states* can be considered synonymous in this task.

## 1.1 Provided files

- `run_gail.py` implements the GAIL algorithm together with a reinforcement learning system based on proximal policy optimization (PPO) on an actor-critic architecture.
- `expert_traj.npy` is a numpy archive containing expert trajectories. Recall that an expert trajectory is a sequence of state-action (observation-action) pairs generated when a (human) expert controls the system. `expert_traj.npy` contains a 2-D numpy array of dimensions $N \times 4$, where each row consists of observations first 3 elements) and expert actions (last element).
- `common` is folder containing file(s) that implement the `Pendulum-v0` environment.

## 1.2 Running the script

The following python packages are required to run the script `run_gail.py`

- `tensorflow 2.x`
- `gym`
- `matplotlib`

Once the requirements are met (**and the second item in the task list is completed**), you can try to run the script through the command:

```
python3 run_gail.py
```

Then you can observe a dynamically updating plot which shows the variation of *reward* of a test run against the number of training iterations.

# 2 Implementation

An outline of the GAIL implementation for the `Pendulum-v0` problem in `run_gail.py` is as follows:

`Lines 18-34` creates the environment `Pendulum-v0`. Note that the environment is created with 16 workers. That means that there are 16 parallel copies and each interaction with the environment results in 16 responses.

`Lines 37-62` creates a Keras layer holding a normal distribution `ProbabilityDistribution`. This is used in sampling the actions

`Lines 65-125` creates a set of Keras models for an actor-critic architecture `ActorCritic` which contains value network, policy network, sampling operations and the surrogate loss for the PPO algorithm.

Lines 129-141 defines a function `compute_gae` which calculates the generalized advantages to be used the PPO algorithm.

Lines 145-200 implements the `PPO` algorithm. The purpose of this class is to update the parameters of the value network and policy network of the actor-critic architecture. The `update()` function in line 184 can be called to make a parameter update for a single batch of input and target data.

Lines 204-212 implements an iterative data feeder in mini-batches.

Lines 217-232 implements a routine for testing the system. It generates `observation-action` pairs and records the environment generated `rewards` in a test run.

Lines 236-246 loads the expert trajectories.

Lines 250-297 implements `Discriminator`. The job of Discriminator is to distinguish the expert trajectories from the policy generated trajectories. More specifically it should be trained to output high probabilities for expert trajectories and low probabilities for policy trajectories. In line 290, the update function, that updates discriminator parameters is partially implemented.

Lines 303-322 specifies hyper-parameters.

Lines 326-411 implements the training procedure for the whole GAIL algorithm. The psuedo code of the GAIL algorithm is listed in Listing 1.

For more details of the implementation, please refer to the comments in the script itself.

```
Initialize policy-network, value-network  and discriminator-network
for  N iterations do
    empty list of observations, actions, values and rewards.

    for N steps do
        generate action  for the current observation using the policy network.
        act upon the environment with the generated action and get the next observation.
        make observation-action pair and get the reward.
        generate value for the current observation using the value network.
        append observation, action, value and reward in the appropriate list.
        make next observation the current observation.
        compute (generalized) advantages using the rewards and values lists.
    done

    if iteration is a multiple of M
        for L epochs do
            Update the policy and value network parameters (PPO optimization).
        done
    endif
    update discriminator parameters (maximize expert trajectory probability,
                                     minimize policy trajectory probability).
```

```
done
```

Listing 1: GAIL psuedo code

## 3 Task

In the task you are going to focus on the implementations of the discriminator (lines 250-297) and GAIL algorithm (lines 326-411).

1. Write down the line numbers of the Python code in `run_gail.py` corresponding to each statement of psuedo code in Listing 1.
2. Complete the statement in line 297, by specifying target values. Describe your choice of target values with respect to the `compile` statement in lines 287/288 and original GAIL algorithm. HINT: Try to use quantities defined in lines 264 and 265.
3. If you are asked to modify the provided script and implement the Guided Cost Learning (GCL) algorithm, describe briefly how you would do that (You are NOT required to implement the GCL algorithm).
4. For this problem, it is possible to define a reward function $r = -(\theta^2 + 0.1\dot{\theta}^2 + 0.001a^2)$ where $a$ is the action(torque). The reward is miximized when the pendulum is in the upright position with a minimum (zero) effort (i.e. $\theta = 0, \dot{\theta} = 0, a = 0$). Why is the GAIL algorithm more useful for complex practical control problems in, for example, navigation and manipulation?

## 4 Submission

Submit only answers to above questions. You are NOT required to submit the code or other simulation results.