# CSC2626: Assignment 1
Due Sept 25 at 6pm ET
15 points

September 11, 2024

## 1 Introduction

The goal of this assignment is to help you get familiar with imitation learning and DAgger applied to the driving domain and to also ensure that you are able to solve supervised learning tasks with neural networks. In order to automate the process of expert demonstrations, without relying on human input, we have modified a car racing environment in OpenAI Gym to also provide expert demonstrations from a feedback controller that allows the car to do road following. Unlike the feedback controller that has access to the true state and coordinates of the track, your driving policy will only have access to an image and will output a steering command.

## 2 Setting Up

This assignment assumes that you are running some version of Ubuntu between 16.04 to 20.04. Newer versions of Ubuntu may work too, but have not been tested. If you are a Windows or Mac user and you are experiencing issues, please email the TAs early on to try to resolve them.

**Starter code** Unzip the `A1.zip` file to get the starter code for this assignemnt.

**Virtualenv** Set up a python virtualenv environment in your home directory, so that you can install python packages without requiring administrative rights in the computer you are using. Specifically, run `python3 -m venv myenv`, which will setup a Python3 environment under the directory `myenv`. Run `source ./myenv/bin/activate` in order to activate that virtual environment. Now you can begin installing python dependencies. Start with `pip3 install scikit-image`.

**Pytorch** Follow the instructions at `https://pytorch.org/get-started/locally/` to install Pytorch for CPU or GPU, depending on whether your system has access to a GPU. If you have a GPU you will also need to know which version of CUDA is your system running. You can do this by executing `nvcc --version`. If you don't have a GPU select `None` when choosing the CUDA version.

**OpenAI Gym** Run `pip3 install gym box2d-py pygame`. This assignment has been tested only with Gym version `0.26.2`.

**Initial training set** The assignment repository includes an initial dataset of images annotated with expert actions, which you can use for the first round of training a steering network. Run `cd A1 && unzip dataset.zip`.

**Run the expert** Try to run `python3 racer.py --expert_drives=True`. If you see a car driving in the middle of the track your installation works. The expert in this case is a feedback controller with only 4 parameters that have been tuned. However, the expert has access to extra information, such as the distance of the car from the middle of the road, while your learner will only have access to the image.

## 3 Supervised Learning (6 pts)

Fill in the code for the training procedure in `train_policy.py`. If you have a GPU make sure that in `utils.py` the device is set to `cuda`. Otherwise, leave it to `cpu`. To start the training procedure run `python3 train_policy.py --n_epochs=50 --batch_size=256 --weights_out_file=./weights/learner_0_supervised_learning.weights`

`--train_dir=./dataset/train/ --weighted_loss=False`

Also fill the network architecture in `driving_policy.py`. Start with an architecture that is similar, but not identical, to the NVIDIA end-to-end driving paper, even though it likely has too many parameters for fitting the dataset used in this assignment. For the features of the convolutional network start with this sequential model: `24 conv, relu, 36 conv, relu, 48 conv, relu, 64 conv, relu, flatten`, where each convolutional layer (called Conv2d in Pytorch) uses a kernel/window of size 4, stride of size 2, and 1 pixel padding. For the classifier start with this fully connected model: `fc 64, relu, fc n_classes` where `fc` stands for a fully-connected layer (called Linear layer in Pytorch). The output of the network will be the probability of `n_classes` possible steering directions, just like in the ALVINN paper. You are not however required to implement the gaussian-shaped discrete predictions that Pomerleau used to make consecutive classes nearby in terms of the angle they represent. You should optimize the categorical cross-entropy loss to learn a mapping from images to actions.

Now, try to execute the learner's policy on the simulator by running `python3 racer.py --learner_weights= ./weights/learner_0_supervised_learning.weights`. You should see that the initial policy eventually gets outside the race track before completing one full round of driving. You should also observe that the steering policy is wobbly on straight lines. You are not required to fix that for the purposes of this assignment.

## 4 Weighted Classification Loss (3 pts)

The majority of the expert demonstrations you will have in your dataset will be for driving straight ahead. Your training set will have a significant class imbalance, where the labels corresponding to driving straight ahead (e.g. classes 9,10,11 if `n_classes=20`) will occur much more frequently than sharp turns. One reasonable question to ask is whether taking this class imbalance into account when training the learner could help. To answer that question use a weighted loss that weighs errors in each class according to the inverse frequency of occurrence. In Pytorch the `cross_entropy` loss function allows you to specify such weights. Implement this minor change in `train_policy.py` and rerun supervised learning with the weighted loss: `python3 train_policy.py --n_epochs=50 --batch_size=256 --weights_out_file=./weights/learner_0_weighted_loss.weights --train_dir=./dataset/train/ --weighted_loss=True`

You will see that the vehicle still veers off track. Experiment by making sure that each mini-batch contains representation from both straight-line data and turning data. You can implement this change as an extra option in `dataset_loader.py`, which checks the steering command associated with each image.

## 5 DAgger (6 pts)

Implement DAgger in `dagger.py` and run `python3 dagger.py`, and run 10 DAgger iterations, which should be sufficient to make the car complete the track without interventions. This process should generate weights `learner_0.weights, ..., learner_10.weights` at the end of each DAgger iteration. Also, keep in mind that when you run this procedure the training set in `dataset/train` will be augmented with expert-labeled images from the execution of the learner's policy. Images from the $i^{th}$ DAgger run will be saved under the names `dataset/train/expert_i_t_cmd.jpg`, where `t` is the timestep of the image recorded during the $i^{th}$ run, and `cmd` is the corresponding steering command that the expert issued for that image in [-1,1]. You should observe that as the training set increases the resulting policy will eventually be able to remain within the track and fully traverse it. To show your work for this question, create a plot showing the number of DAgger iterations on the x axis vs. the cumulative cross-track error for the latest iteration on the y-axis. There is a function that computes the instantaneous cross-track error in the starter code in the environment class in `full_state_car_racing_env.py`. You may need to change the `racer.py` code to keep track of the cumulative cross-track error. Save this plot in a file called `dagger_iterations.png|pdf|jpg`.

## 6 What/How to submit

Submit a file called `assignment_firstname_lastname_studentid.zip` that contains the starter code and your changes. This zip file should also include the file `dagger_iterations.png|pdf|jpg`. It should **not** include the checkpointed weights of the policy. Submissions should be done on Quercus. You do not need to include a writeup of your solutions.