Deep Learning Lab Course
Imitation Learning and Reinforcement Learning

Nick Heppert, Julia Hindel, Jan Ole von Hartz, Baohe Zhang

Due: May 2nd, 2024

The goal of this exercise is to experiment with imitation learning and reinforcement learning.

In the first part you will implement a behavioral cloning agent and evaluate it's performance on the `CarRacing` control task from the OpenAI Gym benchmark suite. Therefore, you have to collect data by driving on the track as an expert.

In the second part, you will implement a DQN agent and use ϵ-greedy exploration, experience replay and target networks for on-policy training. You can read more about Deep Q-Networks in *Playing Atari with Deep Reinforcement Learning*.

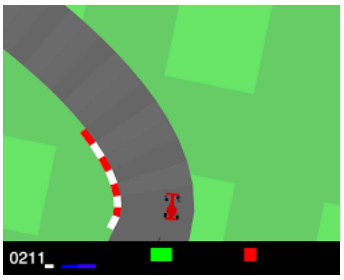

Figure 1: The `CarRacing` environment

Please submit

- a PDF report consisting of
  - a written text of maximum 2 pages
  - all images and figures in the appendix, linked to from the main text
- your models (or a small video)
- the source code

to ILIAS. Please format the source code using `black`.

Getting Set Up

Before starting with the exercise please review the following information carefully.

0.1 Installing python using miniconda

You should install python inside a *conda* environment since this way the installation will be independent of the python installed on the system. *As an alternative to conda, you could also use virtualenv.*

Below are the installation instructions for installing conda on Ubuntu. On other operating systems the workflow is similar.

- Download the latest Miniconda3 Linux 64-bit installer: `https://docs.conda.io/en/latest/miniconda.html` Usually the 64-bit version is correct. Use `wget [link]` to download a file in the terminal.
- Open the terminal and `cd` into the folder containing *Miniconda3-latest-Linux-x86_64.sh.*
- Run `bash Miniconda3-latest-Linux-x86_64.sh` to install it into your home directory. When asked by the installer to initialize Miniconda3, answer yes. This way, your `~/.bashrc` file will be modified so you have conda available at startup.
- Close and reopen your terminal window to make sure the changes take effect.
- Test your installation by running `conda --version` to see the its version.

0.2 Setup conda environment

Next we will set up conda with a new environment named `lab` which uses Python 3.8. Run the following commands in your terminal (macOS, Linux) or miniconda prompt (Windows):

```
conda --version
conda update -n base -c defaults conda -y
conda create --name lab python=3.8 -y
conda activate lab
conda env list
```

The last command should show you all installed environments (including `base` and the activated `lab` environment). This way, you have a separate environment to install everything for this course and avoid possible interference with/from other python projects.

Note: Whenever you start the miniconda prompt/terminal for this course, run `conda activate lab` to switch to the correct environment.

0.3 Pool computers

If you want to train on the computers in the PC pool, make sure you applied for an increase in disk space quota. Otherwise, the space in your home folder will not be enough.

How to use the pool

- Connect to the login server of the technical faculty: `ssh username@login.informatik.uni-freiburg.de`
- Do not run anything on the main server. Connect to one of the pool computers there instead. Preferably use `tfpool25-46` or `tfpool51-63` as they provide the best GPUs. `ssh tfpoolXX` - replace XX with a 2 digit number of the computer you want to connect. Before running make sure no one else is using the selected computer and the GPU is free.
- See GPU load: `nvidia-smi`
- See logged in accounts: `who`
- View running processes: `top`

How to use `screen`

Start a `screen` session to keep your code running in the background:

- Start the session

  ```
  screen -S t01
  # run your experiment
  ```

- Detach from screen using: ctrl+a+d
- Login back into screen:

  ```
  screen -ls
  screen -r t01
  ```

- Write down on which computer you started your screen session.
- ctrl+esc to enter copy mode if you need to scroll up, navigate with arrow/page keys, copy output etc

0.4 Additional resources

If you have difficulties understanding PyTorch, we suggest to check out some of the tutorials for PyTorch, learn the basics is a good tutorial to get started.

A very important concept is indexing of tensors, see numpy indexing. The indexing in PyTorch works very similar to numpy.

We recommend using an IDE like VSCode or PyCharm for writing and debugging python code.

0.5 Using browser-based tools

Throughout the exercises, you will **optionally** use browser-based tools like jupyter notebook or tensorboard. The easy way to do this is to run the server and browser on the same machine (either directly sit in the pool or setup everything on your home computer).

However, if your code is running on a *different computer as the browser* , i.e. you are running the code on a pool machine but running the browser on your home computer) you will need to:

- start the respective server on the pool machine.
- establish a SSH tunnel from your home computer to the tf login node.
- establish a SSH tunnel from the tf login node to the pool machine where the server is running.

The tunneling command looks like this:

```
ssh -v -N -L ${port}:localhost:${port} "${user}@${targethost}"
```

You might need to change the port of your server application if the port is already in use by another person. Note that everyone who can access the login node will be potentially able to see your server. Jupyter notebook protects this with an access token by default, but tensorboard does not.

0.6 Exercise 1 Specific Setup

The code can be found in ILIAS. All necessary dependencies should be listed in the `requirements.txt` inside the repository. You can install them at once using the command `pip install -r requirements.txt` in the root directory (make sure your virtual environment is active). Make sure to follow the given `README.md` for setting up.
Install Tensorboard with `pip install tensorboard`. In the command-line, execute `tensorboard --logdir=path/to/log-directory --port=6006` and watch the progress of your training in your web browser under `localhost:6006`.

1 Imitation Learning

1. Get familiar with the CarRacing environment by running `drive_manually.py` and get used to the car control (keys up/down/left/right). Note that this does not work via ssh, so you have to sit in front of a pool computer. Please press only one key at a time.

2. Collect driving data with `drive_manually.py --collect_data`. The data will be stored every 5000 steps in `./data` . It will take a while to collect all samples, but the more you collect the better you can train your agent[1]. The script will create the folders `./data` and `./results` with your expert score stored in a json file. You will need these results for your report and for the competition (see below).

   Have a look into the source code of `drive_manually.py` and get familiar to the interface to gym.

[1]collect at least 10000 samples

3. Use the collected data to train a behavioral cloning agent: Implement a convolutional neural network in `agent/networks.py`, implement the agent in `agent/bc_agent.py` and train it using `train.py`. Use Tensorboard to monitor progress and create plots of the training progress for the report. Evaluate the performance of your agent in `test.py` over 15 episodes. Check the code for further instructions/hints.

4. Improve your agent:

   - Check the distribution of actions in the dataset. You will see that it is imbalanced because most of the time you drive straight. This can be a problem for the training. Data augmentation methods such as sampling the actions uniformly or other oversampling methods can help here.
   - In your first experiments you used only the current image as input feature for your network. Append the history of the last N images to your input. Check the performance for different values for N, e.g. $N \in \{1, 3, 5\}$.

5. The report should include:

   - Your hyperparameter setting(s).
   - A figure showing the learning curves of training and validation performances. Don't use screenshots. You can read the tensorboard event files to create nice plots with `matplotlib` or `seaborn`. Make sure that your figure includes title, axis labels and a legend.
   - A result table showing the mean performance and standard deviation over 15 episodes for your best-performing agent and different history lengths.
   - Additionally, submit the models or a small video of your agent.

Hint: Very good agents can achieve a mean score of 800-900.

2 Reinforcement Learning: Deep Q-Networks

Hint: Please start early with this exercise. Q-learning can take a long time for the CarRacing environment. You can run the code either on GPU or CPU. A GPU machine will be faster when you work with images and CNNs, but you should be able to get good results in around 6-8 hours of compute on a CPU. Additionally, you cannot start the training via ssh on the pool computers because of rendering and x-forwarding issues. So if you want to work outside the pool, you either have to use your machine or modify the script to avoid loading box2d.

2.1 CartPole

To keep the problem simple, start with CartPole, a classic control task from gym. Implement DQN by extending the starter code in our repository. We provided you some components of Q-Learning. Use `train_cartpole.py` for the training and finish the DQN agent in `agent/dqn_agent.py`. For CartPole, you will find a network in `agent/networks.py` and code for the replay buffer `agent/replay_buffer.py`

You can watch the progress of the total episode reward (achieved with ϵ-greedy exploration) during the training in Tensorboard. Additionally, evaluate the episode reward every 20 episodes with greedy actions only (compute the mean episode reward of the agent with deterministic actions over 5 episodes).

2.2 CarRacing

If DQN works for CartPole, you can start working on the CarRacing environment. Use your CNN architecture for the Q-network and target network. You'll find starter code for the training in `train_carracing.py`. Add a maximum capacity (with first-in-first-out) to the replay buffer to avoid memory errors if you collect a lot of data.

Getting good results with DQN for the CarRacing environment is harder than for CartPole. Only changing the network architecture to CNNs probably won't lead to good results for the RacingCar. However, start with exactly this and watch the episode reward during the training in tensorboard for some time.

You may want to use some of the following tricks to make it work:

1. Exploration: If you turn on the rendering during training, you will see that sampling actions from a uniform distribution doesn't let the agent explore the environment properly. Use higher probabilities for accelerating or going straight.
2. Frame Skipping: skipping n frames and repeating the RL agents action during this time can help (already implemented, you only need to change `skip_frames=0` in `train_carracing.py`).
3. To speed up training, train on shorter episodes in the beginning (by adapting `max_timesteps`).

The report should include:

- Your hyperparameter setting(s).
- A figure showing the learning curves of training and validation performances. Don't use screenshots. You can read the tensorboard event files to create nice plots with `matplotlib` or `seaborn`. Make sure that your figure includes title, axis labels and a legend.
- Mean performance and standard deviation over 15 episodes for your best-performing agent.
- Additionally, submit the models or a small video of your agent.

3 Competition

(Optional/Bonus) Add your driving score and the score of your agents in our google doc to take part in a small competition. Choose a name and enter your final scores HERE. The winner will receive a prize!