Deep Learning Lab Course
Imitation Learning and Reinforcement Learning

Iman Nematollahi, Adrian Röfer

Due: May 10, 2022

The goal of this exercise is to experiment with imitation learning and reinforcement learning. 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. Finally, 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*. Please submit a report (max 2 pages), your models (or a small video) and the source code in Ilias.

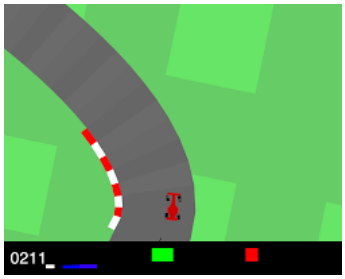

Figure 1: CarRacing

1 Getting Set Up

The code can be found at https://nrgit.informatik.uni-freiburg.de/exercises/dl-lab-2022-rl. All necessary dependencies should be listed in the requirements.txt inside the repository. Please install all dependencies inside a *virtualenv* or *conda* environment. 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).
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`.

2 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.

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:

[1] collect at least 10000 samples

- 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.

3 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. You either have to use your machine or modify the script to avoid loading box2d. As the pool remains closed due to the Covid-19 situation, we ask you to use your local computer to record the data and evaluate the model (because you need the rendering). Therefore, we recommend also training on your local computer, as it should only take around 2-3 hours more in comparison to training on the pool computers.

3.1 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).

3.2 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.

4 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!