Robot Learning
Weekly Exercise 6

Marc Toussaint & Wolfgang Hönig
Learning & Intelligent Systems Lab, Intelligent Multi-Robot Coordination Lab, TU Berlin
Marchstr. 23, 10587 Berlin, Germany

Summer 2024

1 Literature: Privileged and Sensorimotor Policy Training

Here is a prominent application paper for RL:

J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986, 2020-10-21. URL: https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/448343/1/2020_science_robotics_lee_locomotion.pdf

It uses standard RL in simulation to train a *privileged policy* (which they call "teacher policy") which has full access to the simulation's state information (e.g. exact terrain profile). In a second stage they train a *sensorimotor policy* (which they call "student policy") to imitate the privileged policy, but with sensorimotor (partially observable) input only. As the teacher policy can be queried anywhere, they can use DAgger for imitation, which simplifies imitation learning a lot.

[The general idea of training a sensor-based (=partial input) policy from a privileged (=full information) policy is old, previously called input remapping, or just surrogate model.]

Fig. 4 gives an overview of the approach. Here the questions:

a) The input to the previleged policy is full information (exact robot & simulation state). But what is the definition of the output action $\bar{a}_t$? Looking for an answer you'll find words like "leg frequencies" and "foot position residuals" – what are these?

page 8: "The policy outputs f_i's and target foot position residuals ($\Delta r_{f_i,T}$)". f_i's are modulations of the base frequency (1.25Hz) (one number), and $\Delta r \in \mathbb{R}^3$ are foot position offsets, relative to a fixed (expert programmed) reference motion $F(\phi)$. The policy outputs this for each leg, I think. (So, $\bar{a}_t$ should be 16-dim).

b) The Supplement S4 (pdf page 16) explains the reward function – a great example for reward engineering (in the positive sense, as this reflects the authors' understanding of "good locomotion"). Be able to explain each term and how they relate to higher level "commands".

A linear combination of 7 terms.... A "bell-shaped" reward (of the form $\exp(-sx^2)$) is used in the context of following (operator-commanded) reference velocities. Foot Clearance and Collisions are (surprisingly) low weighted soft penalties. Smoothness a standard acceleration penalty, and torque penalties standard control costs (closely related to acceleration penalties).

c) Eq.(1) includes a second loss term, comparing $\bar{l}_t(o_t, x_t)$ with $l_t(H)$. Explain what $l_t(H)$ is and the idea of this term.

My favorite detail of the paper: The privileged policy is force (its architecture designed) to compress the privileged information into a latent code $\bar{l}_t$. The sensor-based policy does not have access to this information, but is trained to predict that same $\bar{l}_t$ from a history H of observations. This predicted l_t is then fed into another MLP (together with observable information o_t) to predict the action a_t. This pipeline of two networks is trained to minimize the prediction error on both, l_t and a_t.

2 Episodes & Terminal States

Standard MDP theory assumes an infinite process $s_0, a_0, r_0, s_1, a_1, r_1, ...$ of states, actions and rewards. Accordingly, the return is defined as the infinite sum $\sum_{t=0}^{\infty} \gamma^t r_t$.

However, practical problems in the literature often involve "terminal states", and one speaks of "episodes". The following exercise clarifies how "terminal states" and "episodes" are meant in an infinite MDP.

a) We define a terminal state as follows: Assume that in step T the agent reaches a terminal state s_T. The agent can then make a very last action a_T, and gets a final reward $r_T = R(s_T, a_T)$, but after this "there are no more states, actions, or rewards", and the total return of the agent is $\sum_{t=0}^{T} \gamma^t r_t$.

At first sight this is inconsistent to how MDPs are defined, because by definition they do not terminate. How can you construct a formal MDP to model such terminal states? (Tip: Extend the state space.)

We can add a 'Nirvana' state to the MDP; informally, the nirvana state s_N has the properties

- the terminal state transitions to the nirvana state for any action.
- one cannot escape from the nirvana state
- there is no reward in the nirvana state

The reward then becomes $\sum_{t=0}^{\infty} \gamma^t r_t = \sum_{t=0}^{T} \gamma^t r_t + \sum_{t=T+1}^{\infty} \gamma^t \cdot 0 = \sum_{t=0}^{T} \gamma^t r_t$

b) Consider an MDP where the goal state is a *tunnel state*, which means that every choice of action in the goal state leads to receiving the goal reward and transitioning to a (maybe random) initial state $s \sim P(s_0)$.

Is the optimal policy for the MDP with tunnel goal state the same as the optimal policy for an MDP where the goal state is a terminal state? Provide arguments (ideally a rough proof or counterexample) for your answer.

The answer is no: One can construct counterexamples.

First recall the geometric series $\sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}$

A counter example is a MDP where at the start we have two choice, and none afterwards. All transitions are deterministic. The first choice leads to 1 transition, then reward a, then termination; the second choice leads to 5 transitions without reward, then reward b, then termination.

If the reward states are terminal states, the first choice has return γa, the second choice has return $\gamma^5 b$. We can choose b large so that the second choice is better.

If the reward states are tunnal states, the first choice has return $\gamma a \frac{1}{1-\gamma^2}$, and the second choice has return $\gamma^5 b \frac{1}{1-\gamma^6}$. The latter decreases drastically with γ; and for same a, b, γ the first choice is better.

c) In practice one never runs (or simulates) a process infinitely long. Instead, one typically aborts/truncates at some finite horizon T. One such truncated run is called *episode*. One then typically repeats many episodes (to collect data for learning or estimation of values/performance). When an episode was *truncated*, discuss how one could actually estimate the expected return of the policy?

Truncation is trickly. (That's why gym environments explicitly report whether an episolde was trunctated instead of reached a terminal state.) When an episode was truncated then the policy "could have" collected more reward; e.g., if the last state was rewarded with r and the policy could just rest there, one should add $r\frac{1}{1-\gamma}$ to the total return to be fair. However, that depends very much on the problem. In practise, people usually ignore the information that the episode was truncated instead of terminated, and only report on the truncated return. Strictly speaking this makes the MDP non-stationary, because absolute time influences the expected total return, not just state.

3 Lunar Lander Domain Randomization

This is a coding exercise. Please bring your laptop and connect to the HDMI in the tutorial to show your results. (If you upload a pdf, just include a screenshot of results in the pdf.)
Install the lunar lander simulation of *gymnasium* (https://gymnasium.farama.org/) using

```
pip install "gymnasium[box2d]"
```

Similar to before, one can create an instance of the lunar lander (with varying wind enabled) using

```
env = gym.make('LunarLanderContinuous-v2', enable_wind=True)
```

a) Train a policy – you should be able to reach rewards of > 200. To avoid finding new hyperparameters, use TD3 rather than SAC for training, where the default settings (with MlpPolicy and action noise) should work well.
Hint: The action noise can be defined as follows:

```
from stable_baselines3.common.noise import NormalActionNoise
action_noise = NormalActionNoise(mean=np.zeros(n_actions), sigma=0.1 * np.ones(n_actions))
```

b) Validate your policy in environments with different wind magnitudes and gravities. You can adjust these settings when making a gym environment, e.g.,

```
env = gym.make('LunarLanderContinuous-v2', enable_wind=True, gravity=-10, wind_power=5)
```

For gravity, use values between -11 and -1; for the wind magnitude use values between 0 and 20.

In which settings does your policy work well and in which does it not?

c) Train a policy with domain randomization on both gravity and wind_power. How does this policy compare to the other policy when validating in different settings as in b)?

Hint: You can use the callback mechanism of the policy (for `_on_rollout_end`) to add the randomization at the end of each episode. To do this, you can directly modify the parameters of the environment, e.g., set `env.gravity = np.random.uniform(min_value, max_value)`.

References

[1] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986, 2020-10-21. URL: https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/448343/1/2020_science_robotics_lee_locomotion.pdf.