Tópicos especiais em Inteligência Computacional II:
Aprendizado por Reforço
Exercício 5: Indirect reinforcement learning

Wouter Caarls
wouter@ele.puc-rio.br

May 21, 2020

Indirect reinforcement learning uses a learned transition model to generalize experience in the space of the system dynamics. The goal of this exercise is to investigate its effectiveness for the pendulum swing-up problem, specificaly by implementing Dyna for the state-value actor critic.

The prerequisites are the same as for the previous exercise, so should already be installed. If not, open the Anaconda Prompt and run

```
(base) C:\Users\You> pip install tensorflow gym
```

1 Understanding the code

The Python code for this exercise (`ex5.py`) contains one important function and three important classes, see Table 1.

Table 1: Main functions and classes

`ex5.rbfprojector`	Gaussian Radial Basis Function projector.
`ex5.Model`	Dynamics model approximator.
`ex5.Memory`	Replay memory.
`ex5.Pendulum`	Pendulum swing-up environment.

Exercise 1.1 The `Model` class has a hidden parameter `diff`. Read `ex5.py` and try to figure out what it does. Why could this be advantageous?

2 Exercise

Exercise 2.1 Adapt the State-Value Actor Critic algorithm from Exercise 4 to keep track of all experienced transitions (use `Memory`), and use these at the end of every episode to train a neural network on the system dynamics (use `Model.fit`).

Exercise 2.2 After every experienced transition, estimate its reward and next state using the model, and track the the root mean squared error (RMSE) of both for each episode. Plot them to make an estimate of when the model is good enough to start using to generate simulated transitions. Also plot the model dynamics using `Pendulum.plotnetwork`, and interpret the graphs.

Exercise 2.3 After every real episode, run N simulated episodes of 20 transitions. Start these episodes using `env.reset()`, but use the model to estimate the next state. Terminate the episode early if any of the state variables exceeds its range (`env.env.observation_space.low` - `env.env.observation_space.high`). Use the same learning rules as in the real episode.

Call `env.normalize` on the estimated next state to make sure it is on the unit circle. Start generating simulated transitions only when the model has settled (see the previous question), and only retrain the model every few real episodes if it takes too long.

Exercise 2.4 Investigate the sample complexity vs computational complexity trade-off by varying the amount of simulated episodes per episode N. How does the rise time change with increasing N? Is the end performance affected?

Note that you may have to run each experiment a number of times to get statistically significant results.

Exercise 2.5 Compare the performance of your model-based state-value actor-critic to DDPG from the previous exercise. Discuss how they are related. What is the advantage of using a model over a replay memory? And disadvantage? Why can SVAC not use experience replay?