Reinforcement Learning Lab

Lesson 10: Actor-Critic (A2C)

Davide Corsi and Alberto Castellini

University of Verona
email: davide.corsi@univr.it

Academic Year 2022-23

UNIVERSITÀ
di VERONA
Dipartimento
di INFORMATICA

Environment Setup

The first step for the setup of the laboratory environment is to update the repository and load the miniconda environment.

- Update the repository of the lab:

```
cd RL-Lab
git stash
git pull
git stash pop
```

- Activate the *miniconda* environment:

```
conda activate rl-lab
```

- Install *Gymnasium* environments:

```
pip install gymnasium
```

Today Assignment

In today's lesson, we will implement the A2C algorithm to solve the CartPole problem. In particular, the file to complete is:

```
RL–Lab/lessons/lesson_10_code.py
```

Inside the file, a python function should be implemented. The objective of this lesson is to complete it.

- **def A2C()**

Notice that the other functions in the file (e.g., *training_loop*, *createDNN*, and *mse* are the same as the previous lessons). Expected results can be found in:

```
RL–Lab/results/lesson_10_results.txt
```

A2C (theory)

In A2C, the update rule consists of replacing the trajectory reward (i.e., as in REINFORCE) with the advantage:

$$J_\pi = \sum_{t=0}^{T} (log \pi_\theta(a_t|s_t) \cdot A(s_t, a_t)) \tag{1}$$

In general, obtaining $A(s_t, a_t)$ can be hard and requires two additional neural networks (i.e., $A(s_t, a_t)$ and $V(s_t)$). However, the advantage function can be simplified as follows:

$$A(s_t, a_t) = r_t + \gamma \cdot V(s_{t+1}) - V(s_t) \tag{2}$$

Exploiting this formulation, we can use only one additional network to estimate $V(s_t)$; this neural network is called critic.

Shaping

These operations require different reshaping: remember to always print the shape of the tensors and arrays before and after all the algebraic operations.

A2C (code snippets)

Following the update function for the actor:

```
# The update rule for the actor directly implements Eq. 1, using the
# simplified version for the advantage (Eq. 2)
adv_a = rewards + gamma * critic_net(next_states).numpy()).reshape(-1)
adv_b = critic_net( states ).numpy().reshape(-1)
objective = log_probs * adv_a - adv_b
```

Following the update function for the critic:

```
# Compute the MSE between the current prediction and the real advantage (Eq. 2)
target = rewards + (1 - dones.astype(int)) * gamma * critic_net( next_states )
prediction = critic_net( states )
mse = tf.math.square(prediction - target.numpy())
```

The critic can be updated multiple times to improve efficiency but requires to shuffle the buffer to avoid overfitting the data:

```
for _ in range(10):
    np.random.shuffle( memory_buffer )
```

Expected Results

Why A2C?

Apparently, A2C can perform even worst than a standard REINFORCE. However, there are many advantages to using an actor-critic architecture.

- You can train the critic off-policy (improve data efficiency).
- The update function for the policy does not require to be related only to one trajectory.
- The *advantage* is not related to the reward of the trajectory.

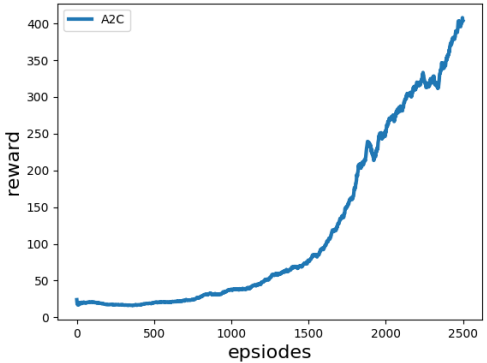

Figure: Note that obtaining this result requires time. You can stop the training after fewer iterations if you observe a growth in the reward.