Reinforcement Learning
Exercise 5

November 5, 2019

In this exercise we will implement a policy gradient algorithm for the Cartpole environment and compare it to the actor-critic approach.

1 Policy Gradient

In this part of the exercise you will be implementing policy gradient for the Cartpole environment with a **continuous action space**.

1.1 Environment with continuous action space

The environment used for this exercise is a modified version of CartPole. The standard CartPole uses a discrete action (applying force of either -10 or 10), while this version works with arbitrary force values.

Task 1 — 15 points Implement policy gradient for the Cartpole environment with continuous action space. Use `agent.py` for implementing the reinforcement learning itself (for example, the agent and policy classes). Instantiate those classes in `cartpole.py`, similarly to how it was done in Exercise 1.

Use constant variance $\sigma^2 = 5$ throughout the training.

(a) basic REINFORCE without baseline **(10 points)**,

(b) REINFORCE with a constant baseline $b = 20$ **(2.5 points)**,

(c) REINFORCE with discounted rewards normalized to zero mean and unit variance **(2.5 points)**,

Hint: The `agent.py` file from the first exercise session contains some useful code.

Hint: Your policy should output a probability distribution over actions. A good (and easy) choice would be to use a normal distribution (`from torch.distributions import Normal`). Log-probabilities can be calculated using the `log_prob` function of the distribution.

Question 1 — 10 points How does the baseline affect the training? **Why?**

Task 2 — 10 points Implement two cases of adjusting variance during training: exponentially decaying variance $\sigma^2 = \sigma_0^2 \cdot e^{-c \cdot k}$ (where $c = 5 \cdot 10^{-4}$ and k is the number of episode) and variance learned as a parameter of the network (in both cases set the initial value σ_0^2 to 10).

Hint: Use `torch.nn.Parameter(some_tensor)` to make your learned variance automatically updated by the optimizer.

Question 2 Compare using a constant variance, as in Task 1, to using exponentially decaying variance and to learning variance during training. Explain your answers to each sub-question.
Question 2.1 — 5 points What are the strong and weak sides of each of those approaches?
Question 2.2 — 5 points In case of learned variance, what's the impact of initialization on the training performance?
Question 2.3 — 5 points Which takes longer to train?

Question 3 — 10 points Could policy gradient methods be used with experience replay? **Why/why not? Explain your answer.**

Question 4 — 5 points Can policy gradient methods be used with discrete action spaces? Why/why not? Which steps of the algorithm would be problematic to perform, if any?

2 Actor critic

Task 3 — 15 points Revisit the policy gradient solution for the continuous Cartpole from Task 2 with learned sigma and implement the actor-critic algorithm. In the initial setup, perform updates at the end of each episode. Put your implementation in a separate files called `ac_cartpole.py` and `ac_agent.py`.

Hint: Check out the PyTorch tutorial to see how to calculate the $A_\theta \nabla_\theta \log \pi_\theta(a_i|s_i)$ term with the `detach()` function.

Task 4 — 10 points Update the actor-critic code to perform TD(0) updates every 10 timesteps, **instead of** updating your network at the end of each episode. Make sure to handle the end of each episode correctly (as in the previous exercises, the value of the terminal state is 0).

Question 5 — 10 points What are the advantages of policy gradient methods compared to action-value methods such as Q-learning? When would you use Q-learning? When would you use a policy gradient method?

3 Submission

Your report should be submitted as a **PDF file** as RL_Exercise3_*LastName_FirstName*.pdf. The report must include:

1. **Answers to all questions** posed in the text.
2. The **training performance plots** for each of the tasks (Task 1 a, b and c, Task 2 - for sigmas, Tasks 3 and 4 - actor critic).

In addition to the report, you must submit the Python code used to solve the exercises (as separate files, in the same folder as the report).

Please, avoid uploading unnecessary files such as self-generated folders (e.g. __pycache__, __MACOSX).

Deadline to submit your answers is **11th of November 2019, 11:55 am** (in the morning before the Monday exercise session).

If you need help solving the tasks, you are welcome to visit the exercise sessions on Monday, Tuesday and Wednesday.

Good luck!