Jonathon Luiten <luiten@vision.rwth-aachen.de> Due before: 09.05.2019, 11.59pm

Exercise 2: Reinforcement Learning

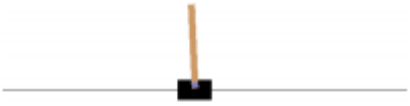

Figure 1: Rendering of the OpenAI gym CartPole-v0 environment.

Question 1: Reinforcement Learning ..... $(\Sigma = 12)$

In this exercise, you will implement policy gradient and temporal difference learning algorithms for the task of cart-pole balancing.

(a) Install OpenAI gym using pip in your tensorflow virtual environment:

```
pip install gym
```

Verify that the package has been installed by executing the script `cartpole.py` provided with this exercise. A window should open in which you see a simulated cart pole being controlled with a random policy (see Fig. 1). On the shell, the return obtained in each episode should be displayed.

The simulated cart pole is the CartPole-v0 environment of OpenAI gym [1]. The OpenAI documentation provides the following description of the environment: "A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center. CartPole-v0 defines "solving" as getting average reward of 195.0 over 100 consecutive trials."

The simulation provides a 4-dimensional continuous state of the cart (vertical angle of the pole, horizontal linear position of the cart, corresponding angular and linear velocities). Additionally, each episode should end after 200 time steps.

(b) Start from the script `cartpole.py` provided with this exercise to implement the REINFORCE algorithm from the lecture. Use tensorflow to define a computational graph for the optimization problem and use a gradient solver implemented in TensorFlow such as ADAM to solve the cart-pole balancing task. Define a parametrized policy as a linear mapping from states to the log probability of taking each action, i.e. $\pi_\theta(a_t = +1 \mid s_t) = \sigma(ws_t + b)$, $\pi_\theta(a_t = -1 \mid s_t) = 1 - \pi_\theta(a_t = +1 \mid s_t)$, $\theta = \left(w^\top, b\right)^\top$. Visualize the training progress using matplotlib by displaying the return in each training episode and smoothed versions of the return. How many training episodes does it take to solve CartPole-v0 using REINFORCE? How does the algorithm perform, if only angular and linear position are used to describe the state? Hint: since you need terminal episodes for REINFORCE, do not forget to stop each episode after 200 time steps. This is also important for the correct assessment of "solving" the cart-pole balancing task.

(c) Start from the script `cartpole.py` provided with this exercise to implement the Q-learning algorithm from the lecture. For approximating the Q function for the continuous state space, discretize the state space into a set of 10 bins for each dimension and estimate the Q function for each bin. Implement the ϵ-greedy as well as the softmax exploration strategy. Visualize the return over training episodes. How many training episodes does it take to solve CartPole-v0 with Q-learning? How does the algorithm perform, if only angular and linear position are used to describe the state? Hint: since this part does not require gradient descent, it can be implemented directly with numpy instead of TensorFlow.

[1] see https://gym.openai.com/envs/CartPole-v0

EXTRA FOR EXPERTS:
This is much more difficult and to get it to work really well probably requires 3 days of training using a powerful GPU.

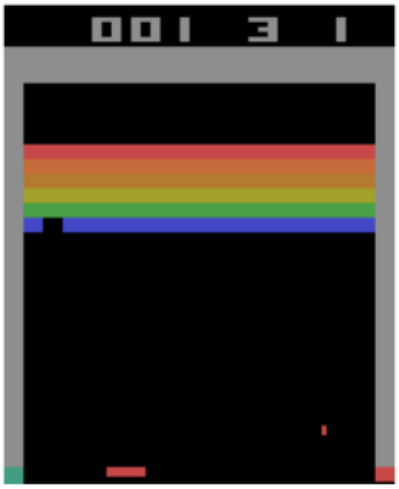

Figure 1: Rendering of the OpenAI gym Breakout-v0 environment.

Question 2: Deep Reinforcement Learning ..... $(\Sigma = 12 + 6)$

In this exercise, you will implement deep reinforcement learning algorithms for learning to play Atari Breakout.

(a) Install OpenAI gym version 0.7.0 and tqdm using pip in your tensorflow virtual environment: **(12 pts)**

```
pip install gym==0.7.0
pip install tqdm
```

Verify that the packages have been installed by executing the script `main.py` provided with this exercise. A window should open for the Atari Breakout game in which you see the bar being controlled with a random policy (see Fig. 1). On the shell, the average return obtained in each episode should be displayed.

The emulated Breakout game is the Breakout-v0 environment of OpenAI gym [1]. The OpenAI documentation provides the following description of the environment: "Maximize your score in the Atari 2600 game Breakout. In this environment, the observation is an RGB image of the screen, which is an array of shape $(210, 160, 3)$ Each action is repeatedly performed for a duration of k frames, where k is uniformly sampled from $\{2, 3, 4\}$. Breakout-v0 is an unsolved environment, which means it does not have a specified reward threshold at which it's considered solved."

Start from the code framework provided with this exercise to implement the DQN algorithm as described in the paper Mnih et al., Human-level control through deep reinforcement learning, Nature vol. 518, pp. 529–533, 2015. Use tensorflow to implement the deep Q-Network and its optimization.

- Visualize the training progress using tensorboard by displaying the return in each training episode and smoothed versions of the return.
- Test the learned controller at every 100000th training time step and average the return over 1000 test episodes in each of the steps.
- What is the maximum average return for the test episodes that you can achieve with up to 20000000 training time steps?
- How much real time does the training take for 20000000 algorithm time steps?

Hint: disable the visualization for efficient training.

(b) Implement the Asynchronous Advantage Actor-Critic (A3C) algorithm within the provided code framework. Use only a single actor-learner (no parallel threads). **(6 bonus)**

- Visualize the training progress using tensorboard by displaying the return in each training episode and smoothed versions of the return.
- Test the learned controller at every 100000th training time step and average the return over 1000 test episodes in each of the steps.
- What is the maximum average return for the test episodes that you can achieve with up to 20000000 training time steps?
- How much real time does the training take for 20000000 algorithm time steps?
- How does the method compare to DQN in learning performance (achieved maximum average return and training time)?

[1] see https://gym.openai.com/envs/Breakout-v0