LAB 2 - Multi-agent Reinforcement Learning

Multi-Agent Systems (TDDE13), Linköping University

November 16, 2023

1 Directions

Solve the exercises below. You should submit your source code (i.e. **frozen_lake.py** and **simple_hockey.py**) and a short report with your results to your TA (Emil Wiman, emil.wiman@liu.se) before the deadline. Please include the course code in the email subject line.

Deadline: 23:59, 15 December 2023

2 Preparations

To set up a suitable environment for your experiments, follow the steps below.

- In your working directory, clone the following repositories:
  git clone https://github.com/johan-kallstrom/frozen-lake-experiment.git
  git clone https://github.com/openai/maddpg.git
  git clone https://github.com/johan-kallstrom/multiagent-particle-envs.git
- In your working directory, create a directory for learning curves :
  mkdir learning_curves
- Add Python 3.7:
  module add prog/python/3.7.11
- In your working directory, create a Python virtual environment according to these instructions (use Python 3.7: **python3.7 -m venv my_env**):
  https://packaging.python.org/guides/installing-using-pip-and-virtual-environments/
  Alternatively (e.g., if installing on your own computer) you could use Anaconda:
  https://www.anaconda.com/
- Activate your virtual environment (**source my_env/bin/activate**) and install the following packages:
  pip install wheel
  pip install gym==0.10.5
  pip install matplotlib==3.3.3
  pip install numpy==1.21.6

pip install protobuf==3.20.0
pip install pyglet==1.5.21
pip install scipy
pip install tensorflow==1.13.1
pip install -e multiagent-particle-envs
pip install -e maddpg

3 Exercises

In this lab you will study aspects of reinforcement learning in two simple environments. The goal of the lab is to give you a basic understanding of reinforcement learning and related challenges, as well as some hands-on experience in training agents.

3.1 Ex 1: Trade-off Between Exploration and Exploitation

In this experiment you will study the effects of exploration and exploitation. For simplicity we will use Q-learning in a single-agent MDP. Implement the Q-learnig algorithm according to **Definition 7.4.1** in the course book, with the following update step:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \quad (1)$$

where α (0, 1] is the desired learning rate, and the tuple (S, A, T, R, γ) defines the MDP

- S: The states of the MDP
- A: The actions of the MDP
- T: The transition dynamics of the MDP
- R: The reward received when moving from state s to state s'
- γ: The discount factor [0, 1] indicating the importance of immediate and future rewards respectively

Use ϵ greedy action selection, i.e., with probability ϵ select a random, exploratory action, otherwise select the action with the maximum Q value. Evaluate your implementation by trying to maximize the cumulative reward (try to reach above 0.5 in average reward over 10k episodes) in the *Frozen Lake* environment:
https://gym.openai.com/envs/FrozenLake-v0/

Use the code in frozen-lake-experiment/frozen_lake.py as a starting point. Implement the agent's *act* and *learn* functions, set the initial and final values of ϵ, and implement a strategy for updating ϵ during learning (to move from exploration to exploitation). The functions *np.argmax*, *np.random.uniform* and *env.action_space.sample* are useful for action selection and random exploration. Run your experiment by executing (a plot of the result will be saved in the *learning_curves* directory):
python frozen-lake-experiment/frozen_lake.py

Evaluate the effects of ϵ on the performance. See if you can get close to or above an accumulated reward of 5000. To get a good result you will need to update ϵ from a large to a small value during training (you could try to update it continuously or periodically, directly from the start or after some episodes of learning).

In the report:

- Evaluate and discuss the effects of ϵ on performance. What strategy for updating ϵ did you use?
- Include a plot of your accumulated reward for your best result.
- What are the major difficulties for learning in this environment? Include a discussion.

3.2 Ex 2: Competitive Multi-Agent Deep Reinforcement Learning

In this experiment you will study competitive learning using the MADDPG algorithm, which uses the centralized learning, decentralized execution approach to multi-agent learning:
https://arxiv.org/pdf/1706.02275.pdf

The environment is implemented using the multi-agent particle environments:
https://github.com/johan-kallstrom/multiagent-particle-envs

The environment (*simple_hockey*) is illustrated in Figure 1. The goal of each agent (red and blue) is to move the puck (black) to a position between the goal posts (grey). Study the code in **scenarios/simple_hockey.py** and then define the functions *agent_reward* and *adversary_reward* of the reward system to achieve the desired behavior. For inspiration you can have a look at the list of environments in the repository README.md, and study the corresponding reward design in **multiagent-particle-envs/multiagent/scenarios/**.

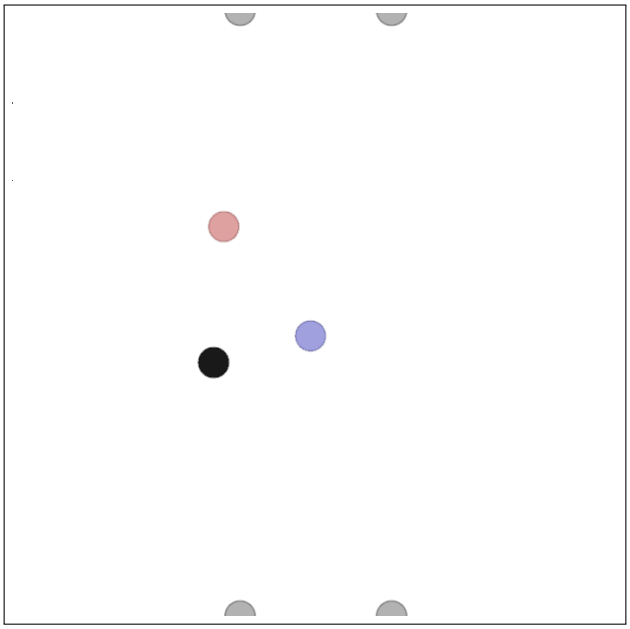

Figure 1: Simple hockey environment

Some notes:

- The position/velocity of an agent is given by agent.state.p_pos/vel
- the position/velocity of the puck is given by world.landmarks[0].state.p_pos/vel
- The positions of goal posts are at [-0.25, -1.0] and [0.25, -1.0] for the lower goal, and at [-0.25, 1.0] and [0.25, 1.0] for the upper goal
- The training can take a long time!

Train agents by executing **python maddpg/experiments/train.py** with the following command line arguments (there may be some warnings about use of deprecated functions):

- --scenario simple_hockey
- --max-episode-len 50
- --num-episodes 60000 (see note below about initial tests)
- --exp-name hockey_01
- --save-dir /tmp/hockey_01/

After training you can study the behavior of the agents by executing the same command with the command line arguments:

- --scenario simple_hockey
- --max-episode-len 50
- --load-dir /tmp/hockey_01/
- --display

Before you run a complete experiment, run using only 1000 episodes (num_episodes) to see that everything is working. After that run completes, also run the command above to display the agents' behavior (which will still be random), to see how efficient the exploration process of these agents is. This will help you understand how you should design your reward system for efficient learning (e.g., if the reward system should be sparse or dense). When designing your reward system, try to reason about the goal state of this task, and what steps the agent needs to take to reach it. How can this information be included in the reward signal?

In the report:

- Describe the reward system you designed.
- How well do the agents perform after training? Discuss your results and relate them to your reward system design.
- Are the agents equally good? Can you see any reason/explanation why they would not be?
- How do you think the length of episodes and size of the hockey rink would affect learning for your choice of reward system?

If you have time and interest to explore further: Try to investigate changes in the scenario, e.g., adding more agents, using agents with different qualities, or giving agents different tasks (e.g., attacker and defender); or try running the experiment using decentralized learning by setting the command line arguments `--goodpolicy` and `--advpolicy` to *ddpg*.