1 Directions

Solve the exercises below. You should submit your source code and a 2-page report with your results to your TA before the deadline.

Deadline: 23:59, 13 December 2019

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/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
- In your working directory, create a Python virtual environment according to these instructions (use Python 3.6):
  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 and install the following packages:
  pip install wheel
  pip install numpy==1.14.5
  pip install gym==0.10.5
  pip install tensorflow==1.8.0
  pip install -e multiagent-particle-envs
  pip install -e maddpg

3 Exercises

In this lab you will study aspects of reinforcement learning in three 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 below as a starting point. The functions *np.argmax*, *random.uniform* and env.action_space.sample are useful.

```
import gym
import numpy as np
import random

class Agent:

    def __init__(self,
                 action_space,
                 q_table_shape,
                 alpha, gamma,
                 epsilon):
        self.action_space = action_space
        self.q_table = np.zeros(shape=q_table_shape)
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon

    def act(self, state):
        # Select action for exploitation or exploitation
        # YOUR CODE HERE
        return action

    def learn(self, state, action, reward, next_state):
        # Update Q table based on experience
        # YOUR CODE HERE
        self.q_table[state, action] = new_value

env = gym.make('FrozenLake-v0')
```

```
alpha = # YOUR CODE HERE
gamma = # YOUR CODE HERE
epsilon = # YOUR CODE HERE
my_agent = Agent(env.action_space,
                 (env.observation_space.n, env.action_space.n),
                 alpha, gamma, epsilon)
num_expisodes = 10000
max_steps = 100
total_reward = 0

for i in range(num_expisodes):
    done = False
    t = 0
    ep_reward = 0
    state = env.reset()

    while t < max_steps:
        # YOUR CODE HERE
        next_state, reward, done, info = env.step(action)
        # YOUR CODE HERE
        t += 1
        ep_reward += reward
        if done:
            break
    if (# YOUR CODE HERE):
        my_agent.epsilon = # YOUR CODE HERE
    total_reward += ep_reward
print(total_reward/num_expisodes)
```

Evaluate the effects of α, γ and ϵ, and plot your accumulated reward for your best set of values. To get a good result you will need to update ϵ from a large to a small value during training. Study the values of the Q table for your best solution. What are the major difficulties for learning in this environment?

3.2 Ex 2: Cooperative Multi-Agent Deep Reinforcement Learning

In this experiment you will study cooperative 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 (*simple_spread*) is illustrated below, and is one of the multi-agent particle environments: **https://github.com/johan-kallstrom/multiagent-particle-envs**

The goal of the blue agents is to coordinate to cover the black landmarks. Study the code and try to understand how the environment works, in particular **scenarios/simple_spread.py** and the definition of the reward system. Then train agents by executing **python maddpg/experiments/train.py** with the following command line arguments:

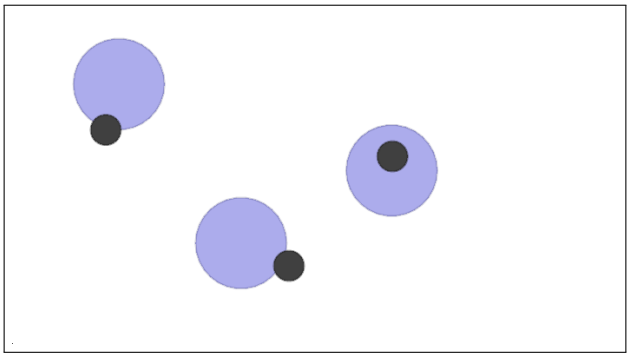
- --scenario simple_spread

- --max-episode-len 25
- --num-episodes 30000
- --exp-name spread_01
- --save-dir "/tmp/spread_01/"

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

- --scenario simple_spread
- --max-episode-len 25
- --load-dir "/tmp/spread_01/"
- --display

Select and investigate the impact of two learning parameters (lr, gamma, batch-size, num-units). Here batch-size and num-units affect the training and structure of the neural network policy. Run as many experiments as possible in parallel to save time. Plot the training progress, which is stored in the *learning_curves* directory. How do the agents perform after training? What do you think are the major challenges for learning in this environment, and how is it different from the *Frozen Lake* environment? For this cooperative environment it would have been possible to use a single agent to control all three blue objects, or alternatively strictly decentralized learning using standard single-agent reinforcement learning algorithms. What is (in general) problematic with such approaches?



3.3 Ex 3: Competitive Multi-Agent Deep Reinforcement Learning

In this experiment you will study competitive learning using the MADDPG algorithm. The environment (*simple_hockey*) is illustrated below. The goal of each agent (in red and blue) is to move the puck (in black) to a position between the goal posts (in 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. Then train agents by executing **python maddpg/experiments/train.py** with the following command line arguments:

- --scenario simple_hockey
- --max-episode-len 50
- --num-episodes 60000
- --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

How well do the agents perform? Are they 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: Try to investigate changes in the scenario, e.g. adding more agents, using agents with different qualities, or giving agents different tasks; or try running the experiment using decentralized learning by setting the command line arguments --goodpolicy and --advpolicy to *ddpg*.