Reinforcement Learning Exercise

In this exercise, you will implement and train a few reinforcement learning algorithms on a demo problem - a snake game. The basic parts will be done alone and the more advanced part with a team.

Preface

Welcome to the fourth exercise! From the first four, This one was the most fun (and hard) for me to write. As always, there is a straight-forward part and a free-form part. In the first, you will implement a few algorithms learned in class for the simple Snake game. In the second you will improve the algorithms toward more advanced versions. This second part **is to be done in teams.**
Please submit a personal report with the first part, and a team report separately in the appropriate place.

Code Sharing

It is fine to share code inside the team and to look for references online, just mention your sources in the report. We still expect you to write your own code (I don’t expect to see copied files).

Teams forming

Each team should contain three to five members. If you want to form a larger team please email me first.

Moving back from Piazza to moodle forum

I imagined the dynamics around the piazza in one way, the reality had her own decisions. We are moving back to a moodle forum.

A note about performance and evaluation

Since the environment is very simple, the advanced algorithms will not always behave better. Your work will be evaluated by your report and not by your agent’s performance. More complicated environments tend to converge too slowly to be homework.

Algorithm reference -
https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f
The recitation notes are also meant to be a reference.

A note about logging

I highly recommend to use tensorboard extensively to log and compare different components of your algorithm. The training reward and epsilon are already logged.

The environment

You can change the supplied files. Please read “models.py”, “q_policy.py” and “train_snake.py” before starting. You will need to add more files. You can run “snake_wrapper” to get a feel of the game, but your agent will only get a 9x9 square around his head (already rotated with the head direction). Details in “q_policy.py.” “snake_wrapper” also contains an explanation of the game.

Setup

You need PyTorch, NumPy, matplotlib and Tensorboard, and openai “gym” library. We will not use the built-in environments from “gym” but use our own one. For computation: I solved this exercise on my laptop, with I5-gen5 CPU.

First part (each student submit a report)

1. Implement and train the following RL algorithms for “Snake”:
    a. Q-Learning
    b. Vanilla REINFORCE (noted as: “Algorithm 1: Reinforce / Monte Carlo Policy Gradient” in the recitation)
2. Test the impact of the following parameters on algorithm convergence and performance. Compare the results to your expectations.
    a. gamma: the future-reward decay factor
    b. epsilon / entropy: exploration factors

Implement algorithms:

the supplied code

The game:

snake.py - Here the game logic resides. There is no reason to change this file
snake_wrapper.py - The interface to the game. run the main in this file to see a random game. There is no reason to change this file either.

partial training code:

train_snake.py - training loop and main function.
base_policy.py - Base class for policy implementations.
memory.py - utility used in base_policy.
models.py - Where to implement your models.
q_policy.py - Where to put the code of the q-learning.

Using this training code is not mandatory but I hope it will help you to arrange your work. You are welcome to change it as you wish.

Implementing and training the algorithms

- While implementing the algorithms, I recommend using a very simple and small model for debugging. In the later evaluation, you should use a model that sees more of the observation in order to see better results.
- Please read the code of the training loop. For q-learning, I recommend starting with the default hyperparameters. since we are not training in each playing-step, q-learning will work fine with a single network (no separate “target-network”).
- Your models are supposed to get a positive score. BUT, you don’t need to make an extra effort in order to get the best possible result.

For each algorithm, write pseudocode, implement the algorithm, and train it. Please submit the pseudo-code and a basic comparison between the different algorithms learning curves and performance, and also compare the results to your expectations. If you made any interesting decisions, explain them.

Second part (each team submit a report) - other improvements and research

There are plenty of things you can do. Please choose one:

- Q-Actor-Critic
- Implement A2C - Advantage Actor Critic
- Implement double-Q-learning.
- Implement PG or A2C or QAC with LSTM
- Implement A3C (probably more work)
- Implement PPO (probably harder, we didn't learn t in class)

And one from here:

- Change the network architecture as described in "Dueling Network Architectures for Deep Reinforcement Learning" and compare the estimated q value of Q-learning to the actual Q-value, with and without the "dueling architecture".
- Take one of the implemented algorithms and compare it's performance with and without reward normalization. Explain how you chose to normalize the rewards and why.

You can suggest other ideas if you like!