Reinforcement Learning May 1, 2019

Project - Q-Learning on Atari

Lecturer:Yishay Mansour *TA: Lee Cohen*

Due Date - 3.6.2019

Introduction

This assignment requires you to implement and evaluate Q-Learning with convolutional neural networks for playing Atari games. The Q-learning algorithm was covered in lecture, and you will be provided with starter code. You may run the code either on GPU or CPU. A GPU machine will be faster, but you should be able to get good results with about 20 hours of compute on a modern CPU. Please start early! The questions will require you to perform multiple runs of Q-learning, each of which can take quite a long time. Furthermore, depending on your implementation, you may find it necessary to tweak some of the parameters, such as learning rates or exploration schedules, which can also be very time consuming. The actual coding for this assignment will involve about 50 lines of code, but the evaluation may take a very long time.

Installation

Obtain the code from the website. To run the code, simply execute: python `main.py` or `ram.py`. You will also need to install the dependencies, which are OpenAI gym and OpenCV (which is used to resize the images):

1. Remember to also follow the instructions for installing the Atari environments for OpenAI gym, which can be found on the OpenAI gym github page. You will need version 0.9.5 - which you can install via `pip install "gym[atari]"==0.9.5`.
2. To install OpenCV, run `pip install opencv-python`. If you have trouble with ffmpeg you can install it via homebrew or apt-get depending on your system.
3. For those of you who registered for GPU access, your group name is "`cs_reinf_learning_#`". You can find out what is your group number by running the 'groups' command. In order to check which GPU is available, use the command 'nvidia-smi'. You can use the servers: *rack-gamir-g0[4,5,6]* and *savant*. Each server is equipped with 8 GPUs.

In order to share a folder and the files in it with your group members, you need to give full privilege to the group. See `https://www.unixtutorial.org/difference-between-chmod-and-chown` for more information.

Implementation

The first phase of the assignment is to implement a working version of Q-learning. The default code will run the Pong game with reasonable hyperparameter settings. The starter code already provides you with a working replay buffer, all you have to do is fill in parts of `dqn_learn.py`, by searching for "YOUR CODE HERE". The comments in the code describe what should be implemented in each section. You may find it useful to look inside utils dir to understand how the replay buffer works, but you should not need to modify it. To determine if your implementation of Q-learning is performing well, you should run it with the default hyperparameters on the Pong game. Our reference solution gets a reward of around -20 to -15 after 500k steps, -15 to -10 after 1m steps, -10 to -5 after 1.5m steps, and around +10 after 2m steps on Pong. The maximum score of around +20 is reached after about 4-5m steps. However, there is considerable variation between runs.

Evaluation

Once you have a working implementation of Q-learning, you should prepare a report. The report should consist of one figure for each question below. You should turn in the report as one pdf and a zip file with your code. If your code requires special instructions or dependencies to run, please include these in a file called README inside the zip file. For all the questions below, it is your choice how long to run for. Although running for 2-4m steps is ideal for a solid evaluation, especially when running on CPU, this may be difficult. We strongly recommend running at least 1m steps, and including at least one run of 4m steps for Question 1.

Question 1: Basic Q-learning performance

Include a learning curve plot showing the performance of your implementation on the game Pong. The x-axis should correspond to number of time steps (consider using scientific notation) and the y-axis should show the mean 100-episode reward as well as the best mean reward. Be sure to label the y-axis. If you needed to modify the default hyperparameters to obtain good performance, include the hyperparameters in the caption. You only need to list hyperparameters that were modified from the defaults.

Note: you can use the Pong RAM version for debugging, but you should submit solution for regular version Pong (with pixel).

Question 2: Experimenting with hyperparameters

Now let's analyze the sensitivity of Q-learning to hyperparameters. Choose one hyperparameter of your choice and run at least **three** other settings of this hyperparameter, in addition to the one used in Question 1, and plot all four values on the same graph. Your choice what you experiment with, but you should explain why you choose this hyperparameter in the caption and the report.

Examples include: learning rates, neural network architecture, exploration schedule or exploration rule (e.g. you may implement an alternative to ϵ-greedy), etc. Discuss the effect of this hyperparameter on performance in the caption. You should find a hyperparameter that makes a nontrivial difference on performance.
Note: You might consider performing a hyperparameter sweep for getting good results in Question 1, in which case it's fine to just include the results of this sweep for Question 2 as well, while plotting only the best hyperparameter setting in Question 1.

Note: for Question 2, you may run on other games or multiple games, but please submit results for Question 1 using only the game Pong. Running on multiple games may require considerable time or computing power, so it is not required, but encouraged. If you have any other interesting experiments you wish to report on, you may include those after your answer to Question 2.

Bonus

Interesting additional experiments or extensions will be considered for bonus points (up to 10 pts). Add to your report explanation regarding the experiments and plot the results as in previous question.