HW #4: Deep RL

Introduction

In this homework you will implement an RL-based agent, which plays the Ms. Pacman Atari game. In this game, the goal is for Ms. Pacman to consume as many pellets as possible, without being eaten by the in-game ghosts. See here for a video of Ms. Pacman game-play. You can also play the game yourself online, here.

This homework will differ from the previous homeworks in several key respects:

- You will have a significant amount of freedom in what you implement.
- Your implementation must be your own, but you may consult any paper, article, or code repository that you wish.
- You will need to describe what you have done in a short report.

Getting started

Download the starter code from this folder; there are only two files. You will also need to install the OpenAI gymnasium library, which can be done with the following commands:

```
pip install gymnasium
pip install “gymnasium[atari]”
pip install “gymnasium[accept-rom-license]”
```

The starter code implements a simple policy-gradient-based agent for you. Your goal will be to improve over this simple baseline. You can train the agent implemented in the starter code by running the following:

```
OMP_NUM_THREADS=4 python -u hw4_agent.py
```

By default, the code evaluates the trained agent after every 50 gradient steps. You can visualize your agent at evaluation time by passing the --render argument to the script:

```
OMP_NUM_THREADS=4 python -u hw4_agent.py --render
```

What you should do

You have 3 subtasks, as follows.

Subtask 1: Improve the learning algorithm

Currently the starter code contains a relatively basic policy gradient implementation. Your first task is to implement a less naive learning algorithm.

Some options you might consider:

- Implement Deep Q-Learning
- Implement Advantage Actor Critic
- Label some data and train with imitation learning

In addition to implementing this improved learning algorithm in code, you should describe what you did in writing. This section of your report does not need to be more than one page, but it should:

- Explain what you did clearly, which will likely involve writing down a few equations.
- Explain why you chose to do what you did.
- Summarize the results you obtained (i.e., report how many points/how much reward your new agent accumulated).

Subtask 2: Improve the model

No matter what learning algorithm you use, you will need a model that scores the goodness of taking an action in a particular state. In the starter code, `PolicyNetwork` (defined in `hw4_agent.py`) uses a very simple convolutional model, which maps from the raw current frame of the game (as pixels) to logits for each action. Your second task is to improve this model.

Some options you might consider:

- Implement a recurrent model, where action scores depend not just on the current frame, but on previous frames too.
- Implement a more sophisticated convolutional model.
- Implement some preprocessing on top of the game frame, which makes a relatively simple model more effective.

You should implement your model by subclassing `nn.Module`, just as `PolicyNetwork` does. You can implement it in any way you want, but it must have the following two capabilities:

1. It **must** implement a `get_action()` function, with the same signature and return values as the `get_action()` function in `PolicyNetwork`. This will allow us to run `hw4_utils.validate()` on it.
2. It must be possible to save and then re-load your model's parameters. (This should not take any extra work; see `hw4_main.py` for how this is done in `PolicyNetwork`).

In addition to implementing your improved model in code, you should describe what you did in writing. This section of the report also does not need to be more than a page, and it should:

- Explain your model clearly.
- Explain why you chose to implement this model.
- Summarize any results you obtained. Since this is supposed to be an improvement on a baseline model, you should have some results comparing the baseline model to your improved model in terms of points/reward accumulated.

Subtask 3: Explain how to run your code

The course staff will attempt to run your code. So the third section of your report should explain how to train your model, and how to evaluate it using saved parameters that you will also provide. “Evaluate” here means run `hw4_utils.validate()` on your model, as is done in `hw4_agent.py`. If your model implements the `get_action()` function correctly, this should not require any extra work.

This section of your report can be very short. It should just provide the commands necessary to train and evaluate your model.

Rubric

As noted above, your report will have 3 sections, one for each subtask. There will also be code associated with the first two subtasks. In excruciating detail, the rubric for grading each subtask is as follows.

Subtask 1 is worth 45% of the total grade. This 45% will further break down into:
- Correctness of the code (15%)
- Clarity of your writing (10%)
- Thoroughness of experiments (10%)
  - You should try a few settings of the two or three most important hyperparameters, and report these results. E.g., try a few learning rates, and try a few batch sizes, and report how these affect performance.
- Ambition of the proposed improvement (10%)
  - Any of the options listed under “options you might consider” in Subtask 1 is sufficient for full credit. If you do something significantly less ambitious (e.g., change the batch size, use a non-parameterized baseline value) you will get less than full credit.

Subtask 2 is worth 45% of the total grade. This 45% will further break down into:
- Correctness of the code (15%)
- Clarity of your writing (10%)
- Thoroughness of experiments (10%)
  - You should try a few settings of the two or three most important hyperparameters, and report these results. E.g., try a few learning rates, and try a few different model dimensions or kernel widths.

- Ambition of the proposed improvement (10%)
    - Any of the options listed under "options you might consider" in Subtask 2 is sufficient for full credit. If you do something significantly less ambitious (e.g., change the number of output channels in a layer, change the nonlinearity) you will get less than full credit.

Subtask 3 is worth 10% of the total grade. You will get full credit if you explain how to run your code, and if following your written instructions indeed gets the code to run.

Extra credit

The top 3 best-performing models will receive 10 points of extra credit on this assignment.

Deliverables

Please upload the following to Gradescope by 11:59 PM on May 5th, 2023:

1. A pdf file containing your report, in the format described above.
2. All the code necessary to train your agent, and to load saved parameters into your model and then run the evaluation/validation function on it.
3. The saved model parameters that should be loaded into your model. If these are too big to upload to Gradescope, put a link at which your model parameters can be downloaded in your report.

Assignment Ground Rules

- All work must be your own. You may discuss the assignment with your classmates, but everything you write (code or text) must be your own.
- You **may** consult any resource you wish (e.g., textbooks, papers, articles, websites) **including online code repositories**. If you do consult someone else's code, you should not copy it directly, but rather adapt it.
- The **only** external libraries you may use are PyTorch (including all its sub-libraries) and gymnasium.
- You should do **all** your experiments on CPUs.
- You can implement your models and training in any way you want, as long as you tell the course staff how to run it (in your report), and as long as your model can be evaluated using `hw4_utils.validate()`.

TLDR

You must submit:

- A script that trains your model. Your model must implement a `get_action()` function that takes the same arguments and has the same return values as that implemented in `hw4_agent.py`.
- A script that loads a saved model and calls `hw4_utils.evaluate()` on it. This can be the same script as the one that trains your model.
- A saved model file. If your model is too big to upload to gradescope, you may put a link at which your model file can be downloaded into your report.
- A pdf report that has three sections:
    - Section 1 describes your improved learning algorithm and the results you obtained from running it.
    - Section 2 describes your improved model and the results you obtained from using it (together with your improved learning algorithm).
    - Section 3 describes how to run your code.

There is also extra credit for the 3 best-performing models/approaches.

FAQ

Q: Will I be able to get very good performance out of the agent I train?
A: Probably not. It's difficult to perform well at these tasks without a lot of compute, and sometimes even with. The goal of this homework is just for you to get some hands-on experience with these models and algorithms.

Q: Is it bad if my improved learning algorithm doesn't significantly improve over the baseline?
A: No. If your implementation is correct and your reasons for choosing your proposed improvement are reasonable, you can still get full credit.

Q: Is it bad if my improved model doesn't significantly improve over the baseline?
A: No. See the previous question.

Q: Do I need to implement one of the suggested improvements (i.e., one of the bullet points under "some options you might consider") to get full credit?
A: No, these are just suggestions to give you a sense of the scale of what you should be aiming for. You can improve the learning algorithm or the model in any way you want.

Q: How many updates should I train for before I report results?
A: Aim for at least a thousand.

Q: If I want to use a CNN or an LSTM or a transformer, do I have to implement it from scratch?
A: No. PyTorch's nn library has implementations of all of these (as well as other standard architectures) and you are free to use them.

Q: What is the `prev_state` in `PolicyNetwork`? It doesn't seem to be used.
A: This allows you to pass in previous agent state if you want action scores to be recurrent (i.e., to condition on previous observations and actions).

Q: The starter code uses Adam to optimize, and a constant learning rate schedule. Can I change these?
A: Certainly.

Q: Can I run experiments in the course container without keeping a browser window open and active?
A: Yes, you can use tmux.

Q: When I run the code inside an IPython/Jupyter notebook the rendering doesn't work?
A: See notebook_viz.ipynb in the Google Drive folder for an example of how you can do this. However, try to use notebooks sparingly since they are pretty silly.

Resources you may find helpful

To read up on deep RL algorithms, it may be useful to consult OpenAI's Spinning Up tutorial or Hugging Face's Deep RL Course. There are of course many other helpful online resources that you should feel free to consult.