Homework 3 : Q-Learning Algorithms

Due to March 09 2023 6pm

Introduction

Part 1 of this assignment requires you to implement and evaluate Q-learning for playing Atari games. The Q-learning algorithm was covered in lecture, and you will be provided with starter code. This assignment will be faster to run on a GPU, though it is possible to complete on a CPU as well. Note that we use convolutional neural network architectures in this assignment. Please start early! For references to this type of approach, see this paper and this paper.

Part 1: DQN

We will be building on the code that we have implemented in the first two assignments. All files needed to run your code are in the `hw3` folder, but there will be some blanks you will fill with your solutions from homework 1. These locations are marked with `# TODO: get this from hw1, hw2` and are found in the following files:

- `infrastructure/rl_trainer.py`
- `infrastructure/utils.py`
- `policies/MLP_policy.py`

In order to implement deep Q-learning, you will be writing new code in the following files:

- `agents/dqn_agent.py`
- `critics/dqn_critic.py`
- `policies/argmax_policy.py`

There are two new package requirements (`opencv-python` and `gym[atari]`) beyond what was used in the first two assignments; make sure to install these with `pip install -r requirements.txt` if you are running the assignment locally.

Implementation

The first phase of the assignment is to implement a working version of Q-learning. The default code will run the `Ms.Pac-Man` game with reasonable hyperparameter settings. Look for the `# TODO` markers in the files listed above for detailed implementation instructions. You may want to look inside `infrastructure/dqn_utils.py` to understand

how the (memory-optimized) replay buffer works, but you will not need to modify it.

Once you implement Q-learning, answering some of the questions may require changing hyperparameters, neural network architectures, and the game, which should be done by changing the command line arguments passed to `run_hw3_dqn.py` or by modifying the parameters of the `Args` class from within the Colab notebook.

To determine if your implementation of Q-learning is correct, you should run it with the default hyperparameters on the `Ms.Pac-Man` game for 1 million steps using the command below. Our reference solution gets a return of 1500 in this timeframe. On Colab, this will take roughly 3 GPU hours. If it takes much longer than that, there may be a bug in your implementation.

To accelerate debugging, you may also test on `LunarLander-v3`, which trains your agent to play Lunar Lander, a 1979 arcade game (also made by Atari) that has been implemented in OpenAI Gym. Our reference solution with the default hyperparameters achieves around 150 reward after 350k timesteps, but there is considerable variation between runs and without the double-Q trick the average return often decreases after reaching 150. We recommend using `LunarLander-v3` to check the correctness of your code before running longer experiments with `MsPacman-v0`.

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. Also provide the log file of your run on gradescope named as `pacman_1.csv`.

Question 1: basic Q-learning performance (DQN). Include a learning curve plot showing the performance of your implementation on `Ms. Pac-Man`. The x-axis should correspond to number of time steps (consider using scientific notation) and the y-axis should show the average per-epoch reward as well as the best mean reward so far. These quantities are already computed and printed in the starter code. They are also logged to the `data` folder, and can be visualized using Tensorboard as in previous assignments. Be sure to label the y-axis, since we need to verify that your implementation achieves similar reward as ours. You should not need to modify the default hyperparameters in order to obtain good performance, but if you modify any of the parameters, list them in the caption of the figure. The final results should use the following experiment name:

```
python ift6131/scripts/run_hw3_dqn.py env_name=MsPacman-v0 exp_name=q1
```

Question 2: double Q-learning (DDQN). Use the double estimator to improve the accuracy of your learned Q values. This amounts to using the online Q network (instead of the target Q network) to select the best action when computing target values. Compare the performance of DDQN to vanilla DQN. Since there is considerable variance between runs, you must run at least three random seeds for both DQN and DDQN. You may use `LunarLander-v3` for this question. The final results should use the following experiment names:

```
python run_hw3.py env_name=LunarLander-v3 exp_name=q2_dqn_1 seed=1
python run_hw3.py env_name=LunarLander-v3 exp_name=q2_dqn_2 seed=2
python run_hw4.py env_name=LunarLander-v3 exp_name=q2_dqn_3 seed=3
```

```
python run_hw3.py env_name=LunarLander-v3 exp_name=q2_doubledqn_1 double_q=
    true seed=1
python run_hw3.py env_name=LunarLander-v3 exp_name=q2_doubledqn_2 double_q=
    true seed=2
python run_hw3.py env_name=LunarLander-v3 exp_name=q2_doubledqn_3 double_q=
    true seed=3
```

Submit the run logs for all the experiments above. In your report, make a single graph that averages the performance across three runs for both DQN and double DQN. See `scripts/read_results.py` for an example of how to read the evaluation returns from Tensorboard logs.

Question 3: 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 chose this hyperparameter in the caption. Examples include: (1) learning rates; (2) neural network architecture for the Q network, e.g., number of layers, hidden layer size, etc; (3) exploration schedule or exploration rule (e.g. you may implement an alternative to ϵ-greedy and set different values of hyperparameters), 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 3 as well, while plotting only the best hyperparameter setting in Question 1. The final results should use the following experiment name:

```
python run_hw3_dqn.py env_name=LunarLander-v3 exp_name=q3_hparam1
python run_hw3_dqn.py env_name=LunarLander-v3 exp_name=q3_hparam2
python run_hw3_dqn.py env_name=LunarLander-v3 exp_name=q3_hparam3
```

You can replace `LunarLander-v3` with `PongNoFrameskip-v4` or `MsPacman-v0` if you would like to test on a different environment.

Part 2: DDPG

Implement the DDPG algorithm.

In order to implement Deep Deterministic Policy Gradient (DDPG), you will be writing new code in the following files:

- `agents/ddpg_agent.py`
- `critics/ddpg_critic.py`
- `policies/MPL_policy.py`

DDPG is programmed a little different that the rl algorithms so far. DDPG does not use n-step returns to estimate the advantage given a large batch of on-policy data. Instead, DDPG is off-policy. DDPG trains a Q-Function $Q(\mathbf{s}_t, \mathbf{a}_t, \phi)$ to estimate the policy *reward-to-go* if for a state and action. This model can then be used as the objective to optimize the current policy.

$$\begin{aligned} \nabla_{\theta^\mu} J &\approx \mathbb{E}_{\mathbf{s}_t \sim \rho^\beta} \left[\nabla_\theta Q(\mathbf{s}_t, \mathbf{a}_t | \phi)|_{s=\mathbf{s}_t, a=\mu(s_t|\theta)} \right] \\ &= \mathbb{E}_{s_t \sim \rho^\beta} \left[\nabla_a Q(\mathbf{s}_t, \mathbf{a}_t | \phi)|_{s=\mathbf{s}_t, a=\mu(\mathbf{s}_t)} \nabla_{\theta_\mu} \mu(s|\theta)|_{s=s_t} \right] \end{aligned} \tag{1}$$

Algorithm 1 DDPG algorithm

1: init $\phi' \leftarrow \phi$ and $\theta' \leftarrow \theta$ to random networks and $\mathcal{D} \leftarrow \{\}$
2: **for** $l \in 0, \ldots, L$ **do**
3: take some action $\mathbf{a}_t$ and receive $\{\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}'_t\}$, add to $\mathcal{D}$
4: Sample batch of data $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}_{i=1}^N$ from $\mathcal{D}$
5: $y_i \leftarrow r_i + \gamma Q(\mathbf{s}'_i, \mu(\mathbf{s}'_i, \theta'), \phi')$ {Compute Target}
6: Update ϕ by minimizing $\frac{1}{N} \sum^N ||Q(\mathbf{s}_i, \mathbf{a}_i, \phi) - y_i||^2$ {Update critic}
7: Update $\theta \leftarrow \theta + \beta \frac{\partial Q_\phi(\mathbf{s}_t, \mathbf{a}_t)}{\partial \mu(\mathbf{s}_t|\theta)} \frac{\partial \mu(\mathbf{s}_t|\theta)}{\partial \theta}$ {Update Actor}
8: Update $\theta' \leftarrow \rho\theta' + (1-\rho)\theta$ and $\phi' \leftarrow \rho\phi' + (1-\rho)\phi$ {Using Polyak averaging}
9: **end for**

Question 4: Experiments (DDPG) For this question the goal is to implement DDPG and tune a few of the hyper parameters to improve the performance. Try different update frequencies for the Q-Function and actor. Also, try different learning rates for the Q-function and actor. First try different learning rates

```
python run_hw3.py  exp_name=q4_ddpg_up<b>_lr<r> rl_alg=ddpg
env_name=InvertedPendulum-v2 atari=false

python run_hw3.py  exp_name=q4_ddpg_up<b>_lr<r>  rl_alg=ddpg
env_name=InvertedPendulum-v2 atari=false
```

```
python run_hw3.py   exp_name=q4_ddpg_up<b>_lr<r>  rl_alg=ddpg env_name=
    InvertedPendulum-v2 atari=false
```

Next try different update frequencies for training the policies.

```
python run_hw3.py    exp_name=q4_ddpg_up<b>_lr<r>  rl_alg=ddpg
env_name=InvertedPendulum-v2 atari=false

python run_hw3.py  exp_name=q4_ddpg_up<b>_lr<r>  rl_alg=ddpg
env_name=InvertedPendulum-v2 atari=false

python run_hw3.py    exp_name=q4_ddpg_up<b>_lr<r>  rl_alg=ddpg
env_name=InvertedPendulum-v2 atari=false
```

Submit the learning graphs from these experiments along with the write-up for the assignment.

Question 5: Best parameters on a more difficult task After you have completed the parameter tuning on the simpler *InvertedPendulum-v2* environment use those parameters to train a model on the more difficult *HalfCheetah-v2* environment.

```
python run_hw4.py   exp_name=q5_ddpg_hard_up<b>_lr<r>  rl_alg=ddpg
env_name=HalfCheetah-v2 atari=false
```

Include the learning graph from this experiment in the write-up as well. Also provide the log file of your run on gradescope named as `half_cheetah_5.csv`.

Part 3: TD3

In order to implement Twin Delayed Deep Deterministic Policy Gradient (TD3), you will be writing new code in the following files:

- `critics/td3_critic.py`

This is a relatively small change to DDPG to get TD3. Implement the additional target q function for TD3

Question 6: TD3 tuning Again the hyper parameters for this new algorithm need to be tuned as well using *InvertedPendulum-v2*. Try different values for the noise being added to the target policy ρ when computing the target values. Also try different Q-Function network structures. Start with trying different values for ρ.

```
python run_hw3.py  exp_name=q6_td3_shape<s>_rho<r>  rl_alg=ddpg
env_name=InvertedPendulum-v2 atari=false

python run_hw3.py  exp_name=q6_td3_shape<s>_rho<r>  rl_alg=ddpg
env_name=InvertedPendulum-v2 atari=false
```

Algorithm 2 TD3

init $\phi' \leftarrow \phi$ and $\theta' \leftarrow \theta$ to a random network and $\mathcal{D} \leftarrow \{\}$
for $l \in 0, \ldots, L$ **do**
  take some action $\mathbf{a}_t$ and receive $\{\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1}\}$, add to $\mathcal{D}$
  Sample batch of data $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}_{i=1}^{N}$ from $\mathcal{D}$
  $y_i \leftarrow r_i + \gamma Q(\mathbf{s}'_i, \mu(\mathbf{s}'_i, \theta') + \epsilon, \phi'),\ \epsilon \sim \text{clip}(\mathcal{N}(0, I) \cdot \rho), -c, c)$ {Compute Target}
  Update ϕ by minimizing $\frac{1}{N} \sum^{N} ||Q(\mathbf{s}_i, \mathbf{a}_i, \phi) - y_i||^2$ {Update critic}
  if $t \mod d$ **then**
    Update $\theta \leftarrow \theta + \beta \frac{\partial Q_\phi(\mathbf{s}_t, \mathbf{a}_t)}{\partial \mu(\mathbf{s}_t|\theta)} \frac{\partial \mu(\mathbf{s}_t|\theta)}{\partial \theta}$ {Update Actor}
    Update $\theta' \leftarrow \rho\theta' + (1-\rho)\theta$ and $\phi' \leftarrow \rho\phi' + (1-\rho)\phi$ {Using Polyak averaging}
  end if
end for

```
python run_hw3.py  exp_name=q6_td3_shape<s>_rho<r>  rl_alg=ddpg
env_name=InvertedPendulum-v2 atari=false
```

Next try different update frequencies for training the policies.

```
python run_hw3.py exp_name=q6_td3_shape<s>_rho<r>  rl_alg=ddpg
env_name=InvertedPendulum-v2 atari=false

python run_hw3.py exp_name=q6_td3_shape<s>_rho<r>  rl_alg=ddpg
env_name=InvertedPendulum-v2 atari=false

python run_hw3.py exp_name=q6_td3_shape<s>_rho<r>  rl_alg=ddpg
env_name=InvertedPendulum-v2 atari=false
```

Include the results from these hyper-parameter experiments in the assignment write-up. Make sure to comment clearly on the parameters you studied and why which settings have worked better than others.

Question 7: Evaluate TD3 compared to DDPG In this last question, evaluate TD3 compared to DDPG. Using the best parameter setting from **Q6** train TD3 on the more difficult environment used for **Q5**.

```
python run_hw3.py    exp_name=q6_td3_shape<s>_rho<r> rl_alg=td3
env_name=HalfCheetah-v2 atari=false
```

Include the learning graph from this experiment in the write-up for the assignment. Make sure to comment on the different performance between DDPG and TD3 and what makes the performance different. Also provide the log file of your run on gradescope named as `half_cheetah_7.csv`.

Bonus: For finding issues or adding features Keeping up with the latest research often means adding new types of programming questions to the assignments. If you find issues with the code and provide a solution you can receive bonus points. Also, adding features that help the class can also result is bonus points for the assignment.

Part 4: SAC

In order to implement Soft Actor-Critic (SAC), you will be writing new code in the following files:

- `agents/sac_agent.py`
- `critics/sac_critic.py`
- MLPPolicyStochastic in `policies/MLP_policy.py`

SAC is similar to TD3 except for his central feature which is entropy regularization. The policy is trained to maximize a trade-off between expected return and entropy, a measure of randomness in the policy.The algorithm and some relevant resources are available here. **Bonus**: Add linear annealing to alpha (the entropy coefficient).

Question 8: SAC entropy tuning Again the hyper parameters for this new algorithm need to be tuned as well using *InvertedPendulum-v2*. Try different values for the entropy coefficient α being added to the loss.

```
python run_hw3.py    exp_name=q6_sac_alpha<a>  rl_alg=ddpg
env_name=InvertedPendulum-v2 atari=false

python run_hw3.py    exp_name=q6_sac_alpha<a>  rl_alg=ddpg
env_name=InvertedPendulum-v2 atari=false

python run_hw3.py    exp_name=q6_sac_alpha<a>  rl_alg=ddpg
env_name=InvertedPendulum-v2 atari=false
```

Question 9: Evaluate SAC compared to DDPG In this last question, evaluate SAC compared to DDPG. Using the best parameter setting from **Q6** train SAC on the more difficult environment used for **Q5**.

```
python run_hw3.py    exp_name=q6_td3_shape<s>_rho<r> rl_alg=td3 env_name=
    HalfCheetah-v2 atari=false
```

Include the learning graph from this experiment in the write-up for the assignment. Make sure to comment on the different performance between DDPG and SAC and what makes the performance different. Also provide the log file of your run on gradescope named as `half_cheetah_9.csv`.

Submission

We ask you to submit the following content on the course GradeScope :

Submitting the PDF.

Your report should be a PDF document containing the plots and responses indicated in the questions above.

Submitting log files on the autograder.

Make sure to submit all the log files that are requested by GradeScope AutoGrader, you can find them in your log directory `/data/exp_name/` by default.

Submitting code, experiments runs, and videos.

In order to turn in your code and experiment logs, create a folder that contains the following:

- A folder named `data` with all the experiment runs from this assignment. **Do not change the names originally assigned to the folders, as specified by `exp_name` in the instructions.**
- The `roble` folder with all the `.py` files, with the same names and directory structure as the original homework repository (not include the `outputs/` folder). Additionaly, include the commands (with clear hyperparameters) and the config file `conf/config_hw3.yaml` file that we need in order to run the code and produce the numbers that are in your figures/tables (e.g. run "python run_hw3.py –ep_len 200") in the form of a README file. Finally, your plotting script should also be submitted, which should be a python script (or jupyter notebook) such that running it can generate all plots from your pdf. This plotting script should extract its values directly from the experiments in your `outputs/` folder and should not have hardcoded reward values.
- You must also provide a video of your final policy for each question above. To enable video logging, set both flags video log_freq to be greater than 0 and render to be true in `conf/config_hw3.yaml` **before** running your experiments. Videos could be fin as `.mp4` files in the folder: `outputs/.../data/exp_name/videos/`. **Note :** For this homework, the atari envs should be in the folder `gym` and the other in the folder `video`.

As an example, the unzipped version of your submission should result in the following file structure. **Make sure to include the prefix `q1_`,`q2_`,`q3_`,`q4_`, and `q5_`.**

```
submit.zip
├─ hw1
├─ hw2
├─ hw3
│  ├─ run_logs
│  │  ├─ q1
│  │  │  ├─ q1_cheetah_n500_arch1x32
│  │  │  │  ├─ events.out.tfevents.1567529456.e3a096ac8ff4
│  │  │  │  └─ eval_step_x.mp4
│  │  │  └─ q1_cheetah_n5_arch2x250
│  │  │     ├─ events.out.tfevents.1567529456.e3a096ac8ff4
│  │  │     └─ eval_step_x.mp4
│  │  ├─ q2
│  │  │  └─ ...
│  │  └─ ...
│  ├─ roble
│  │  ├─ agents
│  │  │  ├─ dqn_agent.py
│  │  │  └─ ...
│  │  ├─ policies
│  │  │  └─ ...
│  │  └─ ...
│  ├─ conf
│  │  └─ config_hw3.yaml
│  ├─ README.md
│  └─ ...
├─ Dockerfile
├─ diff.txt
└─ ...
```

You also need to include a diff of your code compared to the starter homework code.

You can use the command

```
git diff 8ea2347b8c6d8f2545e06cef2835ebfd67fdd608 >> diff.txt
```

1. If you are a Mac user, **do not use the default "Compress" option to create the zip**. It creates artifacts that the autograder does not like. You may use `zip -vr submit.zip submit -x "*.DS_Store"` from your terminal.
2. Turn in your assignment on Gradescope. Upload the zip file with your code and log files to **HW3 Code**, and upload the PDF of your report to **HW3**.