CS285 Deep Reinforcement Learning HW3: Q-Learning and Actor-Critic

Due: October 21st 2019, 11:59 pm

1 Part 1: Q-Learning

1.1 Introduction

Part 1 of 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. 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.

1.2 Installation

Obtain the code from `https://github.com/berkeleydeeprlcourse/homework_fall2019/tree/master/hw3`. To run the code, go into the `hw3` directory and simply execute `python cs285/scripts/run_hw3_dqn.py`. It will not work until you finish implementing the algorithm in `agents/dqn_agent.py`, `critics/dqn_critic.py` and `policies/argmax_policy.py`, in addition to copying the relevant bits that you implemented in HW1 and HW2, as specified in the README.

To install OpenCV, run `pip install opencv-python==3.4.0.12`. The only additional dependency you might have to install is `gym[atari]==0.10.11`, and you can skip this step if you installed `gym[all]` at the start of the class. You

also need to replace `<pathtogym>/gym/envs/box2d/lunar_lander.py` with the provided `lunar_lander.py file`. To find the path, run:

```
locate lunar_lander.py
```

(or if there are multiple options run):

```
source activate cs285_env
ipython
import gym
gym.__file__
```

which will print `<pathtogym>/gym/__init__.py`.

You may want to look at `scripts/run_hw3_dqn.py` before starting the implementation.

1.3 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, and you have to fill in parts of `agents/dqn_agent.py`, `critics/dqn_critic.py` and `policies/argmax_policy.py` by searching for `TODO`. You will also need to reuse your code from homework 1 and homework 2 (e.g., `utils.py`, `tf_utils.py`, `rl_trainer.py`) The comments in the code describe what should be implemented in each section. You could look inside `infrastructure/dqn_utils.py` to understand how the replay buffer works, but you should not need to modify it. You may also look inside `run_hw3_dqn.py` to change the hyperparameters or the particular choice of Atari game. 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`.

To determine if your implementation of Q-learning is performing well, you should run it with the default hyperparameters on the Pong game (see the command below). 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. For Q1, you must run the algorithm for at least 3m timesteps (and you are encouraged to run for more), and you must achieve a final reward of at least +10 (i.e. your trained agent beats the opponent by an average of 10 points).

To accelerate debugging, you may also try out `LunarLander-v2`, which trains your agent to play Lunar Lander, a 1979 arcade game (also made by Atari) that has been implemented in OpenAI Gym. Note that you cannot run lunar

remotely because it opens a window to render images. Our reference solution with the default hyperparameters achieves around 150 reward after 500k timesteps, but there is considerable variation between runs, and the method sometimes experience instabilities (i.e. the reward goes down after achieving 150). We recommend using `LunarLander-v2` to check the correctness of your code before running longer experiments with `PongNoFrameSkip-v4`.

1.4 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. You need to include at least one run of 3m steps for Question 1.

Question 1: basic Q-learning performance. (DQN) 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. 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, similar to 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 run_hw3_dqn.py --env_name PongNoFrameskip-v4 --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 must use `LunarLander-v2` for this question. The final results should use the following experiment names:

```
python run_hw3_dqn_atari.py --env_name LunarLander-v2 --exp_name
↪ q2_dqn_1 --seed 1
python run_hw3_dqn_atari.py --env_name LunarLander-v2 --exp_name
↪ q2_dqn_2 --seed 2
python run_hw3_dqn_atari.py --env_name LunarLander-v2 --exp_name
↪ q2_dqn_3 --seed 3
```

```
python run_hw3_dqn_atari.py --env_name LunarLander-v2 --exp_name
↪ q2_doubledqn_1 --double_q --seed 1
python run_hw3_dqn_atari.py --env_name LunarLander-v2 --exp_name
↪ q2_doubledqn_2 --double_q --seed 2
python run_hw3_dqn_atari.py --env_name LunarLander-v2 --exp_name
↪ q2_doubledqn_3 --double_q --seed 3
```

You need to submit the `run_logs` for each of the runs from above. In your report, you must make a single graph that averages the performance across three runs for both DQN and double DQN.

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: 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 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 PongNoFrameskip-v4 --exp_name
↪ q3_hparam1
```

```
python run_hw3_dqn.py --env_name PongNoFrameskip-v4 --exp_name
↪ q3_hparam2
```

```
python run_hw3_dqn.py --env_name PongNoFrameskip-v4 --exp_name
↪ q3_hparam3
```

You can replace `PongNoFrameskip-v4` with `LunarLander-v2` if you choose to use that environment.

Note: for Questions 1, you must submit results on the pong environment (even if you chose to test it on the lunar lander environment). For Question 2, you must submit results on the lunar lander environment. For Question 3, you can submit on either pong or lunar lander.

2 Part 2: Actor-Critic

2.1 Introduction

Part 2 of this assignment requires you to modify policy gradients (from hw2) to an actor-critic formulation. Part 2 is relatively shorter than part 1. The actual coding for this assignment will involve less than 20 lines of code. Note however that evaluation may take longer for actor-critic than policy gradient (on half-cheetah) due to the significantly larger number of training steps for the value function.

Recall the policy gradient from hw2:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{it}|s_{it}) \left(\left(\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{it'}, a_{it'}) \right) - V_\phi^\pi \left(s_{it}\right) \right).$$

In this formulation, we estimate the reward to go by taking the sum of rewards to go over each trajectory to estimate the Q function, and subtracting the value function baseline to obtain the advantage

$$A^\pi(s_t, a_t) \approx \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'}) \right) - V_\phi^\pi \left(s_t\right)$$

In practice, the estimated advantage value suffers from high variance. Actor-critic addresses this issue by using a *critic network* to estimate the sum of rewards to go. The most common type of critic network used is a value function, in which case our estimated advantage becomes

$$A^\pi(s_t, a_t) \approx r(s_t, a_t) + \gamma V_\phi^\pi \left(s_{t+1}\right) - V_\phi^\pi \left(s_t\right)$$

In this assignment we will use the same value function network from hw2 as the basis for our critic network. One additional consideration in actor-critic is updating the critic network itself. While we can use Monte Carlo rollouts to estimate the sum of rewards to go for updating the value function network, in practice we fit our value function to the following *target values*:

$$y_t = r(s_t, a_t) + \gamma V^\pi(s_{t+1})$$

we then regress onto these target values via the following regression objective which we can optimize with gradient descent:

$$\min_{\phi} \sum_{i,t} (V_{\phi}^{\pi}(s_{it}) - y_{it})^2$$

In theory, we need to perform this minimization everytime we update our policy, so that our value function matches the behavior of the new policy. In practice however, this operation can be costly, so we may instead just take a few gradient steps at each iteration. Also note that since our target values are based on the old value function, we may need to recompute the targets with the updated value function, in the following fashion:

1. Update targets with current value function
2. Regress onto targets to update value function by taking a few gradient steps
3. Redo steps 1 and 2 several times

In all, the process of fitting the value function critic is an iterative process in which we go back and forth between computing target values and updating the value function to match the target values. Through experimentation, you will see that this iterative process is crucial for training the critic network.

2.2 Implementation

Your code will build off your solutions from hw2. You will need to fill in the TODOS for the following parts of the code. To run the code, go into the `hw3` directory and simply execute `python cs285/scripts/run_hw3_actor_critic.py`.

- In `policies/MLP_policy.py` copy over your policy class for PG, you should note that the AC policy class is in fact the same as the policy class you implemented in the policy gradient homework (except we no longer have a nn_baseline).
- In `agents/ac_agent.py`, finish the `train` function. This function should implement the necessary critic updates, estimate the advantage, and then update the policy. Log the final losses at the end so you can monitor it during training.
- In `agents/ac_agent.py`, finish the `estimate_advantage` function: this function uses the critic network to estimate the advantage values. The advantage values are computed according to

  $$A^{\pi}(s_t, a_t) \approx r(s_t, a_t) + \gamma V_{\phi}^{\pi}(s_{t+1}) - V_{\phi}^{\pi}(s_t)$$

  Note: for terminal timesteps, you must make sure to cut off the reward to go (i.e., set it to zero), in which case we have

  $$A^{\pi}(s_t, a_t) \approx r(s_t, a_t) - V_{\phi}^{\pi}(s_t)$$

- `critics/bootstrapped_continuous_critic.py` complete the TODOS in `build`, `update` and `forward`. In `update`, perform the critic update according to process outlined in the introduction. You must perform

  `self.num_grad_steps_per_target_update * self.num_target_updates`

  number of updates, and recompute the target values every `self.num_grad_steps_per_target_update` number of steps.

2.3 Evaluation

Once you have a working implementation of actor-critic, you should prepare a report. The report should consist of figures for the question below. You should turn in the report as one PDF (same PDF as part 1) and a zip file with your code (same zip file as part 1). If your code requires special instructions or dependencies to run, please include these in a file called `README` inside the zip file.

Question 4: Sanity check with Cartpole Now that you have implemented actor-critic, check that your solution works by running Cartpole-v0.

```
python run_hw3_actor_critic.py --env_name CartPole-v0 -n 100 -b
↪  1000 --exp_name 1_1 -ntu 1 -ngsptu 1
```

In the example above, we alternate between performing one target update and one gradient update step for the critic. As you will see, this probably doesn't work, and you need to increase both the number of target updates and number of gradient updates. Compare the results for the following settings and report which worked best. Do this by plotting all the runs on a single plot and writing your takeaway in the caption.

```
python run_hw3_actor_critic.py --env_name CartPole-v0 -n 100 -b
↪  1000 --exp_name 100_1 -ntu 100 -ngsptu 1
```

```
python run_hw3_actor_critic.py --env_name CartPole-v0 -n 100 -b
↪  1000 --exp_name 1_100 -ntu 1 -ngsptu 100
```

```
python run_hw3_actor_critic.py --env_name CartPole-v0 -n 100 -b
↪  1000 --exp_name 10_10 -ntu 10 -ngsptu 10
```

At the end, the best setting from above should match the policy gradient results from Cartpole in hw2 (200).

Question 5: Run actor-critic with more difficult tasks Use the best setting from the previous question to run InvertedPendulum and HalfCheetah:

```
python run_hw3_actor_critic.py --env_name InvertedPendulum-v2
↪  --ep_len 1000 --discount 0.95 -n 100 -l 2 -s 64 -b 5000 -lr
↪  0.01 --exp_name <>_<> -ntu <> -ngsptu <>
```

```
python run_hw3_actor_critic.py --env_name HalfCheetah-v2
↪  --ep_len 150 --discount 0.90 --scalar_log_freq 1 -n 150 -l 2
↪  -s 32 -b 30000 -eb 1500 -lr 0.02 --exp_name <>_<> -ntu <>
↪  -ngsptu <>
```

Your results should roughly match those of policy gradient. After 150 iterations, your HalfCheetah return should be around 150 and your InveredPendulum return should be around 1000. Your deliverables for this section are plots with the eval returns for both enviornments.

3 Submitting the code and experiment runs

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

- A folder named `run_logs` with all the experiment runs from this assignment. For Q-learning, you need to submit one run for Q1, two runs for Q2, and three runs for Q3. These folders can be copied directly from the `cs285/data` folder. For the actor critic section, likewise submit one folder for each run. **Do not change the names originally assigned to the folders, as specified by `exp_name` in the instructions. Also, video logging is disabled by default in the code, but if you turned it on for debugging, you need to run those again with `--video_log_freq -1`, or else the file size will be too large for submission.**
- The `cs285` folder with all the `.py` files, with the same names and directory structure as the original homework repository (excluding the `cs285/data` folder). Also include any special instructions we need to run in order to produce each of your figures or tables (e.g. "run python myassignment.py -sec2q1" to generate the result for Section 2 Question 1) in the form of a README file. Note that this assignments plotting must be done in a python script, such that running a single script like this can generate the plot.

3.1 Turning it in

Turn in your assignment by the deadline on Gradescope. Upload the zip file with your code to **HW3 Code**, and upload the PDF of your report to **HW3**.