HW10: Reinforcement Learning

Due Dec 14, 2021 by 10:59am **Points** 100 **Submitting** a file upload **File Types** zip **Available** Dec 1, 2021 at 11:59pm - Dec 14, 2021 at 11:15am

This assignment was locked Dec 14, 2021 at 11:15am.

Getting started

The starter code is available here: starter.zip

You can create and activate your virtual environment with the following commands:

```
python3 -m venv /path/to/new/virtual/environment
```

```
source /path/to/new/virtual/environment/bin/activate
```

once you have sourced your environment, you can run the following commands to install the necessary dependencies:

```
pip install --upgrade pip
```

```
pip install torch==1.8.0+cpu torchvision==0.9.0+cpu torchaudio===0.8.0 -f https://download.pytorch.org/whl/torch_stable.html
```

```
pip install gym==0.17.1 numpy==1.19.5
```

You should now have a virtual environment that is fully compatible with the skeleton code. You should set up this virtual environment on an instructional machine (CSL machines) to do your final testing. More detail about python virtual environments is available in the assignment 6 prompt: HW6: Neural Networks .

Q-Learning

For the Q-learning and SARSA portion of HW10, we will be using the environment FrozenLake-v0 from OpenAI gym. This is a discrete environment where the agent can move in the cardinal directions (up, down, left, right), but is not guaranteed to move in the direction it chooses (more in the FrozenLake-v0 doc). The agent gets a reward of 1 when it reaches the tile marked G, and a reward of 0 in all other settings. You can read more about FrozenLake-v0 here: https://gym.openai.com/envs/FrozenLake-v0/

You will not need to change any code outside of the area marked TODO, but you are free to change the hyper-parameters if you want to (but keep the random seed unchanged). The update rule for Q-learning is:

$$Q(s,a) = \begin{cases} Q(s,a) + \alpha(r + \gamma \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a)) & \text{if } !\text{done} \\ Q(s,a) + \alpha(r - Q(s,a)) & \text{if done} \end{cases} \text{ for each sampled } (s,a,r,s',\text{done}) \text{ tuple}$$

In this equation, alpha is the learning rate hyper-parameter, and gamma is the discount factor hyper-parameter.

The agent should act according to an epsilon-greedy policy as defined in the Reinforcement Learning 2 slides.

Epsilon-greedy policy: link

Pseudo Code

```
p = random()

if p < ε:
  pull random action
else:
  pull current-best action
```

HINT: tests.py is worth looking at to gain an understanding of how to use the OpenAI gym env.

For this section, you should submit the files Q_learning.py and Q_TABLE.pkl. The Q_TABLE.pkl will be generated automatically when you run your Q_learning.py.

A sample run of *python Q_learning.py*

```
LAST 100 EPISODE AVERAGE REWARD: 0.01
EPSILON: 0.9018873549605959
LAST 100 EPISODE AVERAGE REWARD: 0.0
EPSILON: 0.8178301806491574
LAST 100 EPISODE AVERAGE REWARD: 0.03
EPSILON: 0.7399663251239436
[[ALAST 100 EPISODE AVERAGE REWARD: 0.03
EPSILON: 0.6695157201807330
LAST 100 EPISODE AVERAGE REWARD: 0.03
EPSILON: 0.6057726059163237
LAST 100 EPISODE AVERAGE REWARD: 0.04
EPSILON: 0.5480982608778811
LAST 100 EPISODE AVERAGE REWARD: 0.03
EPSILON: 0.4959159026170678
LAST 100 EPISODE AVERAGE REWARD: 0.02
EPSILON: 0.4486999944616477
LAST 100 EPISODE AVERAGE REWARD: 0.04
EPSILON: 0.4059802359210585
LAST 100 EPISODE AVERAGE REWARD: 0.12
EPSILON: 0.3673727593461257
LAST 100 EPISODE AVERAGE REWARD: 0.12
EPSILON: 0.3323552449295457
LAST 100 EPISODE AVERAGE REWARD: 0.12
EPSILON: 0.3007124156643856
[[ALAST 100 EPISODE AVERAGE REWARD: 0.09
EPSILON: 0.2720822222195578
LAST 100 EPISODE AVERAGE REWARD: 0.16
EPSILON: 0.2461778870937771
```

The first few print outputs are shown above. Two things are worth pointing out:

1. The value of EPISODE AVERAGE REWARD is obtained from the deque variable *episode_reward_record*. So you need to update this variable in your code by appending the **accumulated** reward obtained in this episode into the deque.

2. EPSILON is decreasing, which means you need to update (decay) the hyperparameter variable EPSILON in your code (e.g. using EPSILON_DECAY).

Note: matching the sample output above is not guaranteed for the correctness of your program, please use the tests.py to test your code.

SARSA

SARSA is very similar to Q-learning, although it differs in the fact that it is on-policy. The name stands for "State, Action, Reward, State, Action," and refers to the tuples that are used by the update rule. You will not need to change any code outside of the area marked TODO, although, you are again welcome to change any hyperparameters you want.

The update rules for SARSA is:
$Q(s,a) = Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$ always for each sampled $(s,a,r,s',a',\text{done})$ tuple
$Q(s',a') = Q(s',a') + \alpha(r' - Q(s',a'))$ if done

Similarly to Q-learning, the agent should act according to an epsilon-greedy policy as defined in the Reinforcement Learning 2 slides.

HINT: tests.py is worth looking at to gain an understanding of how to use the OpenAI gym env.

HINT: make sure that the SARSA update is performed for the last action-reward pair for each episode, because for FrozenLake-v0 this is the only possible time for a non-zero reward to be received. For the last State action pair, the expected cumulative reward is equal to the reward for that state and action, since there will be no further rewards.

For this section, you should submit the files SARSA.py and SARSA_Q_TABLE.pkl

A sample run of *python SARSA.py*

```
LAST 100 EPISODE AVERAGE REWARD: 0.01
EPSILON: 0.9038873549605959
LAST 100 EPISODE AVERAGE REWARD: 0.0
EPSILON: 0.8178301806491574
LAST 100 EPISODE AVERAGE REWARD: 0.03
EPSILON: 0.7399663251239436
LAST 100 EPISODE AVERAGE REWARD: 0.01
EPSILON: 0.6695157201807336
LAST 100 EPISODE AVERAGE REWARD: 0.03
EPSILON: 0.6057726059163237
LAST 100 EPISODE AVERAGE REWARD: 0.09
EPSILON: 0.5480982605578811
LAST 100 EPISODE AVERAGE REWARD: 0.02
EPSILON: 0.4959159020176678
LAST 100 EPISODE AVERAGE REWARD: 0.03
EPSILON: 0.4486999994616477
LAST 100 EPISODE AVERAGE REWARD: 0.08
EPSILON: 0.4059802359210587
LAST 100 EPISODE AVERAGE REWARD: 0.11
EPSILON: 0.3673727293461925?
LAST 100 EPISODE AVERAGE REWARD: 0.1
```

DQN (Extra Credit)

Deep Q-learning has produced some exciting results in the past 5 years. First introduced in 2015 in Mnih et al., DQN allows the application of Q-learning to domains with continuous observation spaces.

For the experience replay buffer, you should add tuples of the form (s,a,r,s',done) . The sample_minibatch function should return a list containing a random sample of MINI_BATCH_SIZE tuples from the experience replay buffer

For the first TODO in the main while loop, you should use the current policy_net policy to determine the best action to take then add the resulting (s,a,r,s',done) tuple you encounter from the environment to the experience replay buffer.

HINT: remember to use "with torch.no_grad():" when you do not need to calculate gradients.

HINT: Mnih et al. has the full pseudo-code, and is useful to look at for reference when implementing DQN.

For the second TODO, you will need to sample MINI_BATCH_SIZE tuples from the experience replay buffer, then calculate the appropriate estimates using the policy_net, and y values using the observed rewards, second states, and the target_policy_net.

OpenAI gym Environment

You will need to use several OpenAI gym functions in order to operate your gym environment for reinforcement learning. As stated in a previous hint, tests.py has a lot of the function calls you need. Several important functions are as follows:

env.step(action)
Given that the environment is in state s, step takes an integer specifying the chosen action, and returns a tuple of the form $(s',r,\text{done},\text{info})$. 'Done' specifies whether or not s' is the final state for that particular episode, and 'info' is unused in this assignment.

env.reset()

Resets the environment to its initial state, and returns that state.

env.action_space.sample()
Samples an integer corresponding to a random choice of action in the environment's action space.

env.action_space.n

In the setting of the environments we will be working with for these assignments, this is an integer corresponding to the number of possible actions in the environment's action space.

You can read more about OpenAI gym here: https://gym.openai.com/docs/ .

Submission Format

You can test your learned policies, by calling python3 ./tests.py. Make sure to test your saved Q-tables using tests.py on the instructional machines with a virtual environment set up as specified above. This is the same program, which we will be using to test your Q-tables and Q-network, so you will have a good idea about how many points you will receive for the automated tests portion of the grade. Your submission should contain the following files:

Required: Q_learning.py, Q_TABLE.pkl, SARSA.py, SARSA_Q_TABLE.pkl
Optional: DQN.py, DQN.mdl, DQN_DATA.pkl

Please submit these files in a zipped folder title <yournetid>.zip , where 'yournetid' is your net ID. Please make sure that there is not a folder inside the zipped folder, and that the submitted files are at the top level of the zipped folder.

The assignment is due December 14th at 10:59 am central time. We are not accepting late submissions for this assignment. Regrades will only be accepted if they are due to an error in tests.py.

Grading

- 60 points (out of 100) will be given if you passed the tests.py (by running python tests.py after you generate the pkls). You can ensure that by testing on your own.

- 40 points will be determined via manual inspection of the code. We will inspect whether you implement the learning algorithms correctly.

- 10 points (extra credits) will be rewarded if your DQN passes the tests.py.

HW10: Reinforcement Learning

Criteria	Ratings			Pts
Q-Learning: Trained Q table 100 episode average score	**30 pts** **Full Marks** 100 episode average >= 0.6	**15 pts** **Partial marks** 100 episode average >= 0.4	**0 pts** **No Marks** 100 episode average <0.4	30 pts
Q-Learning: Implementation correctness	**20 pts** **Full Marks** Correct implementation of Q-learning	**10 pts** **Partial marks** Partially correct implementation of Q-learning	**0 pts** **No Marks** Incorrect implementation of Q-learning	20 pts
SARSA: Trained Q table 100 episode average score	**30 pts** **Full Marks** 100 episode average >= 0.6	**15 pts** **Partial marks** 100 episode average >= 0.4	**0 pts** **No Marks** 100 episode average < 0.4	30 pts
SARSA: Implementation correctness	**20 pts** **Full Marks** Correct implementation of SARSA	**10 pts** **Partial marks** Partially correct implementation of SARSA	**0 pts** **No Marks** Incorrect implementation of SARSA	20 pts
DQN: Trained Q network 100 episode average score	**0 pts** **0 points extra credit**	**0 pts** **10 points extra credit** 100 episode average >= 70		0 pts
				Total Points: 100