Assignment 5: Q-Learning, Double Q-learning, and DQN

Due: Apr. 19th, 2024, 23:59pm

Question 1. Consider a reinforcement-learning system with two states (namely, s_1 and s_2) and two actions (namely, a_1 and a_2). Suppose that a Q-learning trial has been conducted with the agent transitioning through the following state sequence by taking the actions as indicated below:

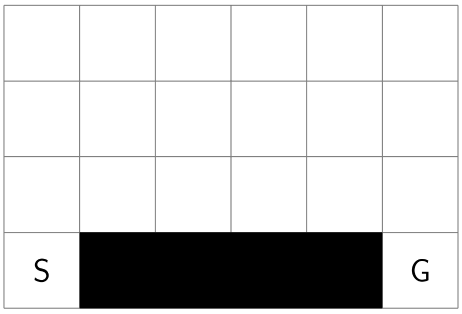
$$s_1 \xrightarrow{a_1,1} s_1 \xrightarrow{a_2,1} s_2 \xrightarrow{a_1,10} s_1 \xrightarrow{a_2,1} s_2 \tag{1}$$

where the number following the action above an arrow indicates the reward received by the agent upon taking that action. For instance, the first arrow implies that the agent at state s_1 takes action a_1, which results in the agent remaining in state s_1 and receiving a reward of 1. Complete the table below to show the values of the Q-function at the end of each action taken by the agent during the trial. For instance, the value of $Q(s_1, a_1)$ is to be entered in the top left empty cell in the table shown. Assume that the initial values of the Q-learning are 0. Use a fixed learning rate of $\alpha = 0.5$ and a discount rate of $\gamma = 0.5$.

Q	s_1	s_2
a_1		
a_2		

Note: Show your detailed calculation steps for obtaining these Q-function values. There are four actions for this trial, so your answer should include four such tables, one for each action taken.

Question 2 (Programming). Consider the grid world shown in the following figure. This is a standard undiscounted, episodic task, with start and goal states, and the usual actions causing movement up, down, right, and left. If the agent takes an action that would move it off the grid, it remains in its current position instead. The reward is -1 on all transitions except those into the black region. Stepping into the black region incurs a reward of -100 and sends the agent instantly back to the start.



1. Implement Q-learning and SARSA on this task respectively with probability for exploration $\epsilon = 0.1$, step size $\alpha = 0.1$, and discount factor $\lambda = 1$. Choose the number of episodes sufficiently large (e.g., 500) so that a stable policy is learned.
   - Plot a figure with two curves that show the "Sum of rewards during the episode" against "Episodes" for Q-learning and SARSA respectively.
   - Plot the learned policy for each method.

   Note that a script is given for your convenience of programming, so you can focus more on coding the Q-function update (search for `TODO` in the code source file to locate where you need to complete the code). To have a smoother plot, the given script performs 100 independent runs.

2. Consider now running both Q-learning and SARSA for 500 training episodes first using $\alpha = 0.1$, $\gamma = 1$, and ϵ-greedy policy for collecting samples where $\epsilon = 0.1$. Then, take the greedy policy resulting from the learned Q-functions of both methods and run another 200 episodes following the obtained policy (that is, stop training after 500 episodes, but test the learned policies from both methods afterward).
    - Plot the cumulative rewards for each episode for both SARSA and Q-Learning.

   Note that
    - Since the greedy policy obtained from the learned Q-function is deterministic, there is no need to perform independent runs to smooth the rewards curves anymore after 500 episodes.
    - It is possible that if you train the agent for not enough number of episodes (say the training episodes = 100), the learned policy could not even lead the agent to the Goal.
3. Change $\epsilon = \frac{1}{10t+1}$, where t is the number of episodes for SARSA. Choose the total number episodes of training to be 500. Work on Task 2 again. Show the policy learned by SARSA and Q-learning. Plot the cumulative rewards for each episode for both SARSA and Q-learning.

In your report, include all the required plots for all 3 tasks. Comment on your simulation results. Submit your codes together with the report.

Question 3 (Programming). Implement Deep Q-learning Network (DQN) and DDQN (Double DQN) to balance an inverted pendulum from the OpenAI Gym environment. In this task, there is only one incomplete code, which is referred to as `dqn.py`, such that upon completion, one should be able to run the `train.py` file and successfully control the inverted pendulum to keep upright after training. All other codes are complete.

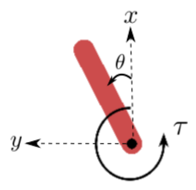


Figure 1: Balancing a pendulum using DQN, DDQN

Tasks:

1. Fill in the blanks in the `dqn.py` file marked with `TODO` to develop the following agents
    a) **DQN** agent, where Q-network takes 1 image and the angular velocity (i.e., the state s), and the toque (i.e., the action a) as input, and outputs the value $Q(s, a) \in \mathbb{R}$ at the current (s, a) pair.
    b) **DQN** agent, where Q-network takes 4 consecutive images (i.e., the state s) and the toque (i.e., the action a) as input, and outputs the value $Q(s, a) \in \mathbb{R}$ at the current (s, a) pair.
    c) **DDQN** agent, where Q-network takes 4 consecutive images as input (i.e., state s), and outputs the values at all actions, that is, outputs a vector $Q(s, \cdot) \in \mathbb{R}^{|\mathcal{A}|}$ and each element of the vector corresponding to $Q(s, a)$ at an action a.
2. Run the `train.py` file to train a model such that it can keep the inverted pendulum upright. Record a video to show that the inverted pendulum is upright under the developed 3 different agents. Submit the completed code and the video.
3. Plot a figure that shows the "mean ep 100 return" against the "used step" as shown in `train.py` for each agent. Play with the parameters in `train.py` file to see how it will affect the return curves. Comment on your exploration/discovery in your accompany report.

4. In the `train.py` file, we use an eight-dimensional vector to denote the discrete action space. Explore more high-dimensional vectors to see whether they can lead to better learning results.

Technical Tips (More in the Appendix 0.1):

- The observation from the environment we provided in the code consists of the lastest four consecutive frames of images and angular velocities, i.e., observation = [image, velocity], where dim(image) = 4x42x42 and dim(velocity)= 1×4.
- Stacking of 4 most recent images and angular velocities is set in `wrappers.py`. Specially, in the `class FrameStack`.
- The higher the index is, the newer the data is. E.g., from index 3 to 0, the corresponding data is from the most recent to the oldest.
- Note that in preprocessing, the RGB channels of the image are combined into one channel in the code. See `class ProcessFrame42` in the `wrappers.py`:

```
img = img[:, :, 0] * 0.299 + img[:, :, 1] * 0.587 + img[:, :, 2] * 0.114

```

- In the setup of the given code, the Q-network takes the observation and the action torque as the input and outputs a Q-value for this state-action pair (please refer to the `forward` function in `dqn.py`). You will need to modify the code to accommodate the different setup described in the problem requirement.

Notes:

- Two example videos are given to you to show the initial output and the desired output for Question 3.
- One should submit a report, all completed codes, and an output video.
- You can modify all the codes freely as long as you can complete all the tasks successfully.
- Refer to the Appendix below for tips on coding environment setup, dependence installation, etc.

Submission

- Make sure to submit one PDF report, one MP4 video, and all the PYTHON codes. Include all your files into a folder and compress it to a zip file. Only submit one zip file for this assignment.
- Naming convention: `Assignment5_YourName.zip`
- **Submission deadline:** Apr. 19th, 2024, 23:59pm.

Appendix

Working Environment:

- We recommend using VScode/PyCharm + Anaconda as your development environment, either in Windows, Mac, or Linux. See Anaconda for the installation of Anaconda.

Installing Dependency:

- You can use anaconda to create a python3 environment by installing the required packages listed in `environment.yml`:

```
cd DQN_DIRECTORY
conda env create -f environment.yml
```

- If some error messages from Anaconda are raised, you could choose to install the required python3 package manually. Run the following command with CMD in Windows or Shell in Linux or MacOS:

```
pip3 install pytorch pygame gym==0.26.1 opencv_python
```

Test Your Built Environment (Dependency):

- When testing the built environment, you could let the code idle by running the following command in the terminal:

```
cd DQN_DIRECTORY
python3 train.py --idling
```

- If there is no error, that means you have installed all dependencies successfully. You can proceed to fill in the blanks in the `dqn.py` file marked with *TODO*.

How to use:

- After completing all blanks in `dqn.py`, you can run `train.py` either in Vscode/PyCharm or in termimal by the following commands

```
cd DQN_DIRECTORY
python3 train.py
```

- We comment off the USE_GPU part in the code, you can enable it if you want to use GPU.
- For more details, please refer to `readme.html` file in DQN folder.

0.1 Tips:

- The state contains 4 consecutive images of 4×42×42, and 4 consecutive angular velocities, returned in the lines below. You can check it using line 3.

```
1 s = env.reset()  # reset environment
2 s_, r, done, infos, _ = env.step(a)   # take a step
3 print(s[0][0].shape)  # images
4 print(s[0][1].shape)  # angular velocities
```

- The code only stops training and begins to animate the pendulum and save the animation if the average return over the last 100 training episodes is above -200 (You can change it to -3000 to see and save the animation without any training). See the code below in train.py,

```
# render
if mean_100_ep_return >= -200:
    evaluate_performance(dqn, disc_actions, episode_length=EPISODE_LENGTH)
```

  where the `evaluate_performance` function will call the OpenAI Gym `env.render()` API to animate the experiment. `evaluate_performance` is set to run 200 steps by default.

- I included the code for saving the animation in train.py as the `save_animation` function, which calls the OpenAI Gym video recorder API (which requires installing the pacakge moviepy):

```
    pip install moviepy
```

  In your python script, import the following,

```
from gym.wrappers.monitoring.video_recorder import VideoRecorder
```

- You can refer to the code is based on this Git repo as commented in the code. To complete the `learn()` function, you can refer to their implementation at here. You can also refer to other code samples, as it is fairly standard to train DNN in PyTorch.

- The convergence and performance of the training can be affected by the exploration scheme (that is, related to the epsilon of ϵ-greedy policy, and how ϵ dimishes), as well as your design convolutional neural networks. One hidden linear layer (in the fully connected layer of CNN) is typically enough. More layers will incur more parameters and slow down the training. If design properly, it should converge within 1 hour of training.

- Returning only angular velocity is set in Pendulum.py:

```
def _get_obs(self):
    theta, thetadot = self.state
    return np.array([thetadot], dtype=np.float32) # return only angular velocity
```