Assignment 3: Q-Learning and Actor-Critic Algorithms

Students : Jules Merigot and Thomas Boudras

Due March 27, 2024

1 Multistep Q-Learning

Consider the N-step variant of Q-learning described in lecture. We learn $Q_{\phi_{k+1}}$ with the following updates:

$$y_{j,t} \leftarrow \left(\sum_{t'=t}^{t+N-1} \gamma^{t'-t} r_{j,t'}\right) + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi_k}\left(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N}\right) \tag{1}$$

$$\phi_{k+1} \leftarrow \arg\min_{\phi\in\Phi} \sum_{j,t} \left(y_{j,t} - Q_\phi(\mathbf{s}_{j,t}, \mathbf{a}_{j,t})\right)^2 \tag{2}$$

In these equations, j indicates an index in the replay buffer of trajectories $\mathcal{D}_k$. We first roll out a batch of B trajectories to update $\mathcal{D}_k$ and compute the target values in (1). We then fit $Q_{\phi_{k+1}}$ to these target values with (2). After estimating $Q_{\phi_{k+1}}$, we can then update the policy through an argmax:

$$\pi_{k+1}\left(\mathbf{a}_t \mid \mathbf{s}_t\right) \leftarrow \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_{\phi_{k+1}}\left(\mathbf{s}_t, \mathbf{a}_t\right) \\ 0 \text{ otherwise.} \end{cases} \tag{3}$$

We repeat the steps in eqs. (1) to (3) K times to improve the policy. In this question, you will analyze some properties of this algorithm, which is summarized in Algorithm 1.

Algorithm 1 Multistep Q-Learning

```
Require: iterations $K$, batch size $B$
1: initialize random policy $\pi_0$, sample $\phi_0 \sim \Phi$
2: for $k = 0 \dots K-1$ do
3:     Update $\mathcal{D}_{k+1}$ with $B$ new rollouts from $\pi_k$
4:     compute targets with (1)
5:     $Q_{\phi_{k+1}} \leftarrow$ update with (2)
6:     $\pi_{k+1} \leftarrow$ update with (3)
7: end for
8: return $\pi_K$
```

1.1 TD-Learning Bias (2 points)

We say an estimator $f_\mathcal{D}$ of f constructed using data $\mathcal{D}$ sampled from process P is *unbiased* when $\mathbb{E}_{\mathcal{D}\sim P}[f_\mathcal{D}(x) - f(x)] = 0$ at each x.

Assume $\hat{Q}$ is a noisy (but unbiased) estimate for Q. Is the Bellman backup $\mathcal{B}\hat{Q} = r(s,a) + \gamma \max_{a'} \hat{Q}(s',a')$ an unbiased estimate of $\mathcal{B}Q$?

☐ Yes

[✓ No]

1.2 Tabular Learning (6 points total)

At each iteration of the algorithm above after the update from eq. (2), Q_{ϕ_k} can be viewed as an estimate of the true optimal Q^*. Consider the following statements:

I. $Q_{\phi_{k+1}}$ is an unbiased estimate of the Q function of the last policy, Q^{π_k}.

II. As $k \to \infty$ for some fixed B, Q_{ϕ_k} is an unbiased estimate of Q^*, i.e., $\lim_{k\to\infty} \mathbb{E}\left[Q_{\phi_k}(s,a) - Q^*(s,a)\right] = 0$.

III. In the limit of infinite iterations and data we recover the optimal Q^*, i.e., $\lim_{k,B\to\infty} \mathbb{E}\left[\|Q_{\phi_k} - Q^*\|_\infty\right] = 0$.

We make the additional assumptions:

- The state and action spaces are finite.
- Every batch contains at least one experience for each action taken in each state.
- In the tabular setting, Q_{ϕ_k} can express any function, i.e., $\{Q_{\phi_k} : \phi \in \Phi\} = \mathbb{R}^{S\times A}$.

When updating the buffer $\mathcal{D}_k$ with B new trajectories in line 3 of Algorithm 1, we say:

- When learning *on-policy*, $\mathcal{D}_k$ is set to contain only the set of B new rollouts of π (so $|\mathcal{D}_k| = B$). Thus, we only train on rollouts from the current policy.
- When learning *off-policy*, we use a fixed dataset $\mathcal{D}_k = \mathcal{D}$ of B trajectories from another policy π'.

Indicate which of the statements **I-III** always hold in the following cases. No justification is required.

	I.	**II.**	**III.**
1. $N = 1$ and …			
(a) on-policy in tabular setting	☐	☐	■
(b) off-policy in tabular setting	☐	■	■
2. $N > 1$ and …			
(a) on-policy in tabular setting	☐	☐	■
(b) off-policy in tabular setting	☐	■	■
3. In the limit as $N \to \infty$ (no bootstrapping) …			
(a) on-policy in tabular setting	☐	☐	■
(b) off-policy in tabular setting	☐	☐	■

1.3 Variance of Q Estimate (2 points)

Which of the three cases ($N = 1$, $N > 1$, $N \to \infty$) would you expect to have the highest-variance estimate of Q for fixed dataset size B in the limit of infinite iterations k? Lowest-variance?

Highest variance:

☐ $N = 1$

■ $N > 1$

☐ $N \to \infty$

Lowest variance:

■ $N = 1$

☐ $N > 1$

☐ $N \to \infty$

1.4 Function Approximation (2 points)

Now say we want to represent Q via function approximation rather than with a tabular representation. Assume that for any deterministic policy π (including the optimal policy π^*), function approximation can represent the true Q^π exactly. Which of the following statements are true?

■ When $N = 1$, $Q_{\phi_{k+1}}$ is an unbiased estimate of the Q-function of the last policy Q^{π_k}.

■ When $N = 1$ and in the limit as $B \to \infty$, $k \to \infty$, Q_{ϕ_k} converges to Q^*.

■ When $N > 1$ (but finite) and in the limit as $B \to \infty$, $k \to \infty$, Q_{ϕ_k} converges to Q^*.

☐ When $N \to \infty$ and in the limit as $B \to \infty$, $k \to \infty$, Q_{ϕ_k} converges to Q^*.

1.5 Multistep Importance Sampling (5 points)

We can use importance sampling to make the N-step update work off-policy with trajectories drawn from an arbitrary policy. Rewrite (2) to correctly approximate a Q_{ϕ_k} that improves upon π when it is trained on data $\mathcal{D}$ consisting of B rollouts of some other policy $\pi'(\mathbf{a}_t \mid \mathbf{s}_t)$.

Do we need to change (2) when $N = 1$? What about as $N \to \infty$?

You may assume that π' always assigns positive mass to each action. [Hint: re-weight each term in the sum using a ratio of likelihoods from the policies π and π'.]

ANSWER: To adapt (2) for off-policy learning using importance sampling, we need to incorporate importance sampling weights into the sum. The importance sampling weight for each term in the sum adjusts for the difference in the action-selection probabilities between the target policy π and the behavior policy π' under which the data was collected. This weight is the ratio of the probabilities of the trajectory under both policies.

$$\phi_{k+1} \leftarrow \arg\min_{\phi \in \Phi} \sum_{j,t} \rho_{j,t} \cdot \big(y_{j,t} - Q_\phi(\mathbf{s}_{j,t}, \mathbf{a}_{j,t})\big)^2$$

$$\text{where (using the hint)} \quad \rho_{j,t} = \frac{\pi(\mathbf{a}_{j,t}|\mathbf{s}_{j,t})}{\pi'(\mathbf{a}_{j,t}|\mathbf{s}_{j,t})}$$

Where $\rho_{i,j}$ represents the importance sampling weight for the transition at time t in trajectory j. The weight $\rho_{i,j}$ compensates for the difference in action-selection probabilities between the policies.

Therefore, equation (2) can be rewritten as the following:

$$\phi_{k+1} \leftarrow \arg\min_{\phi \in \Phi} \sum_{j,t} \left(\frac{\pi(\mathbf{a}_{j,t}|\mathbf{s}_{j,t})}{\pi'(\mathbf{a}_{j,t}|\mathbf{s}_{j,t})}\right) \cdot \big(y_{j,t} - Q_\phi(\mathbf{s}_{j,t}, \mathbf{a}_{j,t})\big)^2$$

When $N = 1$, the multistep Q-learning update reduces to a standard one-step Q-learning update. The importance sampling correction is still required if the behavior policy π' is different from the target policy π to account for the discrepancy between the policies. However, if $N = 1$ and $\pi' = pi$, then no importance sampling correction is necessary.

AS $N \to \infty$, we would be attempting to approximate the return over an very large horizon, which is equivalent to calculating the expected return of following the policy π from state $s_{j,t}$ almost indefinitely. In this case, the importance sampling correction would involve a product of likelihood ratios, and would therefore change equation (2).

2 Deep Q-Learning

2.1 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.

2.2 File overview

The starter code for this assignment can be found at

`https://github.com/berkeleydeeprlcourse/homework_fall2023/tree/main/hw3`

You will implement a DQN agent in `cs285/agents/dqn_agent.py` and `cs285/scripts/run_hw3_dqn.py`. In addition to those two files, you should start by reading the following files thoroughly:

- `cs285/env_configs/dqn_basic.py`: builds networks and generates configuration for the basic DQN problems (cartpole, lunar lander).
- `cs285/env_configs/dqn_atari.py`: builds networks and generates configuration for the Atari DQN problems.
- `cs285/infrastructure/replay_buffer.py`: implementation of replay buffer. You don't need to know how the memory efficient replay buffer works, but you should try to understand what each method does (particularly the difference between `insert`, which is called after a frame, and `on_reset`, which inserts the first observation from a trajectory) and how it differs from the regular replay buffer.
- `cs285/infrastructure/atari_wrappers.py`: contains some wrappers specific to the Atari environments. These wrappers can be key to getting challenging Atari environments to work!

There are two new package requirements (`gym[atari]` and `pip install gym[accept-rom-license]`) beyond what was used in the first two assignments; make sure to install these with `pip install -r requirements.txt` if you're re-using your Python environment from last assignment.

2.3 Implementation

The first phase of the assignment is to implement a working version of Q-learning, with some extra bells and whistles like double DQN. Our code will work with both state-based environments, where our input is a low-dimensional list of numbers (like Cartpole), but we'll also support learning directly from pixels!

In addition to the double Q-learning trick (which you'll implement later), we have a few other tricks implemented to stabilize performance. You don't have to do anything to enable these, but you should look at the implementations and think about why they work.

- **Exploration scheduling for ϵ-greedy actor.** This starts ϵ at a high value, close to random sampling, and decays it to a small value during training.
- **Learning rate scsheduling.** Decay the learning rate from a high initial value to a lower value at the end of training.
- **Gradient clipping.** If the gradient norm is larger than a threshold, scale the gradients down so that the norm is equal to the threshold.
- **Atari wrappers.**
  - **Frame-skip.** Keep the same constant action for 4 steps.
  - **Frame-stack.** Stack the last 4 frames to use as the input.
  - **Grayscale.** Use grayscale images.

2.4 Basic Q-Learning

Implement the basic DQN algorithm. You'll implement an update for the Q-network, a target network, and

What you'll need to do:

- Implement a DQN critic update in `update_critic` by filling in the unimplemented sections (marked with TODO(student)).
- Implement ϵ-greedy sampling in `get_action`
- Implement the TODOs in `run_hw3_dqn.py`.
- Call all of the required updates, and update the target critic if necessary, in `update`.

Testing this section:

- Debug your DQN implementation on `CartPole-v1` with `experiments/dqn/cartpole.yaml`. It should reach reward of nearly 500 within a few thousand steps.

Deliverables:

- Submit your logs of `CartPole-v1`, and a plot with environment steps on the x-axis and eval return on the y-axis.

Timestamp	Step	Reward
1711377184.4595	0	40.400001525878906
1711377185.3272	10000	34.099998474121094
1711377186.3292	20000	45.5
1711377205.1629	30000	165.8000030517578
1711377223.3871	40000	97.5
1711377240.3521	50000	265.5
1711377257.6961	60000	131.89999389648438
1711377274.7014	70000	94.4000015258789
1711377291.5586	80000	106.5999984741211
1711377309.2222	90000	115.0999984741211
1711377337.1219	100000	113.30000305175781
1711377365.5302	110000	98.5999984741211
1711377393.8967	120000	205.1999969482422
1711377422.2310	130000	500.0
1711377451.2274	140000	500.0
1711377479.8463	150000	500.0
1711377509.1360	160000	500.0
1711377537.8622	170000	500.0
1711377566.7787	180000	500.0
1711377595.0650	190000	500.0
1711377623.6252	200000	425.79998779296875
1711377651.5190	210000	500.0
1711377684.7971	220000	56.29999923706055
1711377739.3627	230000	500.0
1711377792.4230	240000	500.0
1711377848.8484	250000	433.70001220703125
1711377898.9886	260000	500.0
1711377957.5315	270000	500.0
1711378007.9223	280000	400.29998779296875
1711378043.4703	290000	500.0

Table 1: Log of `CartPole-v1` run.

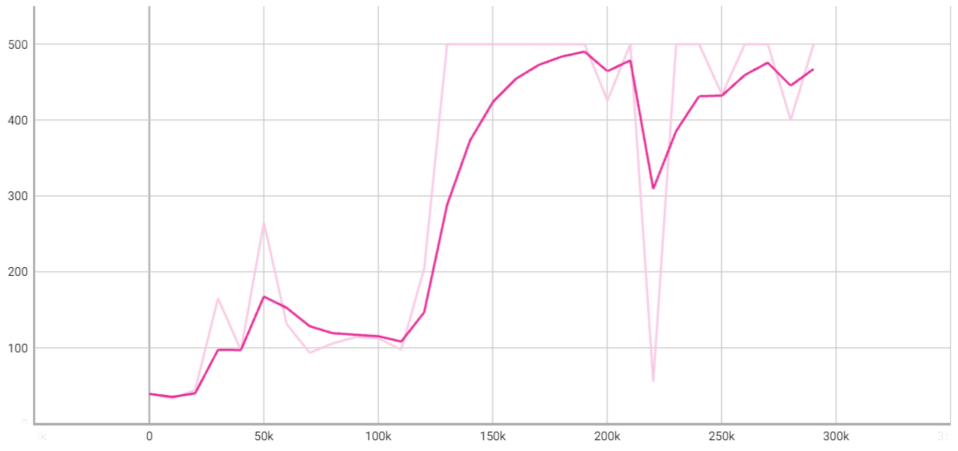

Figure 1: Plot of the `CartPole-v1` run with environment steps on the x-axis and the eval return on the y-axis.

- Run DQN with three different seeds on `LunarLander-v2`: **Your code may not reach high return (200) on Lunar Lander yet; this is okay!**

```
python cs285/scripts/run_hw3_dqn.py -cfg experiments/dqn/lunarlander.yaml --seed 1
python cs285/scripts/run_hw3_dqn.py -cfg experiments/dqn/lunarlander.yaml --seed 2
python cs285/scripts/run_hw3_dqn.py -cfg experiments/dqn/lunarlander.yaml --seed 3
```

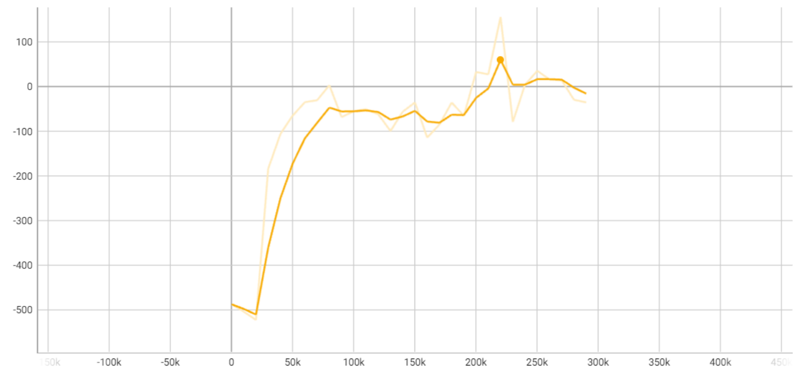

Figure 2: Plot of the `LunarLander-v2` with `seed 1` run with environment steps on the x-axis and the eval return on the y-axis.

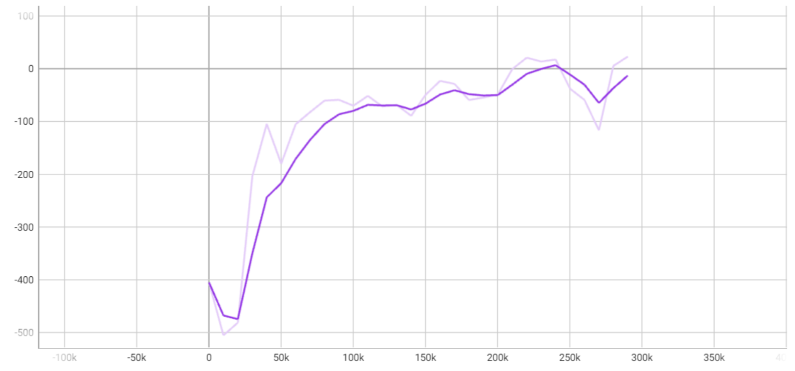

Figure 3: Plot of the `LunarLander-v2` with `seed 2` run with environment steps on the x-axis and the eval return on the y-axis.

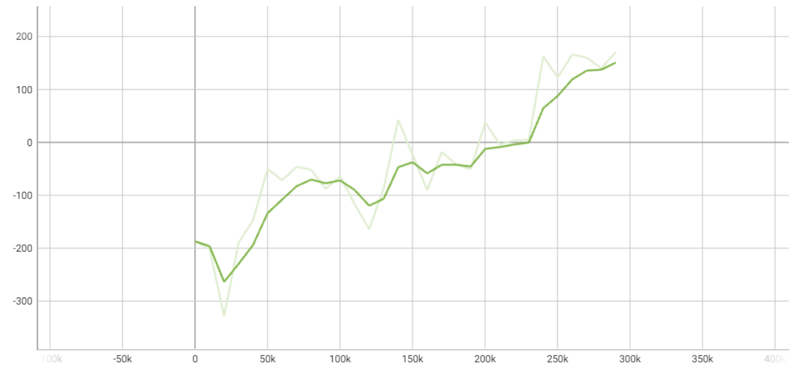

Figure 4: Plot of the `LunarLander-v2` with `seed 3` run with environment steps on the x-axis and the eval return on the y-axis.

- Run DQN on `CartPole-v1`, but change the `learning rate` to 0.05 (you can change this in the YAML config file). What happens to (a) the predicted Q-values, and (b) the critic error? Can you relate this to any topics from class or the analysis section of this homework?

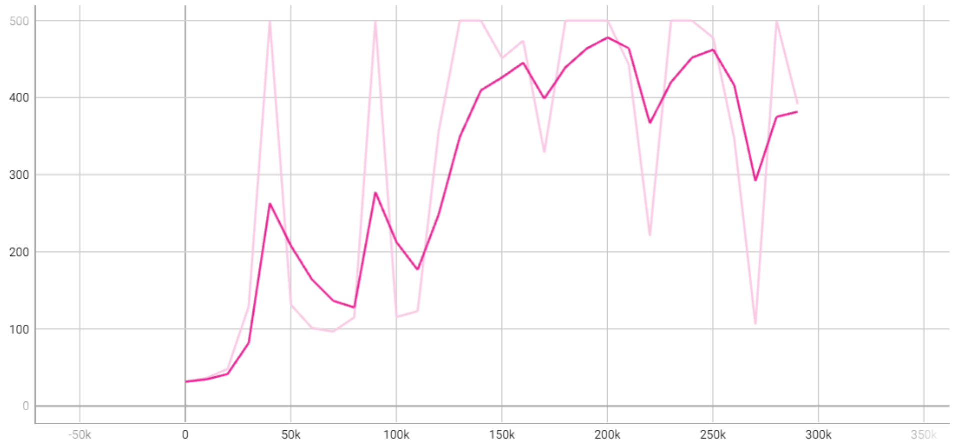

Figure 5: Plot of the `CartPole-v1` with `learning rate` changed to 0.05, run with environment steps on the x-axis and the eval return on the y-axis.

ANSWER: The predicted Q-values seem to increase in a similar manner in both cases as can be seen in the plot below, but the critic error with the new learning rate seems to be more significant at first but eventually diminishes compared to the original learning rate.

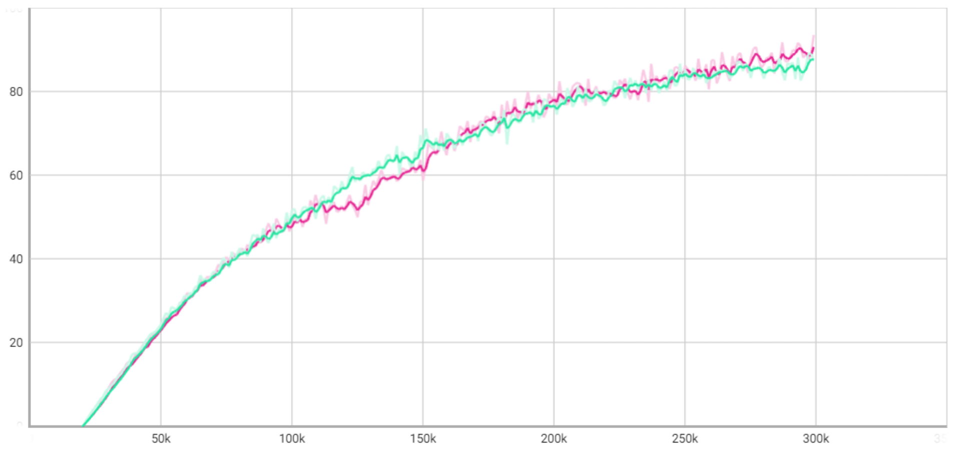

Figure 6: Q-values plot of the `CartPole-v1` with `learning rate` changed to 0.05, run with environment steps on the x-axis and the Q-values on the y-axis.

2.5 Double Q-Learning

Let's try to stabilize learning. The double-Q trick avoids overestimation bias in the critic update by using two different networks to *select* the next action a' and to *estimate* its value:

$$a' = \arg\max_{a'} Q_\phi(s', a')$$

$$Q_{\text{target}} = r + \gamma(1 - d_t)Q_{\phi'}(s', a').$$

In our case, we'll keep using the target network $Q_{\phi'}$ to estimate the action's value, but we'll select the action using Q_ϕ (the online Q network).

Implement this functionality in `dqn_agent.py`.

Deliverables:

- Run three more seeds of the lunar lander problem:

```
python cs285/scripts/run_hw3_dqn.py -cfg experiments/dqn/lunarlander_doubleq.yaml --
    seed 1
python cs285/scripts/run_hw3_dqn.py -cfg experiments/dqn/lunarlander_doubleq.yaml --
    seed 2
python cs285/scripts/run_hw3_dqn.py -cfg experiments/dqn/lunarlander_doubleq.yaml --
    seed 3
```

  You should expect a return of **200** by the end of training, and it should be fairly stable compared to your policy gradient methods from HW2.

  Plot returns from these three seeds in red, and the "vanilla" DQN results in blue, on the same set of axes. Compare the two, and describe in your own words what might cause this difference.

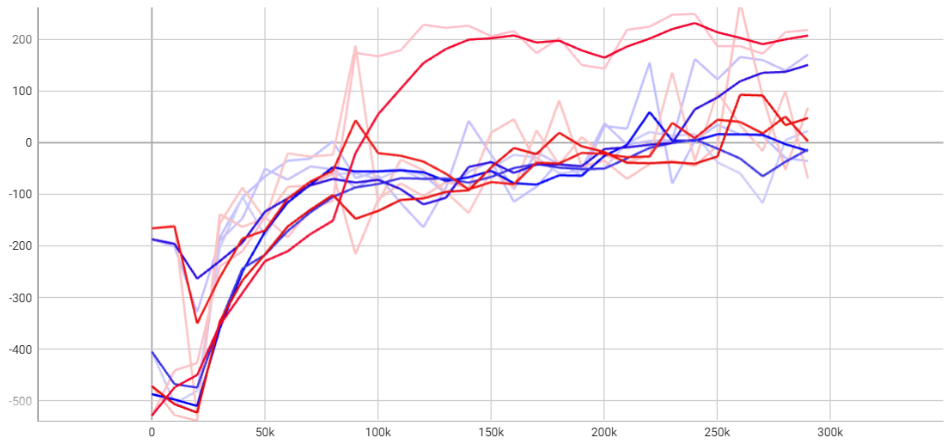

Figure 7: Plot of the "vanilla" DQN `LunarLander-v2` in blue vs `LunarLander-v2` with the double-Q trick in red, run with environment steps on the x-axis and the Q-values on the y-axis.

  ANSWER: The double-Q trick that avoids overestimation stabilizes the learning of the network and therefore allows for better learning overall, allowing for a high return and a consistent reward of 200 by the end of training as can be seen by the plot in red below.

- Run your DQN implementation on the `MsPacman-v0` problem. Our default configuration will use double-Q learning by default. You are welcome to tune hyperparameters to get it to work better, but the default parameters should work (so if they don't, you likely have a bug in your implementation). Your implementation should receive a score of around **1500** by the end of training (1 million steps. **This problem will take about 3 hours with a GPU, or 6 hours without, so start early!**

```
python cs285/scripts/run_hw3_dqn.py -cfg experiments/dqn/mspacman.yaml
```

- Plot the average training return (`train_return`) and eval return (`eval_return`) on the same axes. You may notice that they look very different early in training! Explain the difference.

2.6 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. Create four config files in `experiments/dqn/hyperparameters`, and look in `cs285/env_configs/basic_dqn_config.py` to see which hyperparameters you're able to change. You can use any of the base YAML files as a reference.

Hyperparameter options could include:

- Learning rate
- Network architecture
- Exploration schedule (or, if you'd like, you can implement an alternative to ϵ-greedy)

NOTE:

Dear Therese and Mr. Benhamou,

We would like to apologize for not being able to finish this homework, unfortunately this is as far as we were able to get. This is due to a combination of not having access to a GPU for this homework and some other difficulties, but primarily because of the deadline conflicting with the rest of our courses' final projects and exams. We understand that due to the first homework being pushed back a week, it caused a domino effect with the rest of the assignments that led to this third and final homework being pushed back to after the final Kaggle Connect X project. However, due to the mounting amount of demanding work for our five other courses, it just simply was not feasible for us to dedicate our full attention to completing this homework in its full length. We were able to complete the theoretical questions as well as the first plots of question 2, but we were held back from being able to tackle the rest.

We spent a lot of time on our Kaggle Connect X project, and our other final projects piled up quickly, so we were unfortunately forced to sideline this homework in the meantime in order to complete them. This led us to this moment where we simply no longer have enough time remaining to run the code without access to a GPU, but we have done the work that we can so as to show our commitment to fulfilling this course. The associated code for these plots and their logs (in the `data/` file) will be submitted alongside this report. I hope you can understand this difficult situation we find ourselves in, we know there are other students in the class that will be doing the same due to our shared courses.

We greatly appreciate what we have learned in this course, and the fun and competitive manner in which we were able to apply it in the Kaggle project was very enjoyable, so we would like to thank you for this and for your teaching.

Best,
Jules et Thomas

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 `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. Video logging is disabled by default in the code, but if you turned it on for debugging, you will 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 `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.

As an example, the unzipped version of your submission should result in the following file structure. **Make sure that the submit.zip file is below 15MB and that they include the prefix `q1_`, `q2_`, `q3_`, etc.**

```
submit.zip
├─ data
│  ├─ hw3_dqn_...
│  │  └─ events.out.tfevents.1567529456.e3a096ac8ff4
│  ├─ hw3_sac_...
│  │  └─ events.out.tfevents.1567529456.e3a096ac8ff4
│  └─ ...
├─ cs285
│  ├─ agents
│  │  ├─ soft_actor_critic.py
│  │  └─ dqn_agent.py
│  └─ ...
├─ README.md
└─ ...
```

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 . -x "*.DS_Store"` from your terminal from within the top-level `cs285` directory.

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**.

SAC-related questions. We wanted to address some of the common questions that have been asked regarding Question 6 of the HW. The full algorithm for SAC is summarized below, the equations listed in this paper will be helpful for you: `https://arxiv.org/pdf/1812.05905`. Some definitions that will be useful:

1. What is alpha and how to update it: Alpha is the entropy regularization coefficient denoting how much exploration to add to the policy. You should update based on Eq. 18 in Section 5 in the above paper as follows:
$$J(\alpha) = \mathop{\mathbb{E}}_{a_t \sim \pi_t}[-\alpha \log_{\pi_t}(a_t|s_t) - \alpha\bar{\mathcal{H}}].$$

2. Target entropy is the negative of the action space dimension that is used to update the alpha term.

3. SquashedNorm: This is a function that takes in mean and std as in previous homeworks, and will give you a distribution that you can sample your action from.

4. To update the critic, refer to how to update Q-function parameters in Equation 6 of the paper above as follows:
$$J_Q(\theta) = Q_\theta(s_t, a_t) - (r(s_t, a_t) + \gamma(Q_{\bar{\theta}}(s_{t+1}, a_{t+1}) - \alpha \log(\pi_\phi(a_{t+1}, s_{t+1}))))$$

5. To update the policy, follow Equation 18:
$$J(\alpha) = E_{a_t \sim \pi_t}[-\alpha \log \pi_t(a_t|s_t) - \alpha\bar{\mathcal{H}}]$$

6. You don't need to alter any parameters from the SAC run commands. The correct implementation should work with the provided default parameters.