CS 234 Winter 2022: Assignment #2

Due date:
January 28, 2022 at 6 PM (18:00) PST

These questions require thought, but do not require long answers. Please be as concise as possible.

We encourage students to discuss in groups for assignments. We ask that you abide by the university Honor Code and that of the Computer Science department. If you have discussed the problems with others, please include a statement saying who you discussed problems with. Failure to follow these instructions will be reported to the Office of Community Standards. We reserve the right to run a fraud-detection software on your code. Please refer to website, Academic Collaboration and Misconduct section for details about collaboration policy.
Please review any additional instructions posted on the assignment page. When you are ready to submit, please follow the instructions on the course website. **Make sure you test your code using the provided commands and do not edit outside of the marked areas.** Also note that **all submissions must be typeset. No handwritten submissions will be accepted.** We have released a LaTeX template for your convenience, but you may use any typesetting program of your choice, including, e.g., Microsoft Word.

You'll need to download the starter code and fill the appropriate functions following the instructions from the handout and the code's documentation. Training DeepMind's network on `Pong` takes roughly **12 hours on GPU**, so **please start early!** (Only a completed run will recieve full credit) We will give you access to an Azure GPU cluster. You'll find the setup instructions on the course assignment page.

Introduction

In this assignment, we will first consider a few theoretical questions about Q-learning, and then we will implement deep Q-learning, following DeepMind's paper ([4] and [5]) that learns to play Atari games from raw pixels. The purpose is to demonstrate the effectiveness of deep neural networks as well as some of the techniques used in practice to stabilize training and achieve better performance. In the process, you'll become familiar with PyTorch. We will train our networks on the `Pong-v0` environment from OpenAI gym, but the code can easily be applied to any other environment.

In Pong, one player scores if the ball passes by the other player. An episode is over when one of the players reaches 21 points. Thus, the total return of an episode is between -21 (lost every point) and $+21$ (won every point). Our agent plays against a decent hard-coded AI player. Average human performance is -3 (reported in [5]). In this assignment, you will train an AI agent with super-human performance, reaching at least $+10$ (hopefully more!).

1 Distributions induced by a policy (13 pts)

In this problem, we'll work with an infinite-horizon MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$ and consider stochastic policies of the form $\pi : \mathcal{S} \to \Delta(\mathcal{A})$[1]. Additionally, we'll assume that $\mathcal{M}$ has a single, fixed starting state $s_0 \in \mathcal{S}$ for simplicity.

(a) (**written**, 3 pts) Consider a fixed stochastic policy and imagine running several rollouts of this policy within the environment. Naturally, depending on the stochasticity of the MDP $\mathcal{M}$ and the policy itself, some trajectories are more likely than others. Write down an expression for $\rho^\pi(\tau)$, the probability of sampling a trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots)$ from running π in $\mathcal{M}$. To put this distribution in context, recall that $V^\pi(s_0) = \mathbb{E}_{\tau \sim \rho^\pi}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \,|\, s_0\right]$.

(b) (**written**, 5 pts) Just as ρ^π captures the distribution over trajectories induced by π, we can also examine the distribution over states induced by π. In particular, define the *discounted, stationary state distribution* of a policy π as

$$d^\pi(s) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t p(s_t = s),$$

where $p(s_t = s)$ denotes the probability of being in state s at timestep t while following policy π; your answer to the previous part should help you reason about how you might compute this value. Consider an arbitrary function $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Prove the following identity:

$$\mathbb{E}_{\tau \sim \rho^\pi}\left[\sum_{t=0}^{\infty} \gamma^t f(s_t, a_t)\right] = \frac{1}{(1-\gamma)} \mathbb{E}_{s \sim d^\pi}\left[\mathbb{E}_{a \sim \pi(s)}\left[f(s,a)\right]\right].$$

Hint: You may find it helpful to first consider how things work out for $f(s,a) = 1, \forall (s,a) \in \mathcal{S} \times \mathcal{A}$.
Hint: What is $p(s_t = s)$?

(c) (**written**, 5 pts) For any policy π, we define the following function

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s).$$

Prove the following statement holds for all policies π, π':

$$V^\pi(s_0) - V^{\pi'}(s_0) = \frac{1}{(1-\gamma)} \mathbb{E}_{s \sim d^\pi}\left[\mathbb{E}_{a \sim \pi(s)}\left[A^{\pi'}(s,a)\right]\right].$$

Hint: Try adding and subtracting a term that will let you bring $A^\pi(s,a)$ into the equation

2 SARSA vs. Q-Learning (5 pts)

In the grid world below (1), the agent can move left, right, up or down. Moving onto a green square yields reward -1, while moving onto a red square yields reward -1000 and terminates the episode. The episode also terminates if the agent reaches state 35, the goal state. The agent starts at state 29.
Suppose we train two algorithms, Q-learning and SARSA, with **fixed** $\epsilon = 0.1$ and $\gamma = 1$, and learn two policies: Policy A and Policy B.
Policy A takes the path

$$29 - 22 - 15 - 8 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 14 - 21 - 28 - 35$$

[1] For a finite set $\mathcal{X}$, $\Delta(\mathcal{X})$ refers to the set of categorical distributions with support on $\mathcal{X}$ or, equivalently, the $\Delta^{|\mathcal{X}|-1}$ probability simplex.

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35

Figure 1: Grid World.

and Policy B takes path

$$29 - 22 - 15 - 16 - 17 - 18 - 19 - 20 - 21 - 28 - 35$$

Which policy is learned by Q-learning, and which policy is learned by SARSA? Explain how you arrived at your answer in just a couple of sentences.

3 Test Environment (6 pts)

Before running our code on Pong, it is crucial to test our code on a test environment. In this problem, you will reason about optimality in the provided test environment by hand; later, to sanity-check your code, you will verify that your implementation is able to achieve this optimality. You should be able to run your models on CPU in no more than a few minutes on the following environment:

- 4 states: $0, 1, 2, 3$
- 5 actions: $0, 1, 2, 3, 4$. Action $0 \leq i \leq 3$ goes to state i, while action 4 makes the agent stay in the same state.
- Rewards: Going to state i from states 0, 1, and 3 gives a reward $R(i)$, where $R(0) = 0.1, R(1) = -0.3, R(2) = 0.0, R(3) = -0.2$. If we start in state 2, then the rewards defind above are multiplied by -10. See Table 1 for the full transition and reward structure.
- One episode lasts 5 time steps (for a total of 5 actions) and always starts in state 0 (no rewards at the initial state).

State (s)	Action (a)	Next State (s')	Reward (R)
0	0	0	0.1
0	1	1	-0.3
0	2	2	0.0
0	3	3	-0.2
0	4	0	0.1
1	0	0	0.1
1	1	1	-0.3
1	2	2	0.0
1	3	3	-0.2
1	4	1	-0.3
2	0	0	-1.0
2	1	1	3.0
2	2	2	0.0
2	3	3	2.0
2	4	2	0.0
3	0	0	0.1
3	1	1	-0.3
3	2	2	0.0
3	3	3	-0.2
3	4	3	-0.2

Table 1: Transition table for the Test Environment

An example of a trajectory (or episode) in the test environment is shown in Figure 2, and the trajectory can be represented in terms of s_t, a_t, R_t as: $s_0 = 0, a_0 = 1, R_0 = -0.3, s_1 = 1, a_1 = 2, R_1 = 0.0, s_2 = 2, a_2 = 4, R_2 = 0.0, s_3 = 2, a_3 = 3, R_3 = 2.0, s_4 = 3, a_4 = 0, R_4 = 0.1, s_5 = 0$.

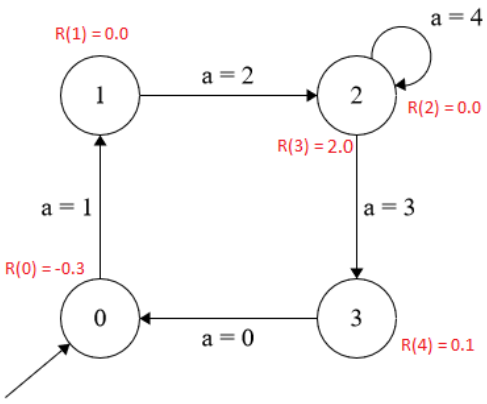


Figure 2: Example of a trajectory in the Test Environment

What is the maximum sum of rewards that can be achieved in a single trajectory in the test environment, assuming $\gamma = 1$? Show first that this value is attainable in a single trajectory, and then briefly argue why no other trajectory can achieve greater cumulative reward.

4 Tabular Q-Learning (3 pts)

If the state and action spaces are sufficiently small, we can simply maintain a table containing the value of $Q(s, a)$, an estimate of $Q^*(s, a)$, for every (s, a) pair. In this *tabular setting*, given an experience sample (s, a, r, s'), the update rule is

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a) \right) \tag{1}$$

where $\alpha > 0$ is the learning rate, $\gamma \in [0, 1)$ the discount factor.

ϵ-Greedy Exploration Strategy For exploration, we use an ϵ-greedy strategy. This means that with probability ϵ, an action is chosen uniformly at random from $\mathcal{A}$, and with probability $1 - \epsilon$, the greedy action (i.e., $\arg\max_{a \in \mathcal{A}} Q(s, a)$) is chosen.

(a) (**coding**, 3 pts) Implement the `get_action` and `update` functions in `q4_schedule.py`. Test your implementation by running `python q4_schedule.py`.

5 Linear Approximation (23 pts)

Due to the scale of Atari environments, we cannot reasonably learn and store a Q value for each state-action tuple. We will instead represent our Q values as a parametric function $Q_{\mathbf{w}}(s, a)$ where $\mathbf{w} \in \mathbb{R}^p$ are the parameters of the function (typically the weights and biases of a linear function or a neural network). In this *approximation setting*, the update rule becomes

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left(r + \gamma \max_{a' \in \mathcal{A}} Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a) \right) \nabla_{\mathbf{w}} Q_{\mathbf{w}}(s, a) \tag{2}$$

where (s, a, r, s') is a transition from the MDP.

(a) (**written**, 5 pts) Let $Q_{\mathbf{w}}(s, a) = \mathbf{w}^\top \delta(s, a)$ be a linear approximation for the Q function, where $\mathbf{w} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ and $\delta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ with

$$[\delta(s, a)]_{s', a'} = \begin{cases} 1 & \text{if } s' = s, a' = a \\ 0 & \text{otherwise} \end{cases}$$

In other words, δ is a function which maps state-action pairs to one-hot encoded vectors. Compute $\nabla_{\mathbf{w}} Q_{\mathbf{w}}(s, a)$ and write the update rule for $\mathbf{w}$. Show that equations (1) and (2) are equal when this linear approximation is used.

(b) (**coding**, 15 pts) We will now implement linear approximation in PyTorch. This question will set up the pipeline for the remainder of the assignment. You'll need to implement the following functions in `q5_linear_torch.py` (please read through `q5_linear_torch.py`):

- `initialize_models`
- `get_q_values`
- `update_target`
- `calc_loss`
- `add_optimizer`

Test your code by running `python q5_linear_torch.py` **locally on CPU**. This will run linear approximation with PyTorch on the test environment from Problem 3. Running this implementation should only take a minute.

(c) (**written**, 3 pts) Do you reach the optimal achievable reward on the test environment? Attach the plot `scores.png` from the directory `results/q5_linear` to your writeup.

6 Implementing DeepMind's DQN (13 pts)

(a) (**coding** 10pts) Implement the deep Q-network as described in [4] by implementing `initialize_models` and `get_q_values` in `q6_nature_torch.py`. The rest of the code inherits from what you wrote for linear approximation. Test your implementation **locally on CPU** on the test environment by running `python q6_nature_torch.py`. Running this implementation should only take a minute or two.

(b) (**written** 3 pts) Attach the plot of scores, `scores.png`, from the directory `results/q6_nature` to your writeup. Compare this model with linear approximation. How do the final performances compare? How about the training time?

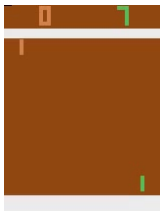
7 DQN on Atari (21 pts)

Reminder: Please remember to kill your VM instances when you are done using them!!

The Atari environment from OpenAI gym returns observations (or original frames) of size $(210 \times 160 \times 3)$, the last dimension corresponds to the RGB channels filled with values between 0 and 255 (`uint8`). Following DeepMind's paper [4], we will apply some preprocessing to the observations:

- Single frame encoding: To encode a single frame, we take the maximum value for each pixel color value over the frame being encoded and the previous frame. In other words, we return a pixel-wise max-pooling of the last 2 observations.
- Dimensionality reduction: Convert the encoded frame to grey scale, and rescale it to $(80 \times 80 \times 1)$. (See Figure 3)

The above preprocessing is applied to the 4 most recent observations and these encoded frames are stacked together to produce the input (of shape $(80 \times 80 \times 4)$) to the Q-function. Also, for each time we decide on an action, we perform that action for 4 time steps. This reduces the frequency of decisions without impacting the performance too much. You can refer to the *Methods Section* of [4] for more details.



(a) Original input $(210 \times 160 \times 3)$ with RGB colors

(b) After preprocessing in grey scale of shape $(80 \times 80 \times 1)$

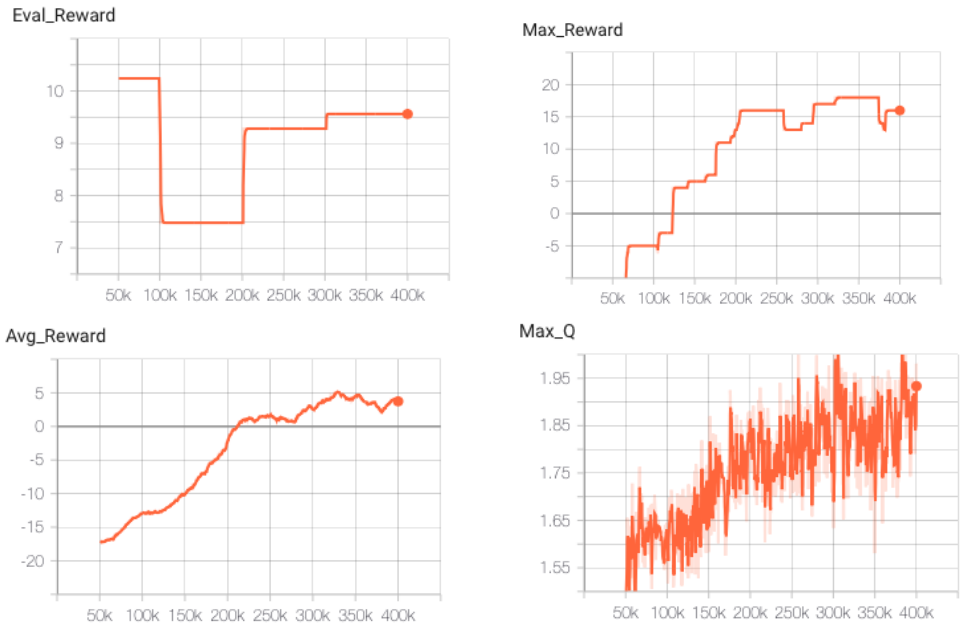
Figure 3: `Pong-v0` environment

(a) (**coding and written**, 5 pts). Now we're ready to train on the Atari `Pong-v0` environment. First, launch linear approximation on pong with `python q7_train_atari_linear.py` **on Azure** either on the GPU or CPU. This will train the model for 500,000 steps and should take approximately an hour on the GPU. Training on the CPU will take longer. Briefly qualitatively describe how your agent's performance changes over the course of training. Do you think that training for a larger number of steps would likely yield further improvements in performance? Explain your answer.

(b) (**coding and written**, 10 pts). In this question, we'll train the agent with DeepMind's architecture on the Atari `Pong-v0` environment. Run `python q7_train_atari_nature.py` **on Azure**. This will train the model for 400k steps. To speed up training, we have trained the model for 2.7 million steps and loaded this in for you which can be found on line 45 of `q7_train_atari_nature.py`. You are responsible for training it to completion, which should take roughly **24 hours** on a *CPU* or around **3 hours** on a *GPU*. Attach the plot `scores.png` from the directory `results/q7_train_atari_nature` to your writeup. You should get a score of around 9-11 after 3.1 million total time steps. As stated previously, the DeepMind paper claims average human performance is -3.

As the training time is roughly 24 hours, you may want to check after a few epochs that your network is making progress. The following are some training tips:

- If you terminate your terminal session, the training will stop. In order to avoid this, you should use `screen` to run your training in the background.
- The max of the q values should also be increasing
- The standard deviation of q shouldn't be too small. Otherwise it means that all states have similar q values
- You may want to use Tensorboard to track the history of the printed metrics. You can monitor your training with Tensorboard by typing the command `tensorboard --logdir=results` and then connecting to `ip-of-you-machine:6006`. Below are our Tensorboard graphs from one training session:



(c) (**written**, 3 pts) In a few sentences, compare the performance of the DeepMind DQN architecture with the linear Q value approximator. How can you explain the gap in performance?

(d) (**written**, 3 pts) Will the performance of DQN over time always improve monotonically? Why or why not?

8 The Environmental Impact of Deep RL

Researchers increasingly turn to deep learning models trained on GPUs to solve challenging tasks such as Atari. However, training deep learning models can have a significant impact on the environment [3][1]. According to a recent study [6], training BERT on GPU produces carbon emissions roughly equivalent to a trans-American flight. Closer to home, [1] considers a "Deep RL class of 235 students (such as Stanford's CS234). For a homework assignment, each student must run an algorithm 5 times on Pong. The class would save 888 kWh of energy by using PPO versus DQN, while achieving similar performance. This is roughly the same amount needed to power a US home for one month" [1]. In the following questions, we will investigate the carbon footprint of our assignment and discuss researcher responsibility.

(a) (**written**, 4pts) Papers such as [3][1] strive for precise calculations of the carbon footprints of models, considering factors such as the type of GPUs used, the location of the data center, and length of model training. Use the tool on this webpage to calculate the likely carbon emissions resulting from your training of Pong on this assignment. What was your result for a single training attempt? For the whole assignment? Explain how you arrived at the final numbers. Compare your answers to the carbon footprint of the original DeepMind Atari paper [5] which takes about 50 hours to train a single game (without hyperparameter tuning). What is your estimate of DeepMind Atari's carbon footprint for a single game? **Clearly state the type of GPU used, time taken and the server location.** If you don't know the exact values for fields such as the type of GPU used or the server location, choose the settings given below:

- GPU: Titan RTX
- Server location: West US

(b) (**written**, 4pts) What are at least two specific ways to reduce the carbon footprint of training your DQN model (or any other deep RL model)? For example, [3] states that one way to reduce the number of models you train is to use random hyperparameter search as opposed to a grid search as the former is more efficient. The linked articles have general ideas for reducing carbon emissions of models. You are welcome to refer to them or other research, and please cite the sources of your ideas in doing so.

(c) (**written**, 5pts) Avram Hiller [2] has argued that since without extenuating circumstances it is "wrong to perform an act which has an expected amount of harm greater than another easily available alternative," we ought to take actions to avoid even minor climate emissions when there are easily available alternatives. Read over the mitigation strategies for individual machine learning researchers and for industry developers/framework maintainers referenced on page 26 of [1]. According to this principle, which of these proposed actions would RL researchers have a responsibility to take? List all the actions you think would be relevant and provide justifications for why two of the actions you listed ought to be taken according to this principle. Finally, do you agree with this principle? Why or why not?

References

[1] Peter Henderson et al. *Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning*. 2020. arXiv: 2002.05651 [cs.CY]. URL: https://www.jmlr.org/papers/volume21/20-312/20-312.pdf.

[2] Avram Hiller. "Climate Change and Individual Responsibility". In: *The Monist* 94.3 (2011), pp. 349–368. ISSN: 00269662. URL: http://www.jstor.org/stable/23039149.

[3] Alexandre Lacoste et al. *Quantifying the Carbon Emissions of Machine Learning*. 2019. arXiv: 1910.09700 [cs.CY].

[4] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[5] Volodymyr Mnih et al. "Playing Atari With Deep Reinforcement Learning". In: *NIPS Deep Learning Workshop*. 2013.

[6] Emma Strubell, Ananya Ganesh, and Andrew McCallum. *Energy and Policy Considerations for Deep Learning in NLP*. 2019. arXiv: 1906.02243 [cs.CL].