Homework #2

CSC2547 – Introduction to Reinforcement Learning
(Spring 2021)

- **Deadline:** Friday, March 26, 2021 at **16:59**.
- **Submission:** You need to submit two files through MarkUs. One is a PDF file including all your answers and plots. The other is a source file (potentially Zipped) that reproduces your answers. You can produce the file however you like (e.g. LaTeX, Microsoft Word, etc) as long as it is readable. Points will be deducted if we have a hard time reading your solutions or understanding the structure of your code. If the code does not run, you may lose most/all of your points for that question.
- **Late Submission:** 10% of the marks will be deducted for each day late, up to a maximum of 3 days. After that, no submissions will be accepted.
- **Collaboration:** Homework assignments must be done individually, and you cannot collaborate with others.

1 Short Questions [10 pts]

(a) **[6pt]** Consider the Stochastic Approximation conditions:

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty.$$

- **[2pt]** Does $\alpha_t = \alpha$ (constant) satisfy these conditions?
- **[2pt]** Verify that the sequence $\alpha_t = \frac{1}{t+1}$ satisfies these conditions.
- **[2pt]** Let $\alpha_t = \frac{1}{t^p+1}$. For what range of p these conditions are satisfied?

(b) **[2pt]** What is the difference between an on-policy and off-policy sampling scenario?

(c) **[2pt]** Is Q-Learning an on-policy or an off-policy algorithm? Why?

2 Overestimation bias in Q-Learning [10 pts]

In Q-Learning, we encounter the issue of overestimation bias. This issue comes from the fact that to calculate our targets, we take a maximum of $\hat{Q}$ over actions. We use a maximum over estimated values ($\hat{Q}$) as an estimate of the maximum value ($\max_a Q(x, a)$), which can lead to significant positive bias.

(a) **[5pt]** Assume that our estimated $\hat{Q}$ function is an unbiased estimator of Q, the true Q function, i.e. $\mathbb{E}\left[\hat{Q}(x, a)\right] = Q(x, a) \quad \forall x, a$ (here the expectation is w.r.t. runs of experiments and data used to estimate $\hat{Q}$). Show that even in this simplified setting, the overestimation issue occurs, i.e.

$$\forall x, \quad \mathbb{E}\left[\max_a \hat{Q}(x, a)\right] \geq \max_a Q(x, a)$$

(b) **[5pt]** Suppose that we use "double Q-Learning". Here, we learn two $\hat{Q}$ functions, $\hat{Q}_1$ and $\hat{Q}_2$, and we use one estimate to determine the maximizing action, $A^* = \text{argmax}_a \hat{Q}_1(x, a)$, and the other to estimate the action's value, $\hat{Q}_2(x, A^*) = \hat{Q}_2(x, \text{argmax}_a \hat{Q}_1(x, a))$. We assume that both $\hat{Q}_1$ and $\hat{Q}_2$ are unbiased estimates of Q. Show that this estimate is unbiased in the sense that

$$\mathbb{E}\left[\hat{Q}_2(x, A^*)\right] = Q(x, A^*).$$

(Note that we also repeat the process with the role of the two estimates reversed to obtain a second estimate $\hat{Q}_1(x, \text{argmax}_a \hat{Q}_2(x, a))$).

3 Convergence of TD [20 pts]

We have proved the convergence of the Q-Learning algorithm (Theorem 4.3 in the Lecture Notes on RL). In this exercise, you prove that the TD learning, used for policy evaluation, generates a sequence of value functions V_t that convergences to V^π. We focus on finite state-action MDPs. You can simply follow the result and the proof in the lecture notes and modify them accordingly. Your solution should have the following parts:

- **[5pt]** A clear statement of the theorem. The theorem should have all the required conditions, i.e., it should be a standalone mathematical statement.
- **[15pt]** A complete proof (it should closely follow the proof for the Q-Learning algorithm).

4 Implementing DQN [60 pts]

In this question, you will implement a simplified version of the Deep Q-Network by Mnih et al. [2015]. To allow us to run this code on our local machines without taking too much time, we will learn directly from the state representations rather than visual observations as Mnih et al. [2015] do. All other elements, however, will be the same. We will be learning a Q_θ function with parameters θ using stochastic gradient descent (SGD) and nonlinear function approximation (i.e. neural nets). DQN is a practical implementation of FQI covered in lecture, but with a couple of modifications to make it suitable for a DNN.

We will implement a replay buffer and a target network. A replay buffer, $\mathcal{D}$, is used to store transitions observed while interacting with the environment. The replay buffer is sampled during training to update the Q function.

The target network, $Q_{\theta'}$ with parameters θ' is used to compute the target values in our loss. In DQN, the target network is held fixed for k number of steps and then updated to have the same weights as Q_θ by copying $\theta' \leftarrow \theta$.

The loss function we use for training the value function is:

$$L(\theta) = \mathbb{E}_{(X_t, A_t, R_t, X_{t+1}) \sim \mathcal{D}} \left[\left(R_t + \gamma \max_{a' \in \mathcal{A}} Q_{\theta'}(X_{t+1}, a') - Q_\theta(X_t, A_t) \right)^2 \right],$$

with $(X_t, A_t, R_t, X_{t+1}) \sim \mathcal{D}$ meaning that the tuple is samples from the replay buffer $\mathcal{D}$.

(a) **[10pt]** Answer these short questions. We will look at some implementation details for the original work. Our implementation will have some differences.

1. **[2pt]** What are the similarities and differences between DQN and the AVI/FQI framework introduced in the class? Briefly discuss.
2. **[2pt]** What are some benefits of using a replay buffer?
3. **[2pt]** Write the gradient of this objective with respect to θ (one line). Note that we do not take gradients through the parameters of the target network, $Q_{\theta'}$.
4. **[2pt]** Recall that we update the target network every k number of steps and then copy the weights of Q_θ, $\theta' \leftarrow \theta$. What are the tradeoffs of updating too quickly or too slowly?
5. **[2pt]** In the original paper, visual observations from Atari are preprocessed by stacking the last $n = 4$ observations to produce the input. Why

would we do this? What are the tradeoffs of having a larger or smaller n? (we won't be doing this for this assignment)

(b) **[30pt]** We will now implement a simple version of DQN and test it on some easy environments. The experiments in this assignment should take around 2 minutes each.

1. **[20pt]** Implement the following functions in the provided code:
   - (functions to implement in `learn.py`) `compute_DQN_loss`, `update_target`
   - (In `model.py`) `QModel`. This sets the architecture of Q_θ. Choose a simple architecture that allows you to get good results without taking too much time. The environments we will be using will be solvable with simple architectures.
   - (In `schedule.py`) `LinearSchedule.update(.)`, `ExplorationSchedule.get_action(.)`. These functions set up our learning rate scheduler, as well as a scheduler for annealing the ϵ in ϵ-greedy action selection (which means that with probability ϵ, we choose a random action, and with probability $(1-\epsilon)$ we choose the greedy action according to Q_θ. We want to start with a large value for ϵ to encourage exploration (note that in the code, we explore for `learning_start` number of steps before starting updates on Q_θ). We then gradually reduce exploration as our policy improves to stabilize training.
2. **[10pt]** Report Training rewards, Evaluation rewards, max Q, and Loss from the logged plots in `results/` on `Acrobot-v1` and `CartPole-v0` (you can set the environment in `main.py`.) (8 plots) The hyperparameters given in `config` are not tuned to be optimal and you may experiment with these to get better performance. Make sure to test your code with multiple random seeds to make sure the performance is consistent (you can set the random seed using `seed_all()` in `main.py`).

(c) **[20pt]** Double DQN (DDQN) [Van Hasselt et al., 2016] is a variation of DQN to deal with the maximization bias issue of DQN. DDQN is the adaptation of Double Q-Learning to DQN. Instead of learning a separate Q network however, we simply take advantage of the fact that we have two networks already: Q_θ and the target $Q_{\theta'}$. Our targets become:

$$y = r + \gamma Q_{\theta'}\left(X', \underset{a' \in \mathcal{A}}{\operatorname{argmax}}\, Q_\theta(X', a')\right)$$

1. **[10pt]** Implement DDQN by filling in the function `compute_DoubleDQN_loss` in `learn.py`. Note that you can test this function by setting `double = True` in `main.py`.
2. **[10pt]** Produce and report the learning curves (Training rewards, Eval rewards, max Q, Loss) on `Acrobot-v1` and `CartPole-v0` for DDQN (8 plots). Do you notice a difference between DQN and DDQN? (It's fine if you don't!)

References

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. 5

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016. 6