MVA: Reinforcement Learning (2020/2021) **Homework 2**

Approximate Reinforcement Learning

Lecturers: *A. Lazaric, M. Pirotta* *(December 21, 2020)*

Solution by Mahdi KALLEL

Instructions

- The deadline is **December 21, 2020. 23h00**
- By doing this homework you agree to the *late day policy, collaboration and misconduct rules* reported on Piazza.
- **Mysterious or unsupported answers will not receive full credit**. A correct answer, unsupported by calculations, explanation, or algebraic work will receive no credit; an incorrect answer supported by substantially correct calculations and explanations might still receive partial credit.
- Answers should be provided in **English**.

1 TRPO

Compare Conservative Policy Iteration (`https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/KakadeLangford-icml2002.pdf`) and TRPO (`https://arxiv.org/abs/1502.05477`). Highlight similarities and differences.
ANSWER:

Conservative policy iteration is a "conservative greedy" policy improvement method. Unlike other Policy gradient, it's performance is guaranteed to improve for each iteration. This is done with lazy updates on the original policy in the form of

$$\pi^{new}(s,a) := (1-\alpha)\pi(s,a) + \alpha\pi'(s,a)$$

By choosing α well, we get a lower bound on the increase of performance of the policy. With this lower bound, the authors prove that the policy converges to a local minimum in a finite number of steps.

The TRPO algorithm can be thought of an extension of CPI. Like CPI, TRPO guarantees improvement of the performance at each iteration. But instead of bounding the updates to mixtures of the old policy, we consider updates directly on the parameters θ of the distribution.
For this purpose we consider an update of the form :

$$\max_{\tilde{\theta}} A_\theta(\tilde{\theta})$$
$$\text{s.t. } D_{KL}^{max}(\pi_\theta || \pi_{\tilde{\theta}}) \leq \eta$$

In policy gradient updates, a small change in the parameters can result in high changes in the policy. This is especially the case for the expressive neural networks. TRPO overcomes this issue by adding a constraint on the change induced on the policy. This allows for changing the parameters around a small trust region around θ and allows for more diverse policies than CPI.

2 Linear TD

Consider the Linear TD(0) algorithm. Given a tuple (s_t, a_t, s_{t+1}, r_t) such that $s_{t+1} \sim p(\cdot|s_t, a_t)$ with $a_t \sim \pi$ and $r_t \sim r(s_t, a_t)$, the TD update rule is given by

$$\theta_{t+1} = \theta_t + \alpha\Big(r_t + \gamma\phi_{t+1}^\intercal\theta_t - \phi_t^\intercal\theta_t\Big)\phi_t$$

where $\phi_t = \phi(s_t)$ and $\alpha_t \geq 0$. We ask to characterize the expected behavior of the steps taken by the TD algorithm in "steady state". Let $A = \mathbb{E}[\phi_t(\phi_t - \gamma\phi_{t+1})]$ and $b = \mathbb{E}[r_t\phi_t]$ be the steady state matrices, where the expectation is such that $s_t \sim d^\pi$, $s_{t+1} \sim p(\cdot|s_t, a_t)$, $a_t \sim \pi(\cdot|s_t)$. Note that d^π is the stationary distribution of policy π.

1. Write the expected updated $\mathbb{E}[\theta_{t+1}|\theta_t]$ as a function of A and b. If the process converges, what is the fixed point?

   ANSWER :

$$\mathbb{E}[\theta_{t+1}|\theta_t] = \mathbb{E}[\theta_t + \alpha\Big(r_t + \gamma\phi_{t+1}^\intercal\theta_t - \phi_t^\intercal\theta_t\Big)\phi_t|\theta_t] = \theta_t + \alpha(\mathbb{E}[r_t\phi_t] + \mathbb{E}[\phi_t(\gamma\phi_{t+1}^T - \phi_t^T)]\theta_t)$$

$$\mathbb{E}[\theta_{t+1}|\theta_t] = \theta_t + \alpha(-A\theta_t + b)$$

If the process converges we have

$$\mathbb{E}[\theta_{t+1}|\theta_t] = \theta_t$$
$$\implies \theta_t = \theta_t + \alpha(-A\theta_t + b)$$
$$\implies \theta_t = A^{-1}b$$

2. If correctly derive, you should notice that A plays a central role in the stability of the update. If A is positive definite, θ_t shrinks at every update. Given the definition of A show that $A = \Phi^\intercal D(I - \gamma P)\Phi$ for appropriate matrices.

   ANSWER :

$$A = \mathbb{E}[\phi_t(\gamma\phi_{t+1}^T - \phi_t^T)] = \gamma\mathbb{E}[\phi_t\phi_{t+1}^T] - \mathbb{E}[\phi_t\phi_t^T]$$

$$\mathbb{E}[\phi_t\phi_t^T] = \sum_s \phi(s)p(s_t = s)\phi(s)^T = \Phi D\Phi^T$$

If we have $A = \Big(C_1|\dots|C_n\Big)$ And $B = \Big(C_1'|\dots|C_n'\Big)$ Then $AB^T = \sum_i C_i C_i'^T$

Let $D = Diag\big[p(s_t = s)\big], \Phi = \Big(\phi(s_1)|\dots|\phi(s_n)\Big)$

For $A = \Phi D, B = \Phi$ We get that $\Phi D\Phi^T = \mathbb{E}[\phi_t\phi_t^T]$

$$\mathbb{E}[\phi_t\phi_{t+1}^T] = \sum_{s',s} p(s_{t+1} = s', s_t = s)\phi(s)\phi(s')^T = \sum_s \phi(s)p(s_t = s)\sum_{s'} p(s_{t+1} = s'|s_t = s)\phi(s')^T$$

For P st, $\Big(P\Big)_{s,s'} = p(s_{t+1} = s'|s_t = s)$ we get that $\Phi DP\Phi^T = \mathbb{E}[\phi_t\phi_{t+1}^T]$

Finally we get that $A = \mathbb{E}[\phi_t(\gamma\phi_{t+1}^T - \phi_t^T)] = \gamma\mathbb{E}[\phi_t\phi_{t+1}^T] - \mathbb{E}[\phi_t\phi_t^T] = \Phi^\intercal D(I - \gamma P)\Phi$

3. Given this decomposition show that A is positive definite.
Hints: 1) Any matrix M is positive definite if and only if the symmetric matrix $S = M + M^\intercal$ is positive definite. 2) Any symmetric real matrix S is positive definite if all of its diagonal entries are positive and greater than the sum of the absolute values of the corresponding off-diagonal entries.
ANSWER :

$$\Phi^\intercal D(I-\gamma P)\Phi = \Phi^\intercal B\Phi = A$$

$$A + A^T = \Phi(B+B^T)\Phi^T = \Phi(2D - \gamma(DP + P^T D))\Phi^T$$

Lets show that $2D - \gamma(DP + P^T D) \in \boldsymbol{S}_n^+$

$$\sum_s \Big(DP + P^T D\Big)_{ss'} = \sum_s P(s_t = s')\big[P(s_{t+1} = s' | s_t = s) + P(s_{t+1} = s | s_t = s')\big]$$

$$\sum_s p(s_{t+1} = s' | s_t = s) = \sum_s \frac{p(s_{t+1} = s' \cap s_t = s)}{p(s_t = s)} \text{ and we have } P(A \cap B) \leq P(B)$$

$$\implies \sum_s \frac{p(s_{t+1} = s' \cap s_t = s)}{p(s_t = s)} \leq \sum_s \frac{p(s_t = s)}{p(s_t = s)} = 1 (1)$$

$$\text{Secondly } \sum_s p(s_{t+1} = s | s_t = s') = \sum_s \frac{p(s_{t+1} = s \cap s_t = s')}{p(s_t = s')} = 1 (2)$$

$$\text{From (1) and (2) we get that } \sum_s \Big(DP + P^T D\Big)_{ss'} \leq 2 * P(s_t = s')$$

therefore if we use hint 2, we get that :

$$2D - \gamma(DP + P^T D) \in \boldsymbol{S}_n^+$$

$\implies \Phi^T(2D - \gamma(DP + P^T D))\Phi \in \boldsymbol{S}_n^+$ by the scalar product definition of semi positive matrices

Using hint 1 we get that A $\in\in \boldsymbol{S}_n^+$

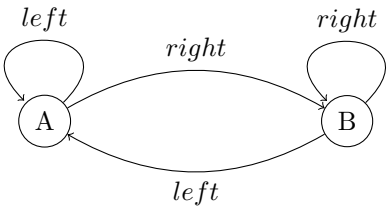
This shows that the TD(0) update is stable in the stationary regime. Additional conditions and steps are necessary to prove convergence.

Off-policy learning. Often we have access to a large amount of data collected through a set of different policies, called behavioral policies. The objective will be to use these samples to compute the value function of a target policy π. For simplicity suppose that we have a single behavioral policy $\rho \neq \pi$. The off-policy Linear TD(0) [**?**] is simply

$$\theta_{t+1} = \theta_t + \alpha \boldsymbol{w_t} \Big(r_t + \gamma \phi_{t+1}^\intercal \theta - \phi_t^\intercal \theta_t\Big)\phi_t \tag{1}$$

where $w_t = \frac{\pi(a_t|s_t)}{\rho(a_t|s_t)}$ is the importance weight.

- Consider the following MDP with 2 states and 2 actions



Consider a behavioral policy $\rho(right|\cdot) = 1/2$ and a target policy $\pi(right|\cdot) = 1$. The reward function is 0 everywhere and the value function is parametrized through a single parameter θ such that $V(A) = 0.8\theta$ and $V(B) = 2\theta$. Show that the update (1) diverges. Assume that $\theta_t = 10$, $\alpha = 0.1$ and $\gamma = 0.9$.

ANSWER :

This is a special case of linear TD(0) where we have $\phi(s_t) = 0.8 * \mathbf{1}_{s_t=A} + 0.2 * \mathbf{1}_{s_t=B}$ and b=0.

From 1) we get that $\mathbb{E}[\theta_{t+1}|\theta_t] = \theta_t + \alpha(A\theta_t)$ where $A = \mathbb{E}[w_t\phi_t(\gamma\phi_{t+1} - \phi_t)]$

To go from A to A or from B to A we need to take a left action

$$\implies w_t \mathbf{1}_{s_t=B}\mathbf{1}_{s_{t+1}=A} = w_t \mathbf{1}_{s_t=A}\mathbf{1}_{s_{t+1}=A} = 0$$

$$\mathbb{E}[w_t\phi_t\phi_t] = \mathbb{E}[0.04 * \mathbf{1}_{s_t=B} + 0.64 * \mathbf{1}_{s_t=A} + 0.32 * \mathbf{1}_{s_t=A} * \mathbf{1}_{s_t=B}]$$
$$= 0.04P(s_t = B) + 0.64P(s_t = A) = 0.04 * \frac{1}{4} + 0.64 * \frac{1}{4} = 0.17$$

$$\mathbb{E}[w_t\phi_{t+1}\phi_t] = \mathbb{E}[2 * 0.04 * \mathbf{1}_{s_t=B} * \mathbf{1}_{s_t+1=B} + 2 * 0.16 * \mathbf{1}_{s_t=B} * \mathbf{1}_{s_{t+1}=A}]$$

$$\mathbb{E}[w_t\phi_{t+1}\phi_t] = \frac{1}{16}(0.08 + 0.32) = 0.025$$

$$\mathbb{E}[\theta_{t+1}|\theta_t] = \theta_t(1 - \alpha(\gamma * 0.025 - 0.17)) = (1.01475)\theta_t$$

We find that on average the update is increasing, therefore it will explode.

3 REINFORCE

In class we have seen the derivation of the policy gradient theorem in the "Monte-Carlo style". Recall that the policy gradient is given by

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\mathbb{P}(\cdot|\pi_\theta)}\left[\nabla_\theta \log \mathbb{P}(\tau|\pi_\theta)R(\tau)\right]$$

where $R(\tau) = \sum_{t=1}^{|\tau|} r_t$.

- By construction the policy gradient is on-policy. Derive an off-policy variant by assuming to collect samples from a behavioral policy $\mu(s, a)$. The target policy, i.e., the policy for which we want to compute the gradient is π_θ

ANSWER :

$$J(\pi_\theta) = \mathbf{E}_{\tau\sim\pi_\theta}\Big[R(\tau)\Big] = \mathbf{E}_{\tau\sim\mu}\Big[\frac{\pi_\theta(\tau)}{\mu(\tau)}R(\tau)\Big] = \int \mu(\tau)\frac{\pi_\theta(\tau)}{\mu(\tau)}R(\tau)\,d\tau$$

$$\nabla J(\pi_\theta) = \int \mu(\tau)\nabla\pi_\theta(\tau)\frac{R(\tau)}{\mu(\tau)}\,d\tau = \int \mu(\tau)\nabla\log(\pi_\theta(\tau))\frac{\pi_\theta(\tau)}{\mu(\tau)}R(\tau)\,d\tau$$

$$= \int \mu(\tau)\nabla\log(\pi_\theta(\tau))R'(\tau)\,d\tau$$

Where $R'(\tau)$ is tractable : $R'(\tau) = R(\tau) * \sum_{t=1}^{|\tau|}\frac{\pi_\theta(a_t,s_t)}{\mu(a_t,s_t)}$

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\mu(\tau)}\left[\nabla_\theta \log \mathbb{P}(\tau|\pi_\theta)R'(\tau)\right]$$

So far we have seen the "Monte Carlo' derivation of the gradient. It is possible to derive the gradient theorem by leveraging the recursive structure of the Bellman equation. Recall that for a γ-discounted infinite-horizon MDP, we define the policy performance $J(\pi_\theta) = \mathbb{E}\left[\sum_{t=1}^{+\infty}\gamma^{t-1}r_t|s_1\sim\rho,\pi\right]$. Then, the policy gradient is given by

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim d^\pi}\mathbb{E}_{a\sim\pi_\theta(s,\cdot)}\left[\nabla_\theta \log \pi_\theta(s,a)Q^\pi(s,a)\right] \tag{2}$$

where $d^\pi(s) = \lim_{T\to\infty}\sum_{t=1}^{T}\gamma^{t-1}\mathbb{P}(s_t = s|\pi, s_1\sim\rho)$

- Derive the gradient in Eq. 2. *Hint: you can start from* $V^\pi(s) = \sum_a \pi(s,a)Q^\pi(s,a)$

ANSWER :

$$\nabla V^\pi(s) = \sum_a \nabla \pi(s,a) Q^\pi(s,a) = \sum_a \nabla \pi(s,a) + \nabla Q^\pi(s,a)$$

$$\sum_a \pi(s,a) Q^\pi(s,a) \nabla log(\pi(s,a)) + \pi(a,s) \nabla (R(s,a) + \gamma \mathbb{E}_{s' \sim P(s,a)}[V^\pi(s')])$$

The distribution of instantaneous reward is independent from the choice of policy, summing over all possible actions we get :

$$\nabla V^\pi(s) = \sum_a \pi(s,a) Q^\pi(s,a) \nabla log(\pi(s,a)) + \nabla \gamma \mathbb{E}_{s' \sim P(s,a)}[V^\pi(s')]$$

Let d_t^π denote the distribution over s_t and a_t induced by policy π from the initial state distribution d_0 :

$$\mathbb{E}_{s \sim d_t^\pi}[\nabla V^\pi(s)] = \mathbb{E}_{s \sim d_t^\pi, a \sim \pi}[\nabla log(\pi(s,a)) Q^\pi(s,a)] + \mathbb{E}_{s \sim d_t^\pi, s' \sim p(s,a)} \nabla \gamma [V^\pi(s')]$$

$$= \mathbb{E}_{s \sim d_t^\pi, a \sim \pi}[\nabla log(\pi(s,a)) Q^\pi(s,a)] + \gamma \mathbb{E}_{s' \sim d_{t+1}^\pi} \nabla [V^\pi(s')]$$

$$= \mathbb{E}_{s \sim d_t^\pi, a \sim \pi}[\nabla log(\pi(s,a)) Q^\pi(s,a)] + \gamma \mathbb{E}_{s', a' \sim d_{t+1}^\pi}[\nabla log(\pi(s',a')) Q^\pi(s',a')] + \mathbb{E}_{s'', a'' \sim d_{t+2}^\pi}[V^\pi(s'')]$$

By using the recursive structure of this equation we get :

$$\mathbb{E}_{s \sim d_t^\pi}[\nabla V^\pi(s)] = \sum_{t'=t}^{\infty} \gamma^{t'-t} \mathbb{E}_{s,a \sim d_t^{\pi\prime}}[\nabla log(\pi(s,a)) Q^\pi(s,a)]$$

By noticing that $J(\theta) = \mathbb{E}_{s \sim d_1^\pi}[V^\pi(s)]$ we get :

$$\nabla J(\theta) = \sum_{t'=1}^{\infty} \gamma^{t'-1} \mathbb{E}_{s \sim d_t^{\pi\prime}, a \sim \pi}[\nabla log(\pi(s,a)) Q^\pi(s,a)]$$

By using that $d^\pi(s) = \sum_{t=1}^{\infty} d_t^\pi(s)$ we get :

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim d^\pi} \mathbb{E}_{a \sim \pi_\theta(s,\cdot)[\nabla log(\pi(s,a)) Q^\pi(s,a)]}$$

4 DQN

The goal of this exercise is to compare different variants of Deep Q-Learning (a.k.a. DQN).

DQN. Recall from the class the DQN aims to minimize the following loss function

$$L(\theta) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta') - Q(s,a;\theta) \right)^2 \right]$$

where θ' is the parameter of the target network updated every C iterations from θ.

1. (written) This objective resembles a classical supervised learning problem. Could you highlight the differences?

   ANSWER

The objective resembles a supervised Regression objective. The main difference in a DQN setting is that the regression targets $r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta')$ are not fixed before training. They depend on the previous set of parameters. Back propagation is done over the predictions $r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta')$.
The fact that the targets depend on the previous weights means that they will change over training.

2. (written) Could you describe the role of C and the trade-off at play in choosing a good value for C?

ANSWER :

When training a DQN agent, it is common practice to keep two copies of our value network. One for to output the regression targets, and the second for the regression predictions. C represents the number of batch updates we realise on the prediction network before transferring it's weights to the target network.
Choosing a large C, means that we realise many updates to the prediction network while the target network is fixed. This should lead to the prediction network to fit the targets better which is good.But, on the other hand after many updates, the targets provided become obsolete for the new network.
Choosing a low C, means that the target network gets updated more frequently. This can be harmful especially if the state space is large. Remember that we perform batch updates, not full data set iterations. Maybe the prediction network did not update for a certain state, and the target for this state changes fast so the prediction model will give lousy predictions for this state.

Replay Memory. As we play, we store our transitions (s, a, r, s') in a buffer $\mathcal{D}$. Old examples are deleted as we store new transitions. To update our parameters, we sample a minibatch from the buffer and perform a stochastic gradient descent update.

ϵ-greedy exploration. For exploration, we use an ϵ-greedy strategy. It is standard to use a linear annealing schedule from ϵ_{start} to $\epsilon_{\min}$.

Q-function representation. As suggested in the original paper, the Q-function is represented using a neural network with input a state s and output a vector of dimension $|\mathcal{A}|$ containing the estimated Q-value for each action $a \in \mathcal{A}$.

1. (written) What is one benefit of representing the Q function as $Q(s; \theta) \in \mathbb{R}^{|\mathcal{A}|}$?

ANSWER:

The other choice would be to model Q function as $functionasQ(s, a, \theta)$ this would mean that for each action a full pass through the network is required to get Q(s,a).
In our setting only a single pass is required to get Q(s,a) for all actions. This comes really handy since we need to get the max of all actions at each update.
The difference becomes more important as we have more actions to choose from. In conclusion this allows for a faster training.

1. (code) Implement DQN. We provided an almost complete version of DQN in `dqn_start.py.py`. Implement the network on Fig. **??**(left).

The code generates two plots. The first plot shows the performance (cumulative reward) over the learning process (i.e., as a function of time steps). The second figure shows the performance of the associated greedy policy on a test environment averaged over multiple episodes.[1] It also saves the results in a file called `dqn_results.txt`.

1. (code) A single run is not enough to evaluate the performance of an algorithm. To have significant results, we need to perform multiple repetitions. Change the code to run multiple versions of DQN and reports the average performance and uncertainty as function of time steps (for both figures). We provide a script that reads files generated by `dqn_start.py.py` that can be used to generate the requested figures.[2]

ANSWER

[1] Usually, this metric is evaluated less frequently since it is more computationally costly. It is a parameter in the code. By default we run every 2 episodes.

[2] It is not necessary to use this script to generate figures.

Here's an example of the generated plot for a simple DQN network.

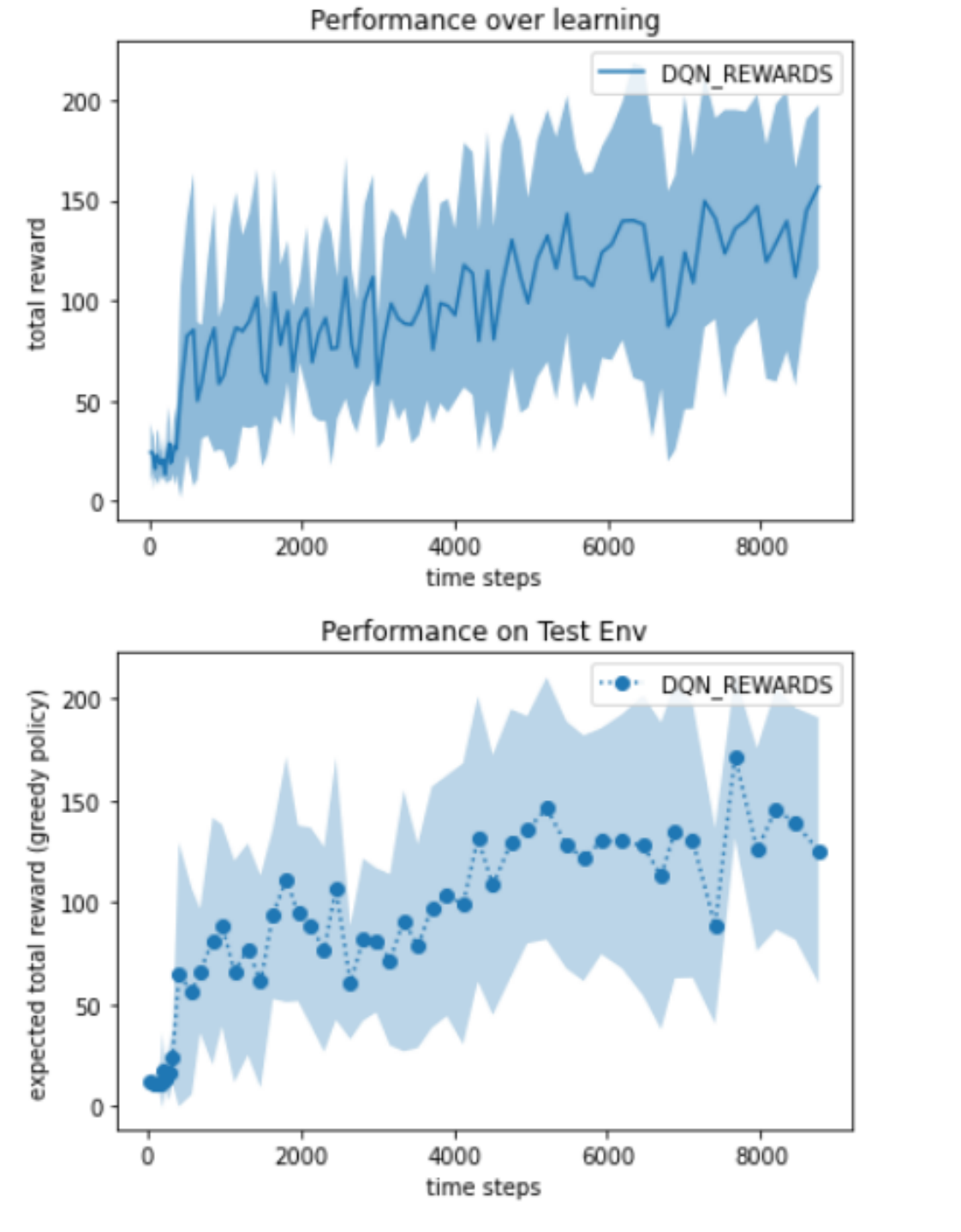


Figure 1: Reward collected using the ϵ greedy policy and the full greedy policy

Competitors. We would like to evaluate DQN with newer variants of DQN, namely Double DQN and Dueling DQN. In class we have seen the principle of Double DQN while we refer to the original paper for Dueling DQN (`https://arxiv.org/abs/1511.06581`).
The difference between DQN and Double DNQ is how the target network is built. In Double DQN we use the greedy action suggested by $Q(s, a; \boldsymbol{\theta})$ to evaluate the target (i.e., θ'), see the appendix of `https://arxiv.org/abs/1511.06581`. In Dueling DQN, the Q function is estimated as

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a')$$

V and A share parameters. Dueling DQN can be implemented "standard" or "double".
Starting from the code of `dqn_start.py.py`,

1. (code) Implement Double DQN. Use the same network of DQN.
2. (code) Implement Dueling Double DQN. See Fig. **??** for the network with shared parameters.
3. Compare the performance of the algorithms

ANSWER

We report on the evolution of the rewards collected by various DQN agents in the plot below. This has been generated using 10 copies of each agent.
We find that the average collected reward by a simple DQN or a Duelling DQN is better.
This is probably because the objective we are learning is simple so having an optimistic estimation of states (induced by the positive bias of the DQN algorithm) helps the model have higher gradients and thus converge faster.
Whereas for architectures including Double Learning, we find they take more time to reach the same reward values. This is probably due to the absence of the positive bias, and thus the model is taking more conservative updates.
Please report to the code.ipynb for the full plots.

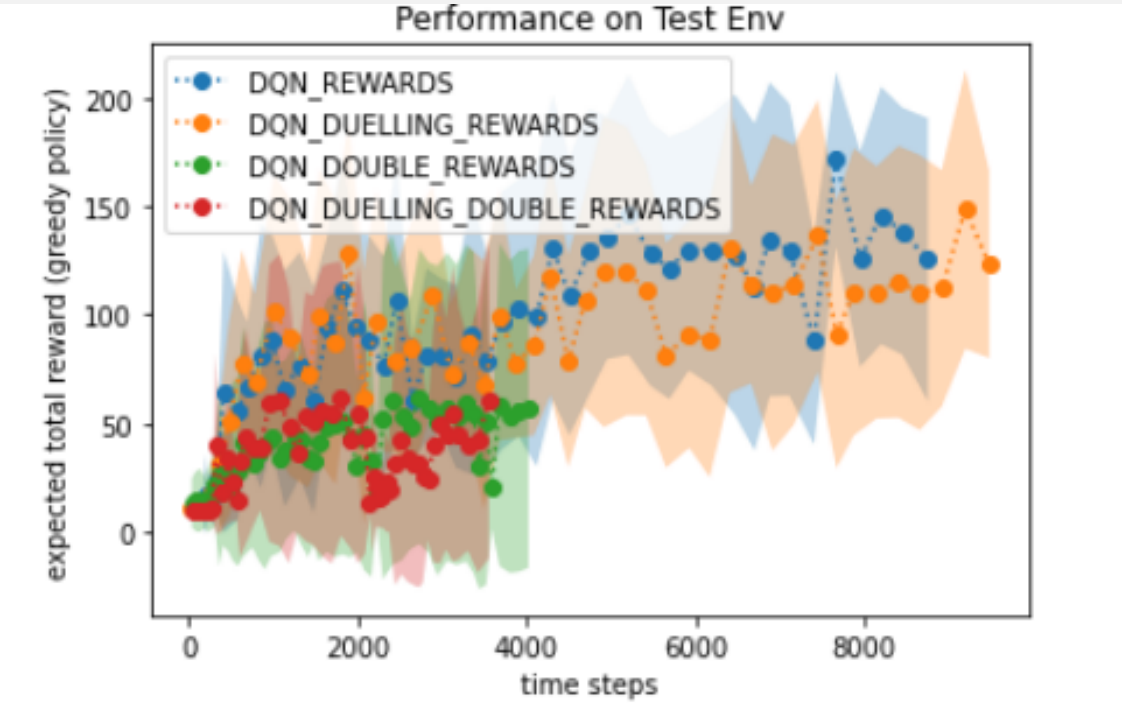


Figure 2: Rewards collected on test env for various algorithms