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

Approximate Reinforcement Learning

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

Solution by AMEKOE Kodjo Mawuena

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.

TRPO Answers

Comparison of the Conservative Policy Iteration and Trust Region policy Optimization (TRPO) :

1. **Similarities**
   - These two approaches are based on a surrogate function of the expected discounted reward and are policy gradient methods.
   - These are conservative methods since the new policy is given by $\pi_{new} = (1-\alpha)\pi_{old} + \alpha\pi_{new}$ which is a mixture of the old policy and the current,
   - These two approaches help to have monotonic improvement of the surrogate of the expected discounted reward,
2. **Differences**
   - in TRPO α is replaced by the total variation (after by the kullback leibler divergence) which is natural metric (so TRPO is kind of natural policy gradient method). This choice helps to improve both the surrogate and the exact function in monotonic way.
   - the conservative policy iteration improves exact function but not always in monotonic way.
   - in TRPO the penalisation problem is replaced by constraint optimization problem,
   - in the conservative policy iteration , the starting distribution is replaced by a restart distribution which is more uniform.

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?
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.
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.

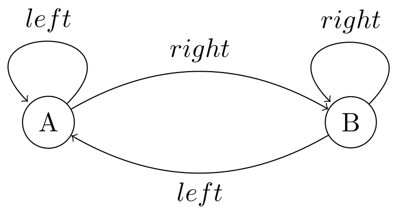
This shows that the TD(0) update is stable in the stationary regime. Additional conditions and steps are necessary to prove convergence.

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 $\gamma = 0.9$.

Linear TD Answers

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

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

where $\phi_t = \phi(s_t)$ and $\alpha_t \geq 0$.

1. The expected updated $\mathbb{E}[\theta_{t+1}|\theta_t]$:

$$\begin{aligned}\mathbb{E}[\theta_{t+1}|\theta_t] &= \mathbb{E}[\theta_t + \alpha_t\Big(r_t + \gamma\phi_{t+1}^\intercal\theta_t - \phi_t^\intercal\theta_t\Big)\phi_t|\theta_t] \\ &= \mathbb{E}[\theta_t|\theta_t] + \alpha_t\mathbb{E}[r_t\phi_t|\theta_t] + \alpha_t\mathbb{E}[(\gamma\phi_{t+1} - \phi_t)^\intercal\theta_t\phi_t|\theta_t] \\ &= \theta_t + \alpha_t\mathbb{E}[r_t\phi_t] + \alpha_t\mathbb{E}[\phi_t(\gamma\phi_{t+1} - \phi_t)^\intercal|\theta_t]\mathbb{E}[\theta_t|\theta_t] \quad \text{because} \quad \mathbb{E}[X|X] = X, \mathbb{E}[XY|X] = X\mathbb{E}[Y|X] \\ &= \theta_t + \alpha_t\mathbb{E}[r_t\phi_t] + \alpha_t\mathbb{E}[\phi_t(\gamma\phi_{t+1} - \phi_t)^\intercal]\theta_t \\ \mathbb{E}[\theta_{t+1}|\theta_t] &= \theta_t + \alpha_t b - \alpha_t A\theta_t\end{aligned}$$

where $A = \mathbb{E}[\phi_t(\phi_t - \gamma\phi_{t+1})^\intercal]$ and $b = \mathbb{E}[r_t\phi_t]$.
Then :

$$\mathbb{E}[\theta_{t+1}|\theta_t] = \theta_t + \alpha_t(b - A\theta_t) \tag{2}$$

If the process conerges, then the fixed point will be θ such $\theta = \theta + \alpha(b - A\theta)$.
So the fixed point is : $\underline{\theta = A^{-1}b}$ for A invertible.

2. We have $A = \Phi^\intercal D(I - \gamma P)\Phi$. Proof :

$$\begin{aligned}A &= \mathbb{E}[\phi_t(\phi_t - \gamma\phi_{t+1})^\intercal] \\ &= \mathbb{E}[\phi_t\phi_t^\intercal] - \gamma\mathbb{E}[\phi_t\phi_{t+1}^\intercal].\end{aligned}$$

With :

$$\begin{aligned}\mathbb{E}[\phi_t\phi_t^\intercal] &= \sum_s d^\pi(s)\phi_s\phi_s^\intercal \\ &= \Phi^\intercal D\Phi\end{aligned}$$

where $\underline{D = diag(d^\pi) \in \mathbb{R}^{S\times S}}$ and $\underline{\Phi = [\phi_1^\intercal; ...; \phi_S^\intercal] \in \mathbb{R}^{S\times d}}$
Moreover,

$$\begin{aligned}\mathbb{E}[\phi_t\phi_{t+1}^\intercal] &= \sum_s d^\pi\phi_s\Big(\sum_{s'} P_{ss'}^\pi\phi_s\Big)^\intercal \\ &= \Phi^\intercal DP\Phi\end{aligned}$$

where $\underline{P = [P_{ss'}^\pi] \in \mathbb{R}^{S\times S}}$ and $P_{ss'}^\pi = \sum_a p(s'|s,a)\pi(a|s)$
So :

$$A = \Phi^\intercal D\Phi - \gamma\Phi^\intercal DP\Phi = \Phi^\intercal D(I - \gamma P)\Phi \tag{3}$$

3. A is positive definite: We consider $A = \Phi^\intercal D(I - \gamma P)\Phi$ with $\Phi \in \mathbb{R}^{S\times d}$ where $d << S$. Then to show that A is positive definite, we will simply show that $M = D(I - \gamma P)$ is positive definite.
If M is symmetric, then by the hint 2, we have simply to prove that all its diagonal entries are positive and greater than the sum of the absolute values of the corresponding off-diagonal entries.
Let $M = [M_{ss'}]$ where $M_{ss} = d^\pi(s)(1 - \gamma P_{ss}^\pi) > 0$ and $M_{ss'} = -\gamma d^\pi(s)P_{ss'}^\pi < 0$ for $s \neq s'$.
As the diagonal entries are positive and off-diagonal entries are negative, we simply have to prove that the sum of all rows and column is positive.

   - Rows:
     Since P is probability (stochastic) matrix, $\sum_{s'} P_{ss'}^\pi = 1$. Then for $\gamma < 1$, we have: $\sum_{s'} M_{ss'} = d^\pi(s)(1 - \gamma\sum_{s'} P_{ss'}^\pi) > 0$. So the sum of all rows is positive.
   - Columns The row vector obtained by summing colums of M is :

$$\begin{aligned}r &= (I - \gamma P)d^\pi \\ &= d^\pi - \gamma Pd^\pi \\ &= d^\pi - \gamma d^\pi \quad \text{because} \quad d^\pi\text{is the stationary distribution} \\ &= d^\pi(1 - \gamma)\end{aligned}$$

     The vector is r is already postive for $\gamma < 1$.

Then M is positive definite.

If M is not symmetric: Let $S = M + M^\intercal$. Then by using the hint 1 and the previous proof to the matrix S with is already symmetric, we have M positive definite.

Finally then matrix A is positive definite since M is positive definite.

Off-policy

The off-policy Linear TD(0) is simply

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

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

If we consider the behavioral policy ρ such that $\rho(right|\cdot) = 1/2$ and a target policy $\pi(right|\cdot) = 1$, $r_t = 0$ everywhere. $V(A) = 0.8\theta \implies \phi_A = 0.8$ and $V(B) = 2\theta \implies \phi_B = 2$. $\theta_t = 10$, $\alpha = 0.1$ and $\gamma = 0.9$, then the update diverges:

For one update(right transition) from the state A to B, we have :

$$\begin{aligned}\theta_{t+1} &= 10 + 0.1 * \frac{1}{0.5}(0 + 0.9 * 2 * 10 - 0.8 * 10) * 0.8 \\ &= 10 + 0.1 * 2(18 - 8) * 0.8 \\ &= 11.6\end{aligned}$$

And update (right transition) from the state B to B gives:

$$\begin{aligned}\theta_{t+1} &= 10 + 0.1 * \frac{1}{0.5}(0 + 0.9 * 2 * 10 - 2 * 10) * 2 \\ &= 10 + 0.1 * 2(18 - 20) * 2 \\ &= 9.2\end{aligned}$$

When the action is left, $\pi(left|\cdot) = 0$, hence , $w_t = 0$ and $\theta_{t+1} = \theta_t$ (so only right action will will make update).

The two updates occur with the same probability, so θ_t will increase in average and then we will have divergence.

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

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 $J(\pi_\theta) = \mathbb{E}\left[\sum_{t=1}^{+\infty}\gamma^{t-1}r_t|s_1\sim\rho,\pi_\theta\right]$. Then, the policy gradient is given by

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim d^{\pi_\theta}}\mathbb{E}_{a\sim\pi_\theta(s,\cdot)}\left[\nabla_\theta \log \pi_\theta(s,a)Q^\pi(s,a)\right] \qquad (4)$$

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

- Derive the gradient in Eq. 4. *Hint: you can start from* $V^\pi(s) = \sum_a \pi(s,a)Q^\pi(s,a)$

REINFORCE Answers

- The policy gradient is given by

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

Derivation of the off-policy variant :

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\tau\sim\mathbb{P}(\cdot|\pi_\theta)}\left[\nabla_\theta \log \mathbb{P}(\tau|\pi_\theta)R(\tau)\right]\\
&= \int \nabla_\theta \left[\log \mathbb{P}(\tau|\pi_\theta)R(\tau)\right]\mathbb{P}(\tau|\pi_\theta)d\tau\\
&= \int \nabla_\theta \log \mathbb{P}(\tau|\pi_\theta)\frac{\mathbb{P}(\tau|\pi_\theta)}{\mathbb{P}(\tau|\mu)}\mathbb{P}(\tau|\mu)R(\tau)d\tau\\
&= \int \left[\frac{\mathbb{P}(\tau|\pi_\theta)}{\mathbb{P}(\tau|\mu)}\nabla_\theta \log \mathbb{P}(\tau|\pi_\theta)R(\tau)\right]\mathbb{P}(\tau|\mu)d\tau\\
\nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\tau\sim\mathbb{P}(\cdot|\mu)}\left[\frac{\mathbb{P}(\tau|\pi_\theta)}{\mathbb{P}(\tau|\mu)}\nabla_\theta \log \mathbb{P}(\tau|\pi_\theta)R(\tau)\right]
\end{aligned}$$

- In γ-discounted infinite-horizon MDP, we have have : $J(\pi_\theta) = \mathbb{E}\left[\sum_{t=1}^{+\infty}\gamma^{t-1}r_t|s_1\sim\rho,\pi_\theta\right]$ and $\nabla_\theta J(\theta) = \mathbb{E}_{s\sim d^{\pi_\theta}}\mathbb{E}_{a\sim\pi_\theta(s,\cdot)}\left[\nabla_\theta \log \pi_\theta(s,a)Q^\pi(s,a)\right]$ with $d^{\pi_\theta}(s) = \lim_{T\to\infty}\sum_{t=1}^{T}\gamma^{t-1}\mathbb{P}(s_t = s|\pi, s_1\sim\rho)$.
  Proof of the expression of the gradient:
  We have $J(\pi_\theta) = \mathbb{E}_{s\sim d^{\pi_\theta}}[V^{\pi_\theta}(s)]$. Then :

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathbb{E}_{s\sim d^{\pi_\theta}}[V^{\pi_\theta}(s)]\\
&= \mathbb{E}_{s\sim d^{\pi_\theta}}[\nabla_\theta V^{\pi_\theta}(s)]
\end{aligned}$$

So let compute first $\nabla_\theta V^\pi(s)$, with $\pi = \pi_\theta$

$$\begin{aligned}
\nabla_\theta V^\pi(s) &= \nabla_\theta \sum_a \pi(s,a)Q^\pi(s,a)\\
&= \sum_a \Big(\nabla_\theta \pi(s,a)Q^\pi(s,a) + \pi(s,a)\nabla_\theta Q^\pi(s,a)\Big)\\
&= \sum_a \left(\nabla_\theta \pi(s,a)Q^\pi(s,a) + \pi(s,a)\nabla_\theta\left(r(s,a) + \gamma\sum_{s'} p(s'|s,a)V^\pi(s')\right)\right)\\
&= \sum_a \left(\nabla_\theta \pi(s,a)Q^\pi(s,a) + \gamma\pi(s,a)\sum_{s'} p(s'|s,a)\nabla_\theta V^\pi(s')\right)\\
&= f(s) + \gamma\sum_{s'}\sum_a \pi(s,a)p(s'|s,a)\nabla_\theta V^\pi(s') \quad \text{with} f(s) = \sum_a \nabla_\theta\pi(s,a)Q^\pi(s,a)\\
&= f(s) + \gamma\sum_{s'} P(s,s')\nabla_\theta V^\pi(s') \quad \text{with} \quad P(s,s') = \sum_a \pi(s,a)p(s'|s,a)\\
&= f(s) + \gamma\sum_{s'} P(s,s')\left(f(s') + \gamma\sum_{s''} P(s',s'')\nabla_\theta V^\pi(s'')\right)\\
&= f(s) + \gamma\sum_{s'} P(s,s')f(s') + \gamma^2\sum_{s''} P(s,s'')\nabla_\theta V^\pi(s'')
\end{aligned}$$

And then by using this recursion of the Bellman equation, we have after k step from the state s to x, we have $\nabla_\theta V^\pi(s) = \sum_x \sum_{k=0}^{\infty} \gamma^k P(s,x) f(x)$

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

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \mathbb{E}_{s\sim d^{\pi_\theta}}[\nabla_\theta V^{\pi_\theta}(s)] \\
&= \sum_s d^{\pi_\theta}(s) f(s) \\
&= \sum_s d^{\pi_\theta} \sum_a \nabla_\theta \pi_\theta(s,a) Q^{\pi_\theta}(s,a) \\
&= \sum_s d^{\pi_\theta} \sum_a \nabla_\theta \log \pi_\theta(s,a) Q^{\pi}(s,a)\pi_\theta(s,a) \quad \text{we use:} \quad \nabla_\theta \log \pi(s,a) = \frac{\nabla_\theta \pi(s,a)}{\pi(s,a)} \\
\nabla_\theta J(\pi_\theta) &= \mathbb{E}_{s\sim d^{\pi_\theta}} \mathbb{E}_{a\sim \pi_\theta(s,\cdot)} [\nabla_\theta \log \pi(s,a) Q^\pi(s,a)]
\end{aligned}$$

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?

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

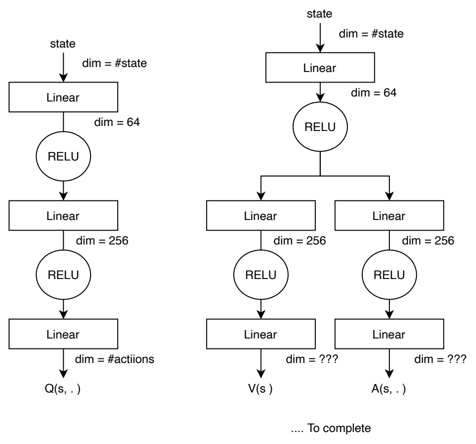


Figure 1: DQN network (left) and partial Dueling DQN (right).

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 $Q(s;\theta) \in \mathbb{R}^{|\mathcal{A}|}$?

Implementation

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

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. 1 for the network with shared parameters.
3. Compare the performance of the algorithms

DQN Answers

In this section we want to compare different variants of Deep Q-learning (DQN)

Standard DQN with replay Memory

The aims DQN aims to minimize the loss :

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

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

1. The objective function of DQN resembles a supervised learning problem but here the target $y = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta')$ depends on the the parameter of interest θ', so could change for the same input (s,a). This is not the case in common supervised learning problem.
2. The parameter C helps to keep the target parameter θ' constant as possible is order to help the algorithm (the Q function) to converge to its target (ie C help to have more stable algorithm).

   However for very large value of C, the algorithm will use many iteration to converge to one target which is not necessary the optimal. Then a good trade-off is important in the choice of C.

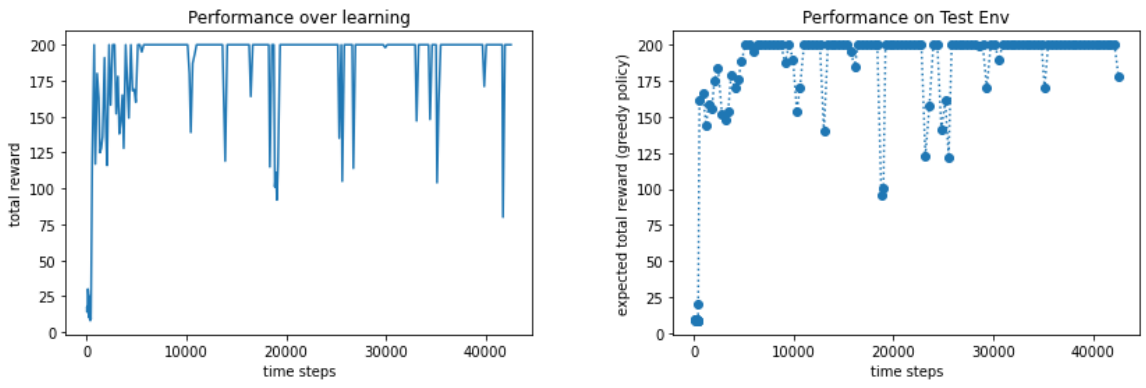
[1] Usually, this metric is evaluated less frequently since it is more computationally costly. It is a parameter in the code. By default we run every 2 episodes.
[2] It is not necessary to use this script to generate figures.

- **Q function representation**

  The representation of the Q function as $Q(s;\theta)$ would help us to work in the continuous state space and there is also possibility to use pretrain models with this representation.

- **Implementation**

  1. Code : see the file *dqn_start*

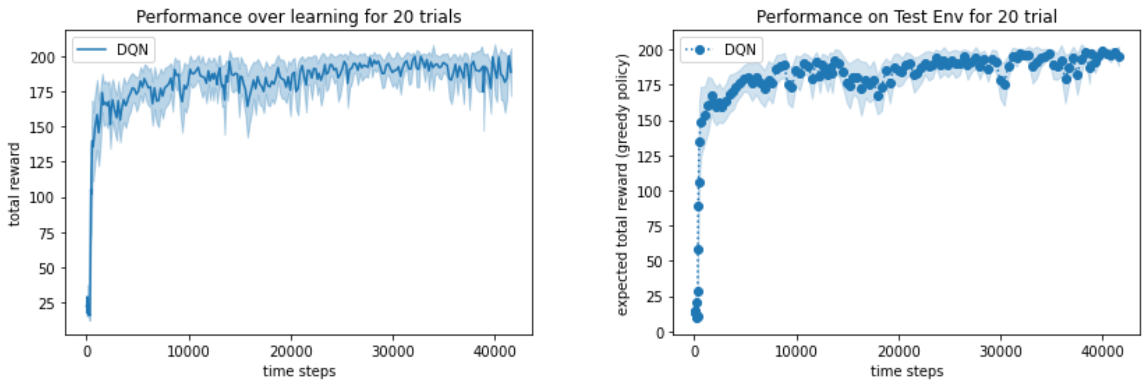


(a) The cumulative reward over the learning process, $C = 20$ (b) The cumulative reward of the associated greedy policy on the test

Figure 2: Single run of the DQN over 250 episodes

The Figure 2 shows the performance of single run of the DQN algorithm. We can see that the maximal cumulative reward that the agent can get in the traning and testing process is 200. However a single is note enough to evaluate the performance of the algorithm.

  2. Multiple trials of DQN



(a) The cumulative reward over the learning process, $C = 20$ (b) The cumulative reward of the associated greedy policy on the test

Figure 3: Multiple run of the DQN over 250 episodes

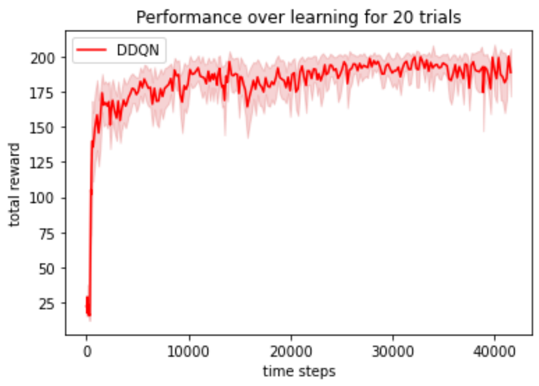
The Figure 2 shows the performance of the DQN for 20 trials. The total reward increases and becomes quasi-stationary after roughly 5000 time steps. We repord also the uncertainty using the standard error : $\sigma_R = 2\frac{std(R)}{\sqrt{t}}$ where $std(R)$ is standard error for $t = 20$ trials. So we plot $y \in \left[\bar{R} - \sigma_R, \bar{R} + \sigma_R\right]$ where $\bar{R}$ is the average over t trials.

It is important to specify that we used arbitrary the time steps obtained for the first trial. However the order of magnitude remains almost the same for the other trials.

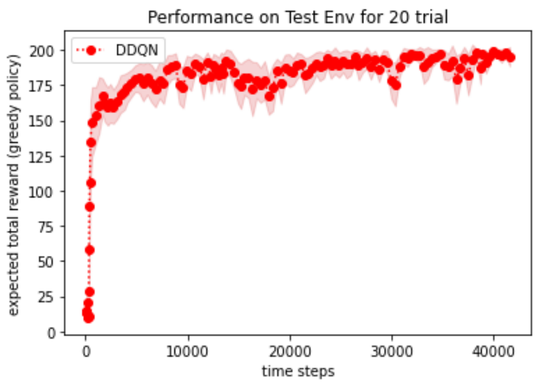
The DQN uses max operator so could be overoptimistic value estimates. Therefore we compare its performance to other algorithms.

Competitors

1. Double DQN (DDQN) :



(a) The cumulative reward over the learning process, $C = 20$



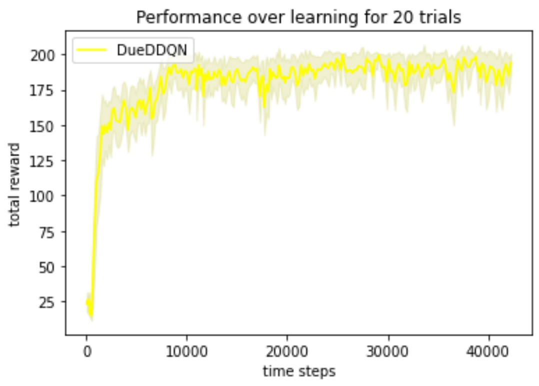
(b) The cumulative reward of the associated greedy policy on the test

Figure 4: Multiple run of the DDQN over 250 episodes

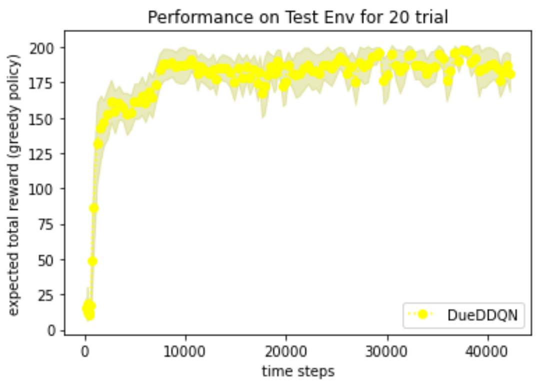
The Figure 4 shows the performance shows the performance of the DDQN for 20 trials. see the code *dqn_start* (THIRD QUESTION).

We also experienced the performance of the variant Dueling Double DQN.

2. Dueling Double DQN (DueDDQN)



(a) The cumulative reward over the learning process, $C = 20$



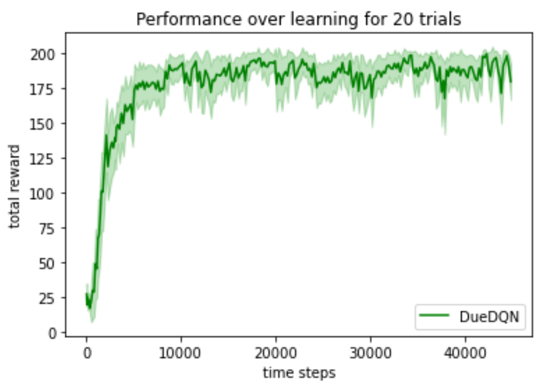
(b) The cumulative reward of the associated greedy policy on the test

Figure 5: Multiple run of the DueDDQN over 250 episodes

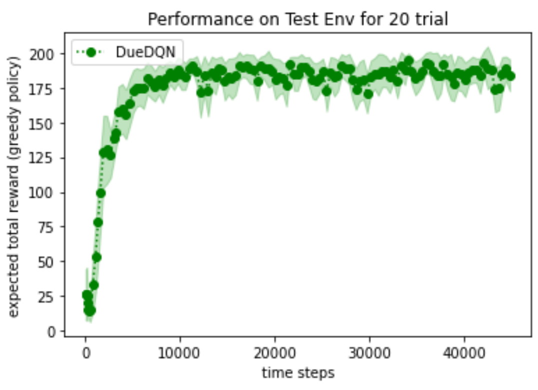
The Figure 5 shows the performance of the DueDDQN for 20 trials.

3. Standard Dueling DQN (DueDQN) (Optional part) :

   We also propose to test the performance of the standard Dueling DQN:



(a) The cumulative reward over the learning process, $C = 20$



(b) The cumulative reward of the associated greedy policy on the test

Figure 6: Multiple run of the DueDQN over 250 episodes

The Figure 6 shows the performance of the DueDQN for 20 trials.

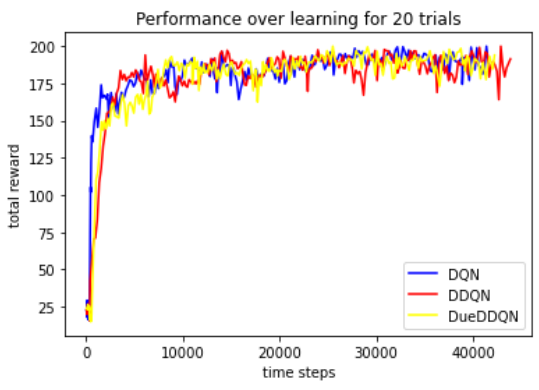
4. Comparison of the performances

   In this section we make a qualitative comparison of algorithms.

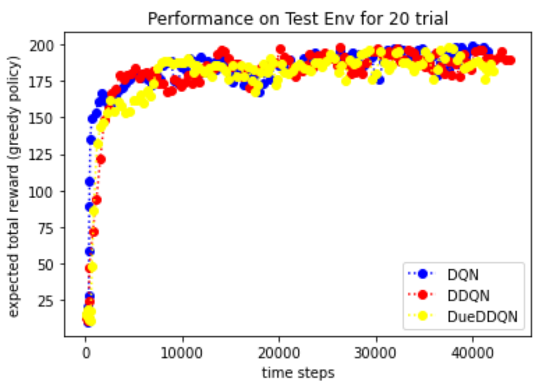
   Any figure shown above illustrates the performance of the DQN algorithm and its variants on the environment CartPolev0 (thanks to the gym package).

   The maximum total reward that agent can obtain is 200 and we can see that the trend is almost the same for algorithms DQN , DDQN et DueDDQN. The algorithms DDQN, DueDDQN are supposed to be more efficient than the standard DQN in general but in our case the difference is not significant.

   We could also put all graphs together as below :



(a) The cumulative reward over the learning process, $C = 20$



(b) The cumulative reward of the associated greedy policy on the test

Figure 7: Comparison of the algorithms

The Figure 7 shows the performance of the DueDDQN for 20 trials. As we said before the difference is not significant. I would have to play with a hyperparameter like C, reduce to the number of episodes (so time steps) and compare the performs since the total reward is roughly the same after 10000 time steps. Unfortunately, we have not explored these cases because of the time .

Conclusion

In this mission we reviewed the theoretical results of the course about approximate reinforcement learning (RL) algorithms. We also worked on some policy gradient results namely Trust Region Policy Optimization (TRPO) and the conservative policy iteration. In the experimental part we have seen that the DQN algorithms and its variants are good methods of approximate RL. Unless there was a programming error, the results I got were as good as I expected and sounded in accordance with the course. See you soon on the exploration part.