MAP670C - Reinforcement Learning

Safe and efficient off-policy reinforcement learning

Authors:
Bechir Trabelsi
Kais Cheikh

M2 Data Sciences

Academic year : 2020/2021

28 février 2021

1 Introduction

In this project we have implemented several algorithms for off-policy, return-based reinforcement learning, our goal is to compare these algorithms in terms of performance and complexity. Our work is based on the research paper **Safe and efficient off-policy reinforcement learning** [2] in which the authors derive from previous algorithms a novel algorithm, **Retrace(λ)**.
Our work consists in writing a python script allowing the reproduction of the numerical results demonstrated in the paper by simulating each of the mentioned algorithms on a game of Taxi-V3 from the gym python library, which is a toolkit for developing and comparing reinforcement learning algorithms.
In the first section of this report we shall present the algorithms we worked with and their mathematical formulation, while the second section will focus on the numerical results we obtained and their results.

2 Off-Policy Algorithms

2.1 Definitions

We will re-introduce the same notations and definitions as the paper [2].
Our problem is the interaction between an agent and a Markov Decision Process (MDP) defined by $(\mathcal{X}, \mathcal{A}, \gamma, P, r)$ where :

— $\mathcal{X}$ defines the state space (finite).

— $\mathcal{A}$ defines the action space.

— $\gamma \in [0,1)$ defines the the discount factor.

— P from $\mathcal{X} \times \mathcal{A}$ to a distribution over $\mathcal{X}$ a transition function mapping from a pair state-action (x,a) we associate the probability to arrive to the state $x' \in \mathcal{X}$. $P(x'|x,a) \in [0,1]$.

— $r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ define the reward for each state-action pair. $r \in [-R_{max}, R_{max}]$

We also define a policy π from $\mathcal{X}$ to a distribution over $\mathcal{A}$.
And a Q-function $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$. We note that the reward r is considered as a Q-function.
For each policy π we can define the operator P^π :

$$\left(P^{\pi} Q\right)(x, a) := \sum_{x' \in \mathcal{X}} \sum_{a' \in \mathcal{A}} P\left(x' \mid x, a\right) \pi\left(a' \mid x'\right) Q\left(x', a'\right)$$

and from that we introduce the operator Q^π as :

$$Q^{\pi} := \sum_{t \geq 0} \gamma^{t} \left(P^{\pi}\right)^{t} r.$$

We must also define the *Bellman operator* $\mathcal{T}^\pi$ as $\mathcal{T}^\pi Q := r + \gamma P^\pi Q$. Where the fixed point of it is :

$$\begin{aligned} \mathcal{T}^{\pi} Q = Q &= r + \gamma P^{\pi} Q \\ \Longleftrightarrow r &= (I - \gamma P^{\pi}) Q \\ \Longleftrightarrow Q &= (I - \gamma P^{\pi})^{-1} r \\ \Longleftrightarrow Q &= \sum_{t \geq 0} \gamma^{t} \left(P^{\pi}\right)^{t} r = Q^{\pi} \end{aligned}$$

And we can also define The *Bellman optimality operator* over all policies as :

$$\mathcal{T} Q = Q = r + \gamma \max_{\pi} P^{\pi} Q$$

We note Q^* it's fixed point and the solution for the "optimal control settings".
These two *Bellman operators* are the commonly used operators in the optimal

policy problems. However in the paper [2] they introduced the **Return-based Operators**, in fact these operators takes in consideration the exponential weighted sums of $n-$steps returns multiplied by a factor λ :

$$\mathcal{T}_\lambda^\pi Q := (1-\lambda)\sum_{n\geq 0}\lambda^n\left[(\mathcal{T}^\pi)^n Q\right] = Q + (I-\lambda\gamma P^\pi)^{-1}(\mathcal{T}^\pi Q - Q)$$

We shall note that the action $a_t \sim \mu(.|x_t)$ and the new state $x_{t+1} \sim P(.|a_t, x_t)$. Where μ is a behaviour policy.
In case $\pi = \mu$ we will be working on a on-policy basis. However in the general case, we will work on a off-policy situation.

For a chosen policy π with value Q, we will try to estimate using a behaviour policy μ using the Return-based operator defined by the equation (1).

$$\mathcal{R}Q(x,a) := Q(x,a) + \mathbb{E}_\mu\left[\sum_{t>0}\gamma^t\left(\prod_{s=1}^{t} c_s\right)\left(r_t + \gamma\mathbb{E}_\pi Q\left(x_{t+1},\cdot\right) - Q\left(x_t,a_t\right)\right)\right] \quad (1)$$

Where (c_s) are the *traces* of the operator. The c_s terms will be specified for 4 cases later on in the section 2.2.

2.2 Algorithms

To simulate the trade-off between exploration and exploitation we used an $\epsilon-$greedy approach combined with a decay factor. In fact, we sample uniformly a value in [0,1], then if it's :

— higher than ϵ : We **exploit** our environment and knowledge, meaning that we choose the highest probable action.

— lower than ϵ : We **explore** our environment nevertheless of our knowledge , meaning that we choose a random action.

This switch between exploration and exploitation allow the agent to learn about it's environment and to break free from local stationary. However with each episode we decrease the coefficient ϵ by multiplying it by a decay factor to focus on the exploitation phase since it contains the true knowledge of the agent.

All the four algorithms tackled in this work are based on the The general operator that we consider for comparing several return-based off-policy algorithms (1) as presented in [2], they however differ in the definition of the operator trace coefficients c_s :

— **Importance Sampling (IS) :** $c_s = \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)}$. It's based on the likelihood of the sequence. However it's highly unstable if π and μ are far from each other because the ratio will vary a lot.

— **Off-policy $\mathbf{Q}^{\pi}(\lambda)$ and $\mathbf{Q}^{*}(\lambda)$:** $c_s = \lambda$ It helps to avoid the exploding/vanishing generated, in the case of IS, by the multiplication of the ratio of $\frac{\pi}{\mu}$.

— **TB**(λ) **:** $c_s = \lambda\pi(a_s|x_s)$. It's helpful in the case that π and μ are far from each other.

— **Retrace**(λ) **:** $c_s = \lambda \min(1, \frac{\pi(a_s|x_s)}{\mu(a_s|x_s)})$ It takes the advantages of the 3 previous algorithms. In fact, it has less variance compared to IS because of the utilization of the min. And has the advantage of TB and Off-policy since it can be used on any π and μ.

2.3 psuedo-code

Algorithm 1: Simulation algorithm

```
Input  : Environment, number_of_episodes, max_steps, decay_rate,
         $\lambda, \gamma, \epsilon$
$N_a, N_c$ : we get the number of action and states from the Environment.
$r \leftarrow \emptyset$
$Q_t \leftarrow 0$, a null matrix
$Game\_over \leftarrow False$
for $episode = 1$ to $number\_of\_episodes$ do
    $S \leftarrow$ initial state
    for $step = 1$ to $max\_steps$ do
        $t \sim \mathcal{U}(0,1)$
        if $t > \epsilon$ then
            $A^* \leftarrow \underset{\mathcal{A}}{\operatorname{argmax}}(Q_t(S))$ get the best action.
            $\mu = 1 - \epsilon$
        else
            $A^* \sim \mathcal{U}(\mathcal{A})$
            $\mu = \frac{\epsilon}{N_a - 1}$
        $S, r, Game\_over \leftarrow$ Environment after doing the action $A^*$
        Compute $\pi$
        $c_{step} \leftarrow function(\lambda, \mu, \pi)$ based on the examples seen in 2.2.
        $Q_t \leftarrow Q_t + \gamma^{step}(\prod_{s=1}^{step} c_s)(r + \gamma \sum_a \pi(a) Q_t(a) - Q_t)$
        if $Game\_over = True$ then
            break
        end
    end
    $\epsilon \leftarrow \epsilon \times$ decay_rate
end
```

3 Experiments

3.1 Game : Taxi

This task was introduced in [1],illustrating some issues in hierarchical reinforcement learning. There are 4 locations (labeled by different letters) and the goal is to pick up the passenger at one location and drop him off in another. The reward is +20 points for a successful dropoff, and -1 point for every timestep it takes. There is also a 10 point penalty for illegal pick-up and drop-off actions.

3.2 Simulations

We implemented the algorithm described above and we tried different settings to observe it's performance.
In the first part we did 10 simulation using all 4 algorithms. The figure (1) represent the average of the reward on the number of simulation done. We ca observe that

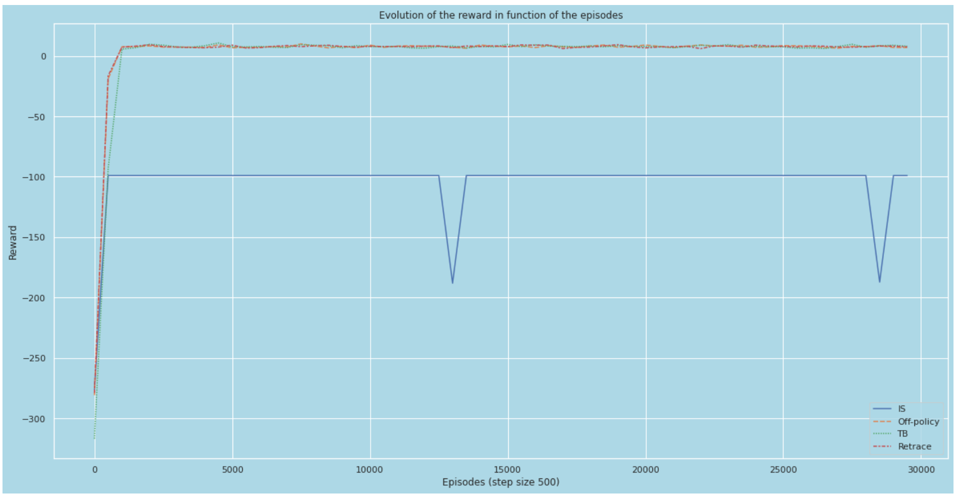

Fig. 1: The average of the evolution of the reward in function of episodes

the IS algorithm is highly unstable compared to the other 3 and converge to a lower reward compared to the others. We can also see that the algorithm converge eventually to the same reward ≈ 8.

And to compare the stability of these algorithms, We observed the evolution of the average of $|r_{t+1} - r_t|$ on the different simulation over the episodes presented in the figure (2).

We can see that the algorithm IS is unstable compared to the other 3 algorithms.

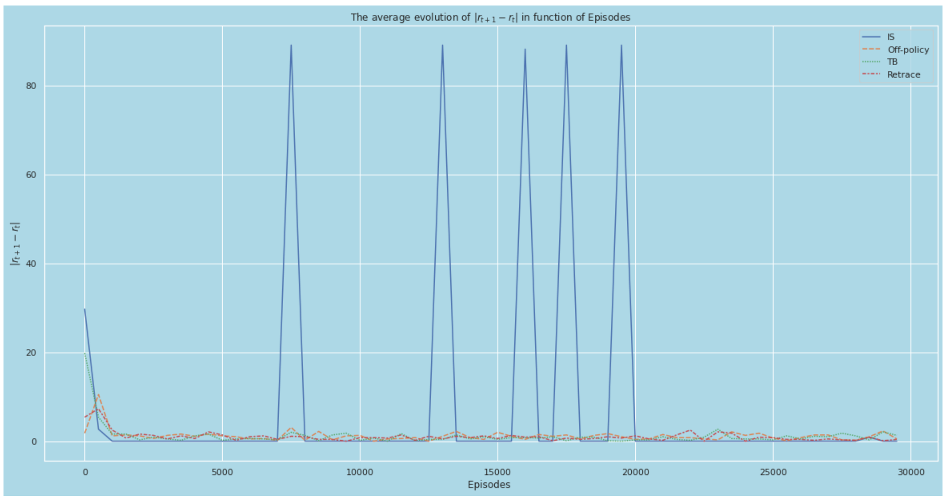

Fig. 2: the evolution of the average of $|r_{t+1} - r_t|$ on the different simulation over the episodes

We wanted to observe the impact of λ on the Retrace algorithm, the figure (3) present the evolution of the average of reward done over 10 simulation. For $\lambda = 0$,

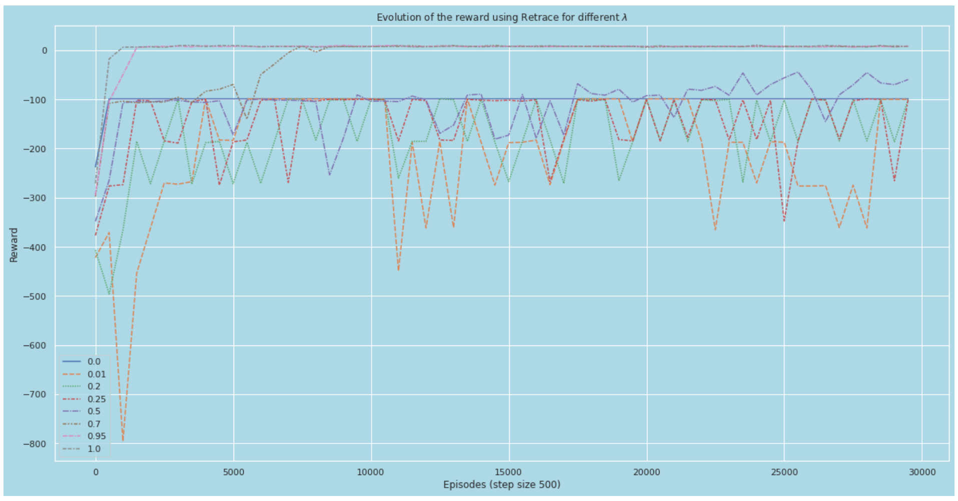

Fig. 3: The average of the reward for different λ

we have a fast convergence and high stability but it converged to a small reward. As We increased the value of λ we have more instability and they kept oscillating on the sale reward. However with higher values of λ we got a faster convergence to a higher reward.

3.3 Summary

We summarize the results from the numeical experiments in the following table from [2] :

	Definition of c_s	Estimation variance	Average reward $\pm$ std	min reward	max reward	Execution time (seconds)
Importance sampling	$\frac{\pi(a_s\|x_s)}{\mu(a_s\|x_s)}$	High	-199.238 ± 283.613	-1050	-99.292	82.673
$Q^{\pi}(\lambda)$	λ	Low	10.691 ± 15.206	5.518	56.310	12.587
TB (λ)	$\lambda\pi\left(a_s \mid x_s\right)$	Low	8.092 ± 11.500	4.120	42.587	13.083
Retrace (λ)	$\lambda \min\left(1, \frac{\pi(a_s\|x_s)}{\mu(a_s\|x_s)}\right)$	Low	10.695 ± 15.190	5.514	56.267	12.879

Tab. 1: Summary of numerical results : first column : Algorithm used, second column : Definition of c_s, third column : Variance of the estimation, fourth column : Average reward and its standard deviation, fifth and sixth columns : minimum and maximum rewards, seventh column : Execution time of each algorithm in seconds.

In accordance with the results presented in [2], we find that the Importance sampling algorithm is unstable with high variance and high standard deviation, the Off-Policy, Tree-backup and Retrace algorithm behave in a more stable way giving consistent results on each simulation, with the Retrace algorithm having an overall better performance.

4 Conclusion

In this work we implemented the different algorithms tackled in [2], we designed a simulation environment based on the game Taxi presented in [1], this environment allowed us to simulate the four algorithms and to run several numerical experiments to draw comparison between them in terms of performance and efficiency. Our results are in accordance with the findings in [2]. The Retrace(λ) is proven to be a well performing algorithm with fair stability .

Références

[1] Thomas G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13 :227–303, 2000.

[2] Rémi Munos, Thomas Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and efficient off-policy reinforcement learning. *Advances in Neural Information Processing Systems*, (Nips) :1054–1062, 2016.