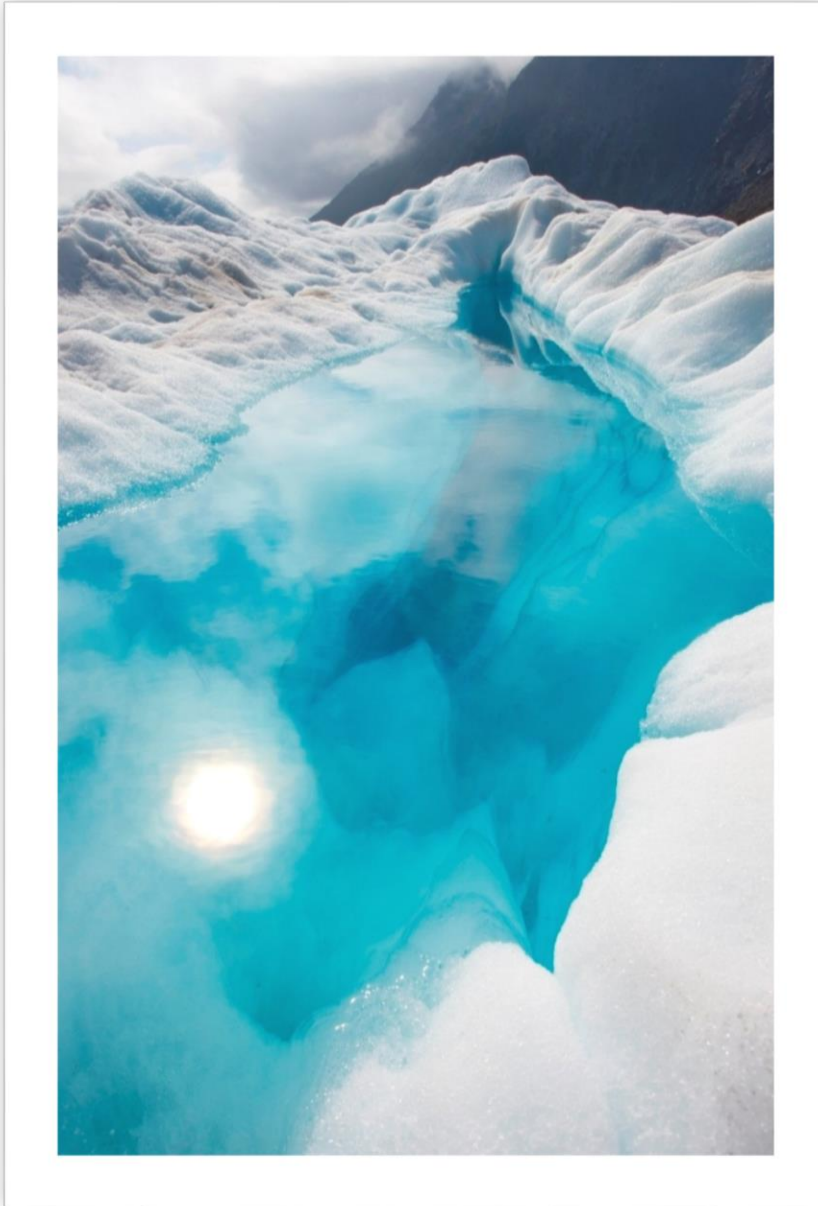


Frozen Lake

SARSA(LAMBDA) & N-STEP SARSA

Fateme Noorzad | ML | Azar 1398

Part1: Solving frozen lake problem using n-step SARSA:

In this part n-step SARSA is implemented using the below pseudo code:

- Initialize Q(s,a) arbitrary for all s in state space, and a in action space.
- Initialize π to be ε-greedy with respect to Q
- Algorithm parameters are: $0 < \alpha \leq 1$ & small $\varepsilon > 0$ & a positive integer n
- Loop for each episode:
    - Initialize and store $S_0 \neq$ terminal
    - Select and store an action A_0 using policy π
    - $T \leftarrow \infty$
    - Loop for t=0, 1, 2,... :
        - If t<T, then:
            - Take action A_t
            - Observe and store the next rewards as r_{t+1} and the next state as S_{t+1}
            - If S_{t+1} is terminal, then:
                - $T \leftarrow t+1$
            - Else:
                - Select and store an action A_{t+1} using policy π
        - $\tau \leftarrow t - n - 1$
        - if $\tau \geq 0$:
            - $R = \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} r_i$
            - if $\tau+n < T$, then :
                - $R \leftarrow R + \gamma^n Q(S_{\tau+n},A_{\tau+n})$
            - $Q(S_\tau,A_\tau) \leftarrow Q(S_\tau,A_\tau) + \alpha [R - Q(S_\tau,A_\tau)]$
            - If π is being learnt, then ensure that policy of each state is greedy with respect to Q
    - Until $\tau = T - 1$

For defining T to be infinite "sys.maxsize" is used. Algorithm parameters are set as:

$\gamma = 1$, $\alpha = 0.1$, $\varepsilon = 1/$(number of episodes until now)

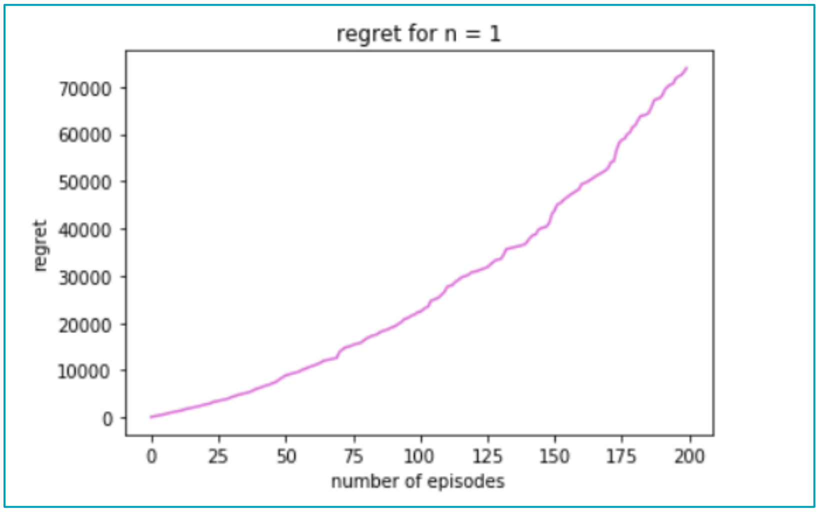
Regret is defined as:

Regret = (max reward of an episode until now)*(number of episodes until now) – (total rewards until now)

So regret of each episode is found.

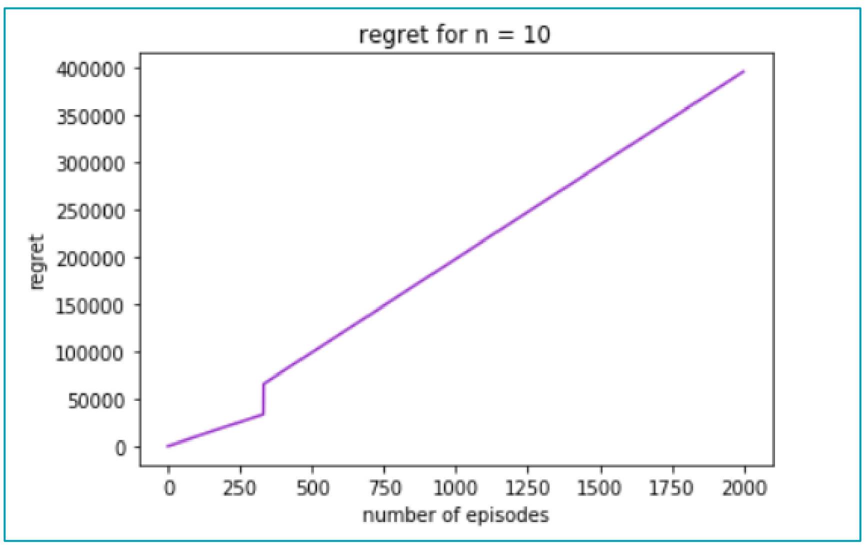
Using all of the above explanation a class called n-step SARSA is created.

For the first plot n is one, since sum of rewards are negative, they are added to the first term, resulting in an increasing regret.

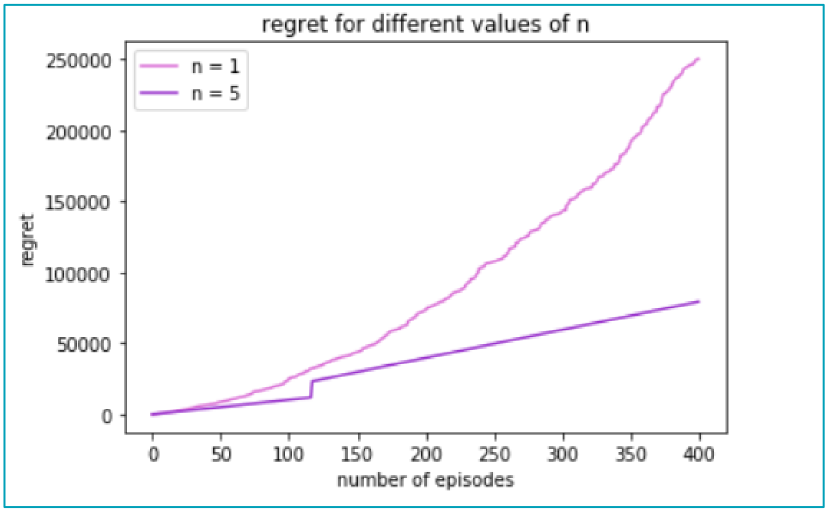


The number of episode is 200, since it takes a long time to calculate regret for more than 200 episodes. This means it for smaller values of n calculating takes a longer time. (The reason for it is explained in the next part.)

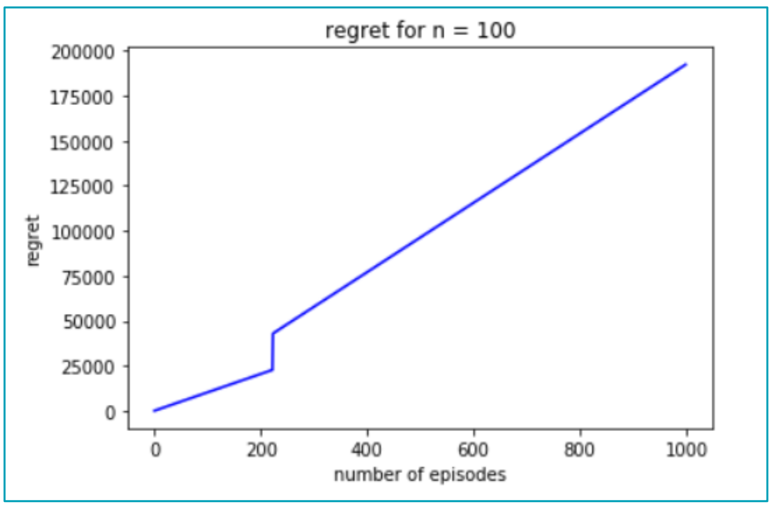
The next value for n is 10.



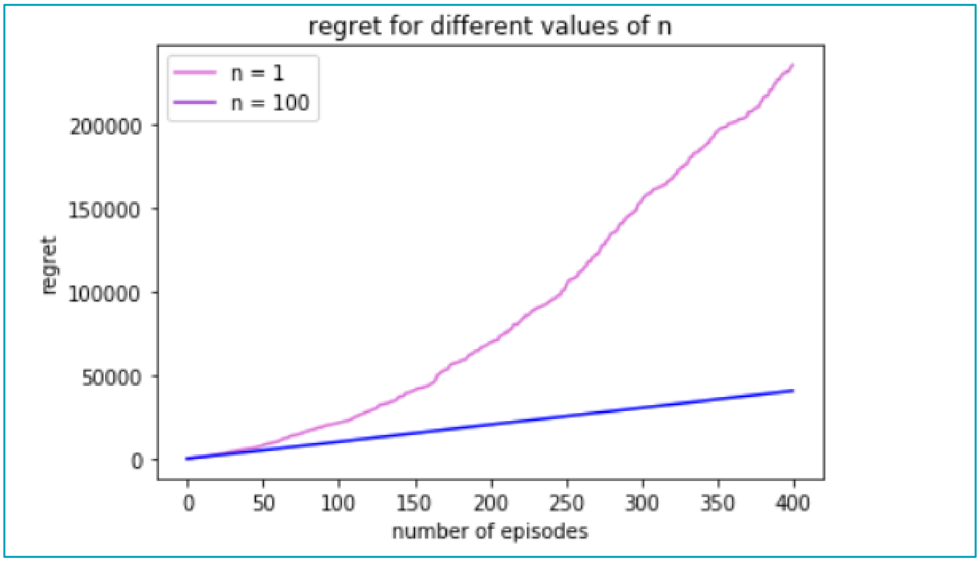
The jump is because before that episode maximum reward was 0 and after that it became a number near 100. The other thing that needs to be mentioned is that regret became more linear in with this value of n which is a good thing. Next page's plot gives a clearer view of the difference of regret and convergence for n=1 and n=5.

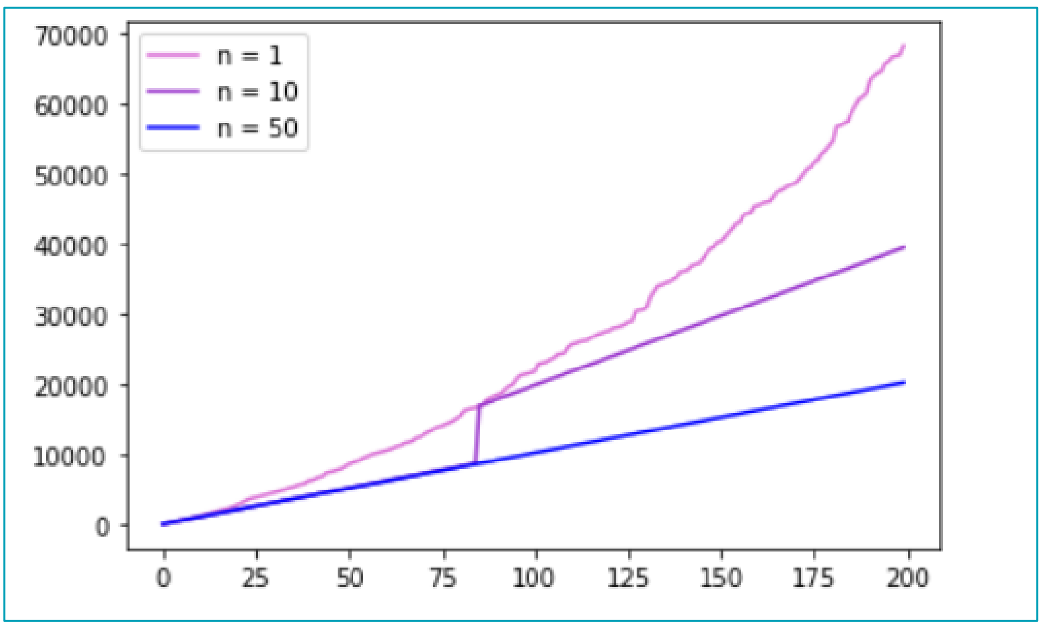


After that for n=100 regret is plotted. The learning curve will be:



The 2 plots below can give a better vision on how results are improved when n grows larger and larger:





Part 2: Comparing convergence rate:

In n-step SARSA intermediate values of n perform better than the 2 extremes. It means n=1 and n=infinity are not as good as the ones in between. As can be seen in the above plot, n=1 reaches to a specific regret sooner than the other 2.

In n steps which n is larger than 1, learning is delayed by n steps, because n other rewards must be seen in order to update discounted return. This also means the last n rewards. States and actions must be stored. This needs a pretty good memory when n is very large. But per step computation is small that's why regret was found sooner for larger n.

So choosing n is a trade-off.

When n is 1, the agent is myopic, since he only uses the next states' reward as discounted return. But for larger values of n since other rewards are added, the results are more accurate and better. For n larger than, the agent becomes more and more far-sited.

Using a timer, it took 111.17s for n=1 to learn in 400 episode while it took 0.33s for n=5 and 0.51s for n=100.

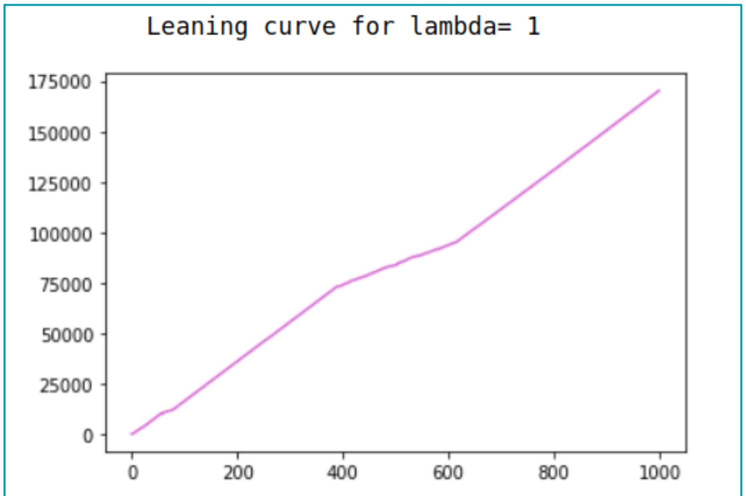
Part 3: SARSA(λ):

This algorithms is implemented using pseudo code below:

- Initialize Q(s,a)=0 and e(s,a)=0 for all s in state space and action space
- Repeat for each episode:
    - Select a based on a soft Q-based policy.
    - Take a and observe r_{t+1} and S_{t+1}
    - Select A_{t+1} in S_{t+1} using a soft Q-based policy
    - $\delta \leftarrow r + \gamma\, Q(S_{t+1}, A_{t+1}) - Q(S_t,A_t)$
    - $e(S_t,A_t) \leftarrow e(S_t,A_t) + 1$
    - for all s and a so far in episode:
        - $Q(s,a) \leftarrow Q(s,a) + \alpha\, \delta\, e(s,a)$
        - $e(s,a) \leftarrow \gamma\, \lambda\, e(s,a)$
    - Until s is terminal
    - e = 0

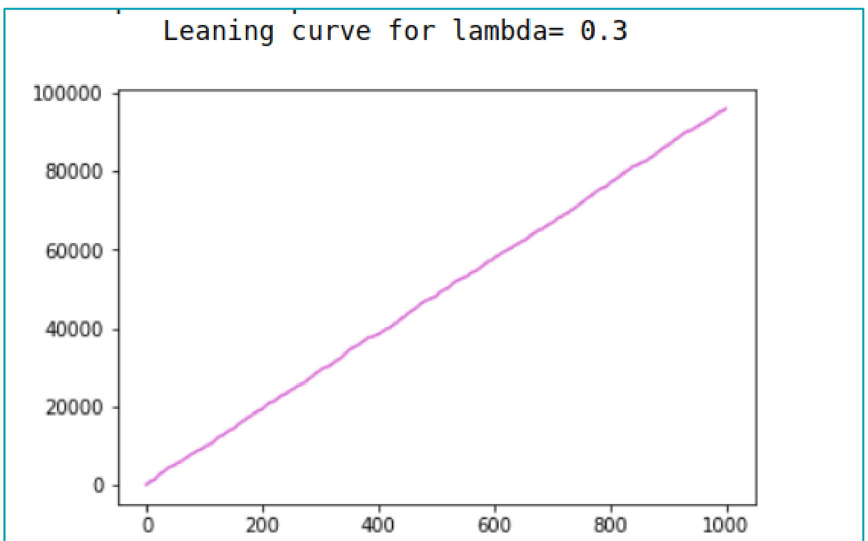
Using the above pseudo code, the SARSA(λ) algorithm is implemented.

For λ = 1 :



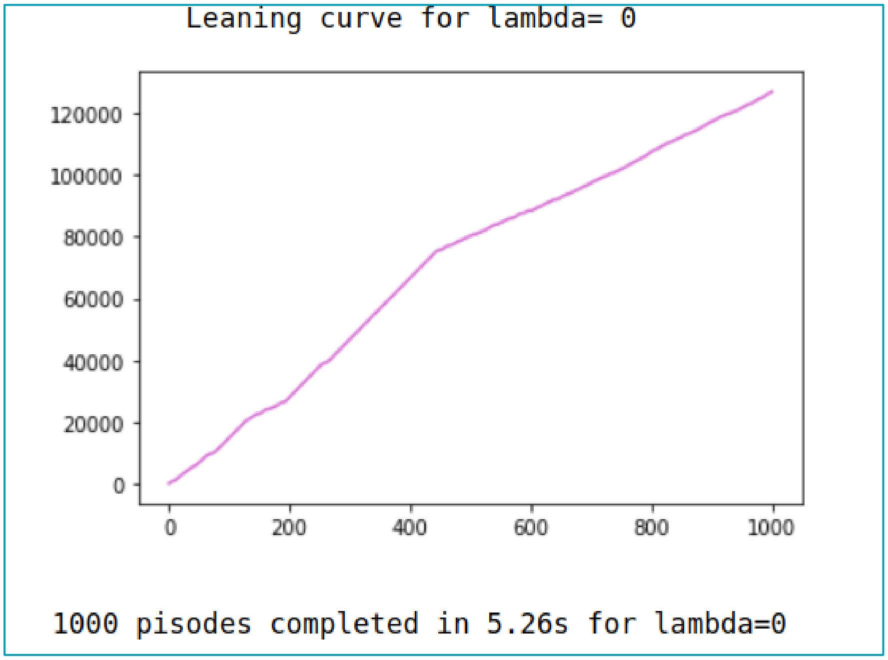
1000 pisodes completed in 5.25s for lambda=1

For λ = 0.3:



1000 pisodes completed in 5.24s for lambda=0.3

For $\lambda = 0$:



By looking at the vertical axis, it shows that $\lambda = 0.3$ is between $\lambda = 0$ and $\lambda = 1$ which are the 2 extremes. Based on the time it takes to finish the learning in 1000 episodes, it can be said in different conditions, which of these values perform better.