Abstract

Monte Carlo :

a way to learn from raw experience

1. On-policy agents attempt to evaluate or improve the same policy that is used to make decisions.
   - First-visit MC estimates the value of state *s* by averaging the returns observed after first visit to *s*.

     (Implemented ✓)

     $$Q(s,a) = \frac{\sum_{i=1}^{n} G_i(s)}{n}$$

   - Every-visit MC estimates the value of state *s* by averaging the returns observed after all visits to *s*.

2. Off-Policy agents explore and generate behaviour on one policy (*behaviour policy*) and is aimed at becoming optimal on another policy(*target policy*).

   $$Q_{n+1}(s,a) \leftarrow Q_n(s,a) + \frac{W_n}{C_n}\left[G_n - Q_n(s,a)\right] \qquad n \geq 1$$

   $$C_0 = 0 \qquad C_{n+1} = C_n + W_n$$

   An advantage of this separation is that the estimation policy may be deterministic, while the behaviour policy can continue to sample all possible actions.

Temporal Difference :

a way to learn from raw experience using bootstrapping

TD updates a guess towards a guess and revise the guess based on real experience

1. SARSA agents take one step from one *state-action* value pair to another *state-action* value pair and along the way collect reward *R*.

   $$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\Big[\underbrace{R + \gamma Q(S_{t+1},A_{t+1})}_{target} - Q(S_t,A_t)\Big]$$

   SARSA is an on-policy method. SARSA agents take action based on policy π (ε-greedy to ensure exploration as well as greedy to improve the policy)

2. Double Q-learning agents use two estimators instead of using one set of data and one estimator. This means that instead of using one Q-Value for each *state-action* pair, we should use two values – Q_A and Q_B.

   $$Q^{A(B)}(s,a) \leftarrow Q^{A(B)}(s,a) + \alpha\Big[R + \gamma Q^{B(A)}(s',a) - Q^{A(B)}(s,a)\Big]$$

   we use double Q-learning because Q-Learning performs very poorly in some stochastic environments.(the poor performance is caused by large overestimation of action values due to the use of Max $Q(s',a)$)

3. Tree Back-up agents add the estimated values of the action nodes on each of the states to the the discounted rewards it received along the path as well as with the bottom nodes' values.

   $$G_{t:t+n} = R_{t+1} + \gamma^n \sum_{a \neq A_{t+1}} \pi(a|s_{t+n}) Q_{t+n-1}(s_{t+n},a) + \gamma\pi(A_{t+1}|S_{t+1}) G_{t+1:t+n}$$

   $$Q_{t+n}(s,a) = Q_{t+n-1}(s,a) + \alpha\Big[G_{t:t+n} - Q_{t+n-1}(s,a)\Big]$$

4. Two-Step Expected SARSA agent update the action values based on the last *n* actions of the sequence of actions and weight the probabilities of each action to happen under the policy.

   $$G_{t:t+n} = R_{t+1} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_{a} \pi(a|s_{t+n}) Q_{t+n-1}(s_{t+n},a)$$

   $$Q_{t+n}(s,a) = Q_{t+n-1}(s,a) + \alpha\Big[G_{t:t+n} - Q_{t+n-1}(s,a)\Big]$$

Advantages of TD methods over Monte Carlo methods :

- TD can learn in every step online or offline
- TD can learn from the incomplete sequence
- TD can work in non-terminating environments (continuing)
- TD has a lower variance compared to MC as depends on one random action, transition, reward
- Usually more efficient than MC
- TD exploits Markov property and thus more effective in Markov environments

---

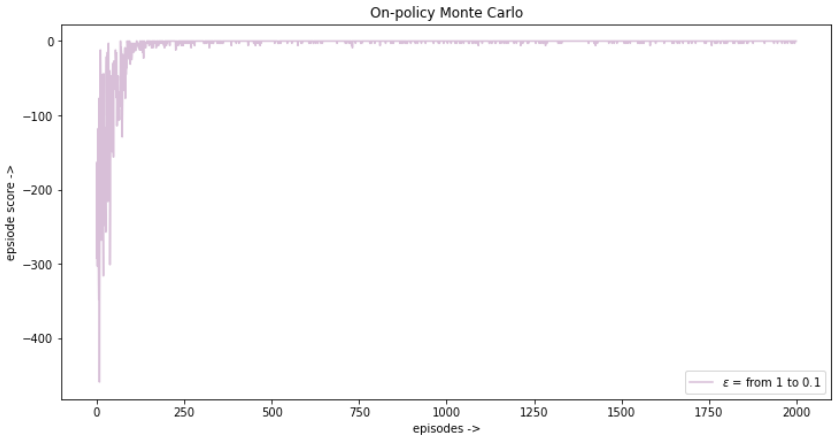
Results

Monte Carlo :

1. ***On-policy MC - First Visit*** : To implement this algorithm, I used both *decaying epsilon* and *constant epsilon* . Because this MAZE is a continuous task, we need to define a step limit for Monte Carlo to truncate each episode.

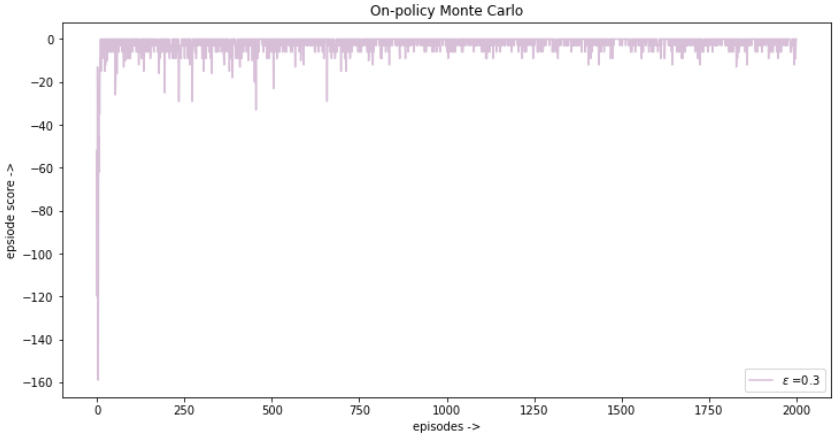
   Although the agent may not reach the island, shortening the episodes does not cause significant error, because $\gamma < 1$, and as the number of steps increase, $\gamma^{steps} \to 0$.

- **Decaying epsilon (** $\varepsilon : 1 \to 0.1 - decay\ factor : 0.99 - each\ two\ episodes$) :

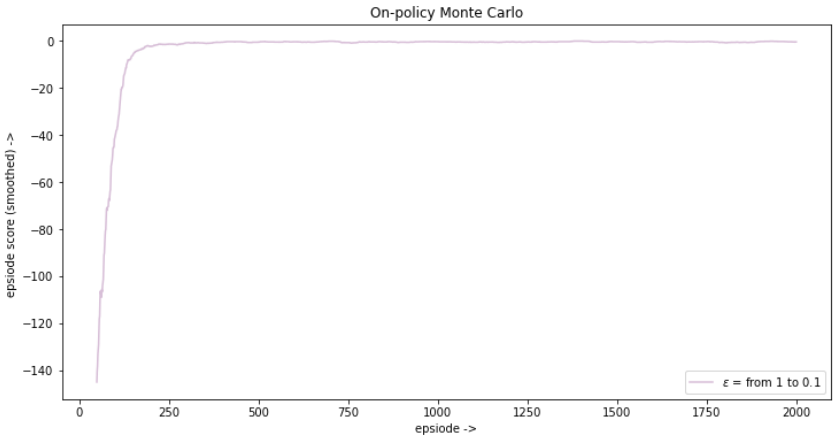


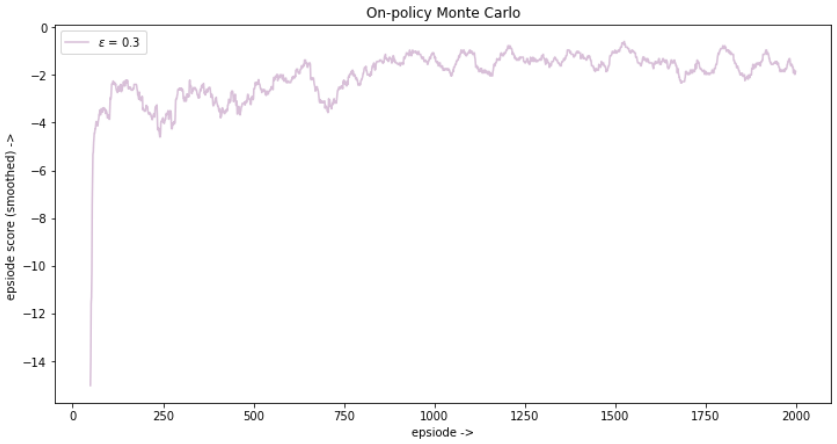
Although many fluctuations are observed at first, as the epsilon decreases (policy becomes more greedy), the plot becomes almost stable.

- **Constant epsilon (** $\varepsilon : 0.3$**)** :



To have better visualization, I also plotted *reward/episode* using sliding window with size 50





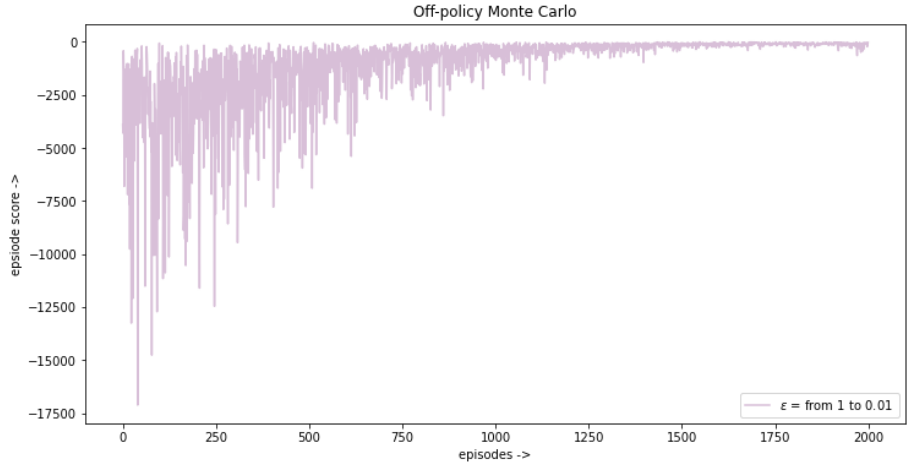
Although at first the reward is more negative in the *decaying epsilon* figure, it reaches stability soon, but in *constant epsilon* figure, we don't see any stability even after 20000 episodes.

In general, it is obviouse that the regret is much less when using decaying epsilon.

In *decaying epsilon* case, we reach the average reward of 0 which is the best reward we can get.

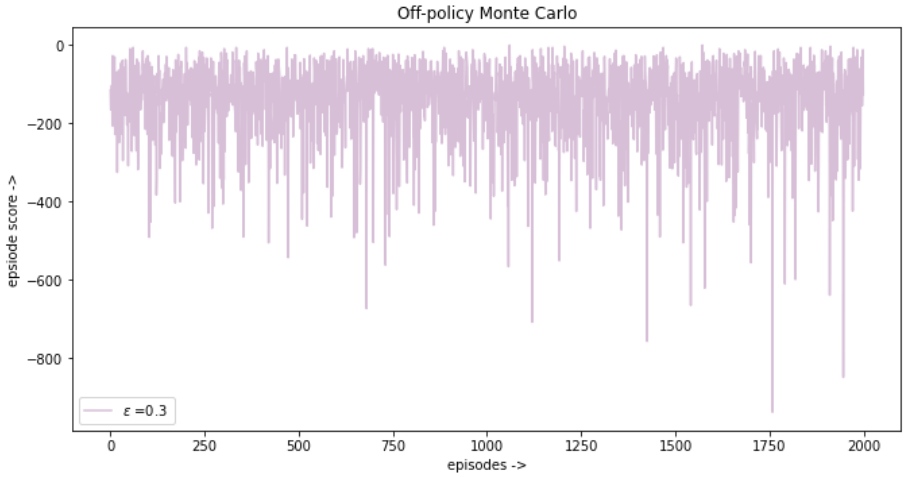
2. **Off-policy MC - Weighted Importance Sampling**:

- Target Policy: Greedy Policy
- Behavior Policy : ε-greedy Policy
  - Decaying epsilon ($\varepsilon : 1 \rightarrow 0.01 - decay\ factor : 0.99 - each\ two\ episodes$):



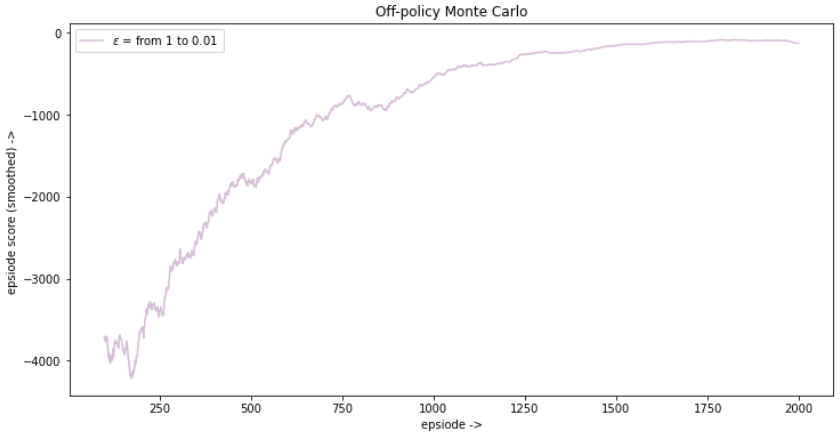
By reducing ε per episode, our b*ehavior policy* is gradually approaching our *target policy* that will allow us to finally stabilize after about 1500 episodes.

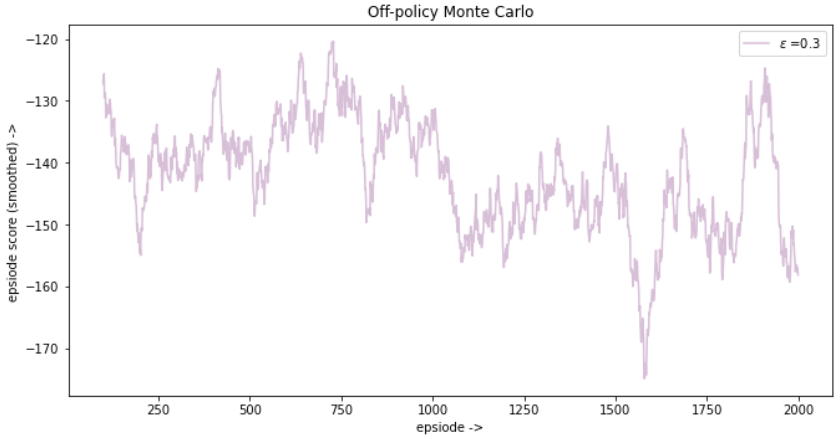
  - Constant epsilon ($\varepsilon : 0.3$):



As can be seen from the figure, the agent never stabilizes because it can't correctly follow the *target policy*.

To have better visualization, I also plotted *reward/episode* using sliding window with size 100.





In general, it is obviouse that the regret is much more when using off-policy methods, and also the convergence rate is much slower

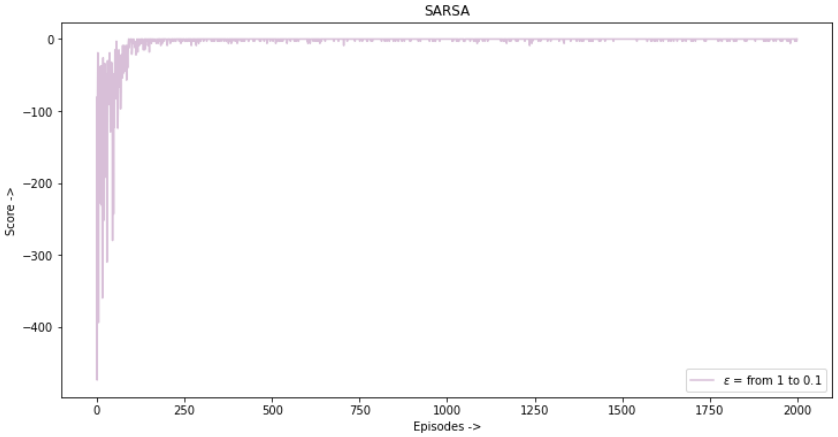
(We can use *Per-decison Importance Sampling* to reduce the regret. Also we can use *On-Policy Methods* to initialize our $Qvalue$ - off policy methods are sensitive to initial values.)

In neither case can we reach the best possible reward.

Temporal Difference :

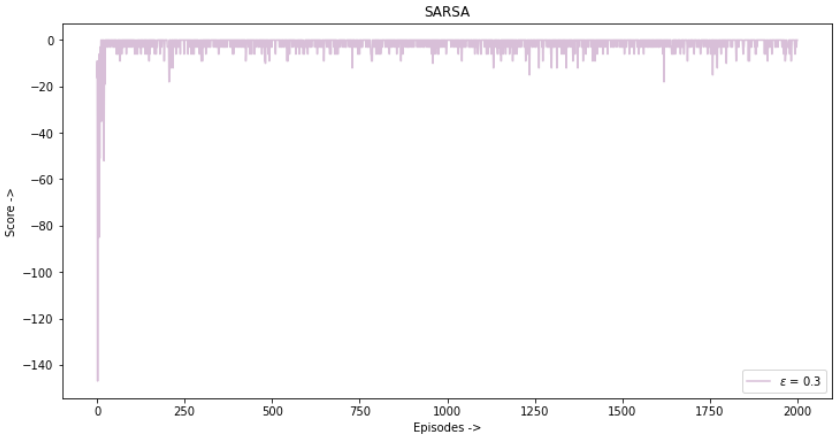
1. **SARSA** : To implement this algorithm, I used both decaying epsilon and constant epsilon .

- **Decaying epsilon (** $\varepsilon : 1 \rightarrow 0.1 - \textit{decay factor :} 0.99 - \textit{each two episodes}$) :

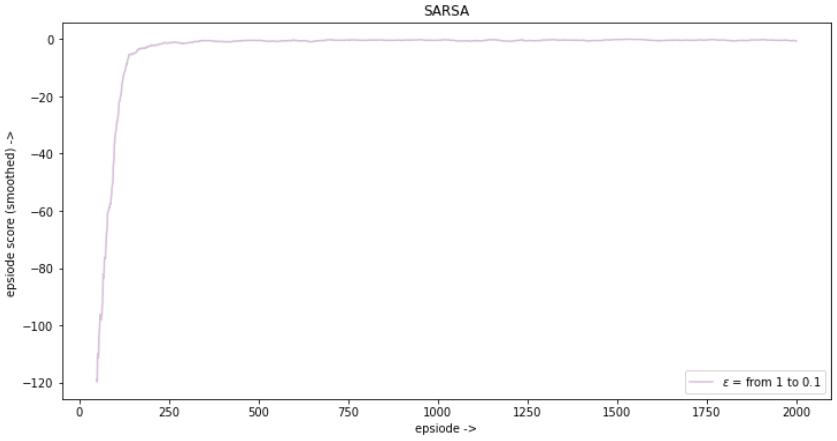


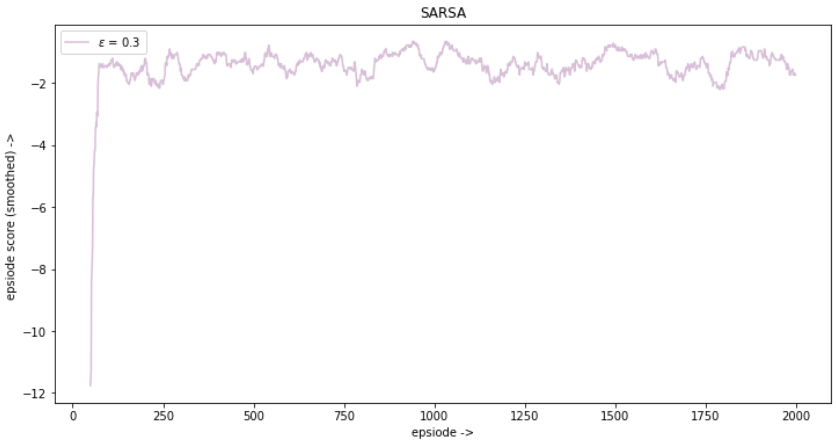
Although many fluctuations are observed at first few episodes, as the epsilon decreases (policy becomes more greedy), the plot soon becomes stable.

- **Constant epsilon (** $\varepsilon : 0.3$**)** :



To have better visualization, I also plotted *reward/episode* using sliding window with size 50





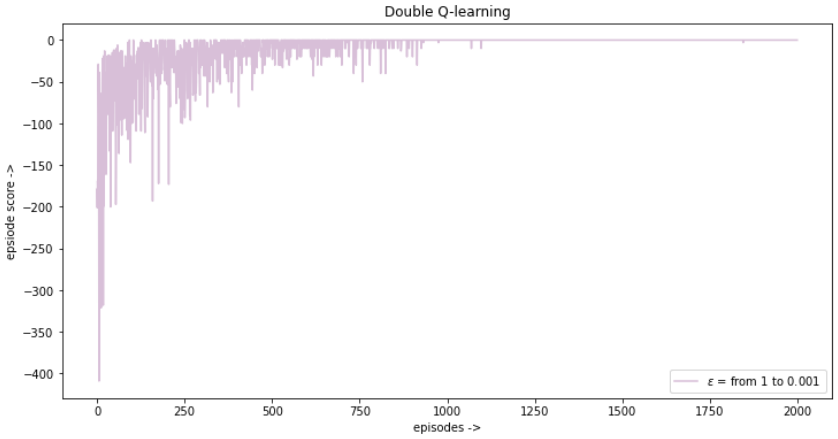
Although at first the reward is much more negative in the *decaying epsilon* figure, it reaches stability soon(decaying ε : episode #400 - sooner than *On-Policy MC*), but in the *constant epsilon* figure convergance doesn't happen in 2000 episodes.

In general, it is obviouse that the regret is much less when using decaying epsilon.

In *decaying epsilon* case, we reach the average reward of 0 which is the best reward we can get.

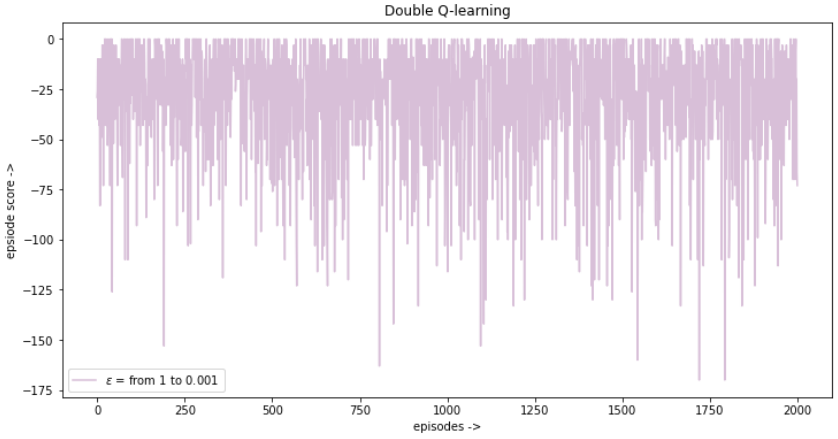
2. **Double Q-learning** :

- **Decaying epsilon (** $\varepsilon : 1 \rightarrow 0.01 - decay\,factor: 0.99 - each\,two\,episodes$) :



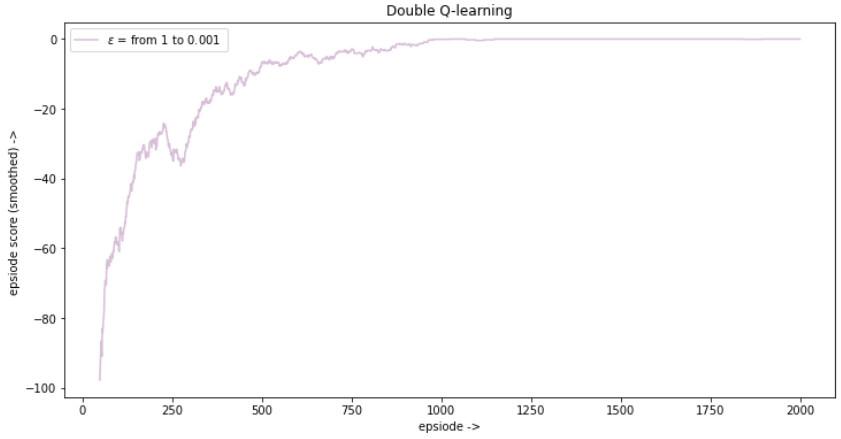
Although fluctuations are visible up to the middle of the figure, it stabilizes well after sufficient reduction of the epsilon.

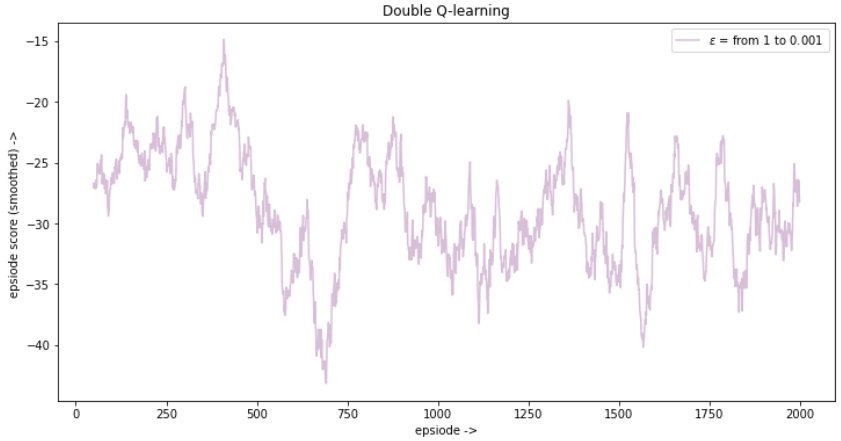
- **Constant epsilon (** $\varepsilon : 0.3$**)** :



Note : Unlike the previous two policies, this policy did not converge with $\varepsilon = 0.1$ (Even after 4000 episodes) and I had to reduce it to 0.001.

To have better visualization, I also plotted *reward/episode* using sliding window with size 50





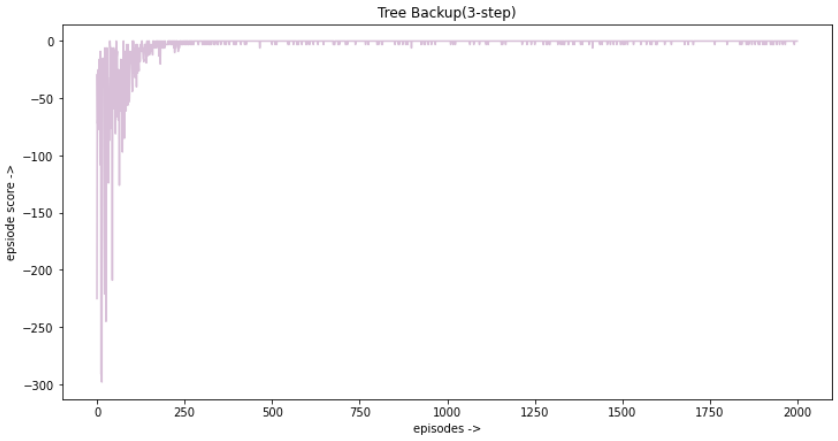
Although at first the reward is less negative in this policy, It stabilizes much later than two previous policies(decaying ϵ : episode #1250), but in the *constant epsilon* figure convergance does not occurin 2000 episodes.

In general, it is obviouse that the regret is much less when using decaying epsilon.

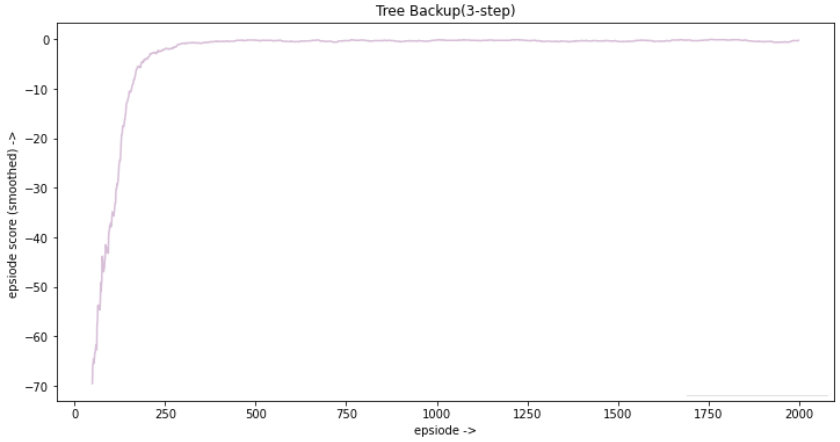
In *decaying epsilon* case, we reach the average reward of 0 which is the best reward we can get.

3. **Tree backup** :To implement this algorithm, I used constant epsilon as my fixed policy and $n = 3$

   - **Constant epsilon (** $\varepsilon : 0.3$**)** :

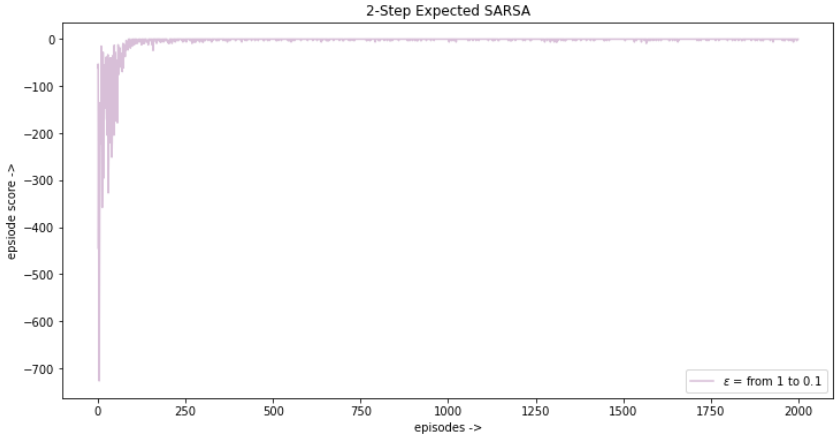


   To have better visualization, I also plotted *reward/episode* using sliding window with size 50



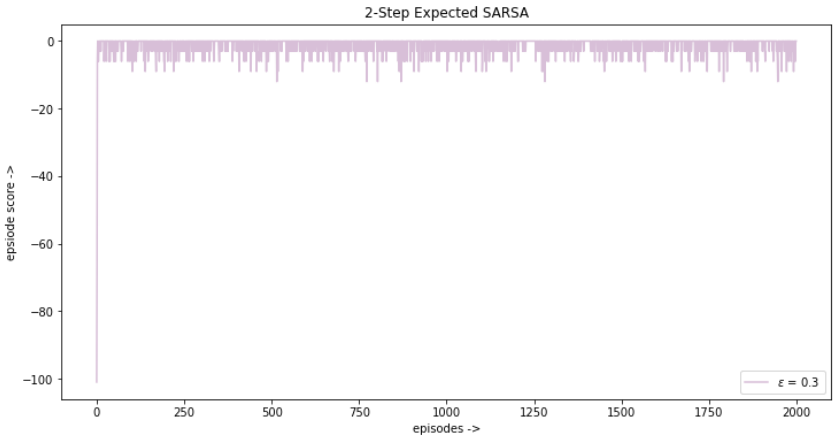
4. **2-Step Expected SARSA**: To implement this algorithm, I used both decaying epsilon and constant epsilon .

   - **Decaying epsilon (** $\varepsilon : 1 \rightarrow 0.1 - \textit{decay factor :} 0.99 - \textit{each two episodes}$**)** :

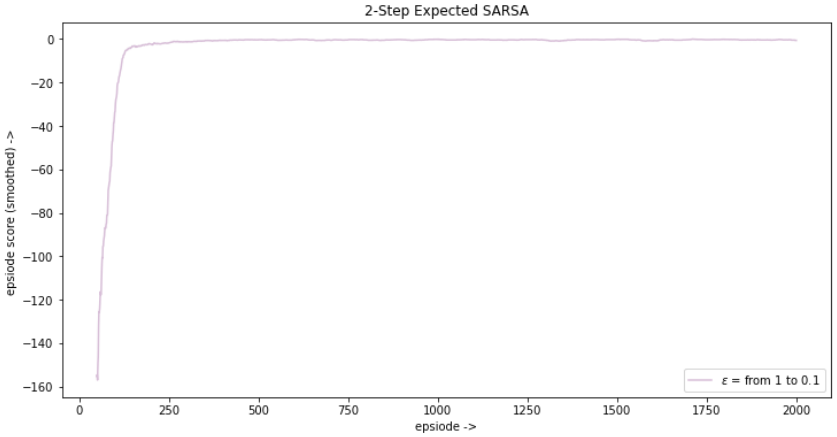


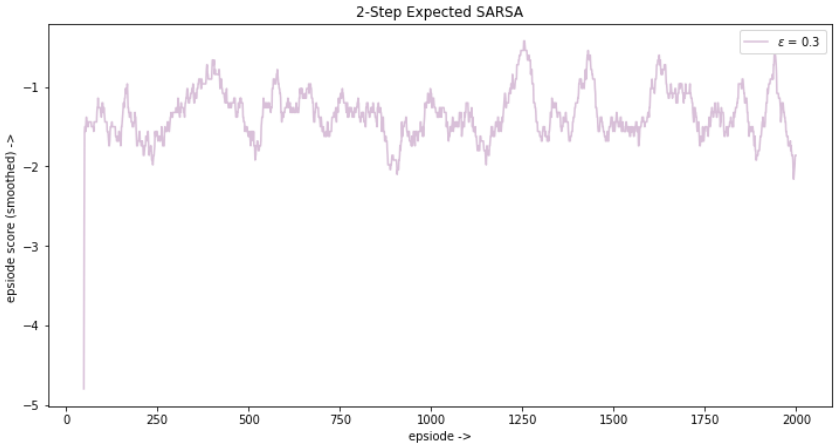
   `Although many fluctuations are observed at first few episodes, as the epsilon decreases (policy b ecomes more greedy), the plot soon becomes stable.<br><br>`

   - **Constant epsilon (** $\varepsilon : 0.3$ **)** :



To have better visualization, I also plotted *reward/episode* using sliding window with size 50



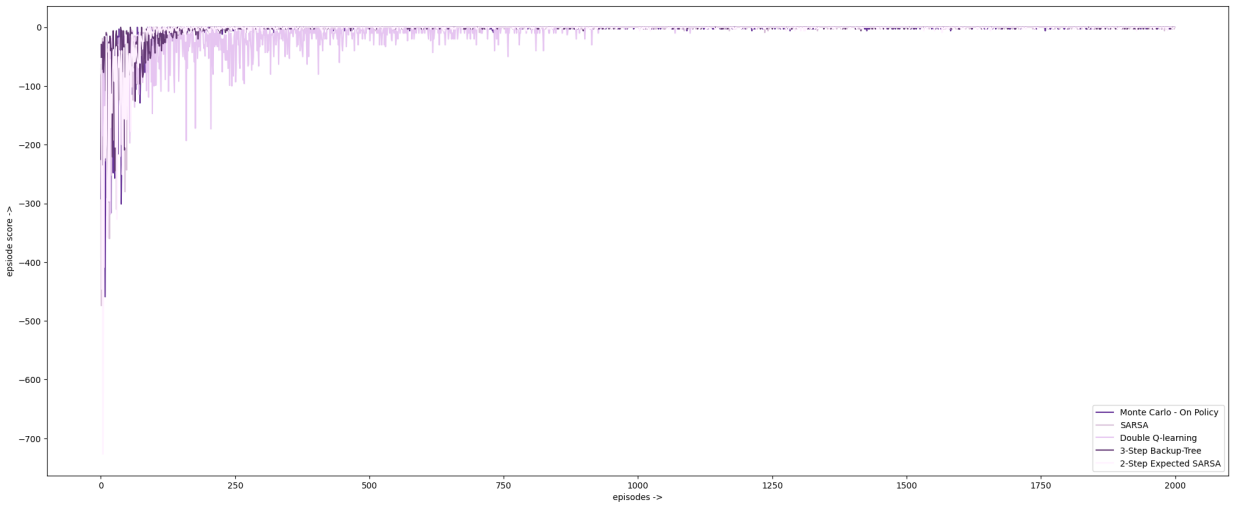


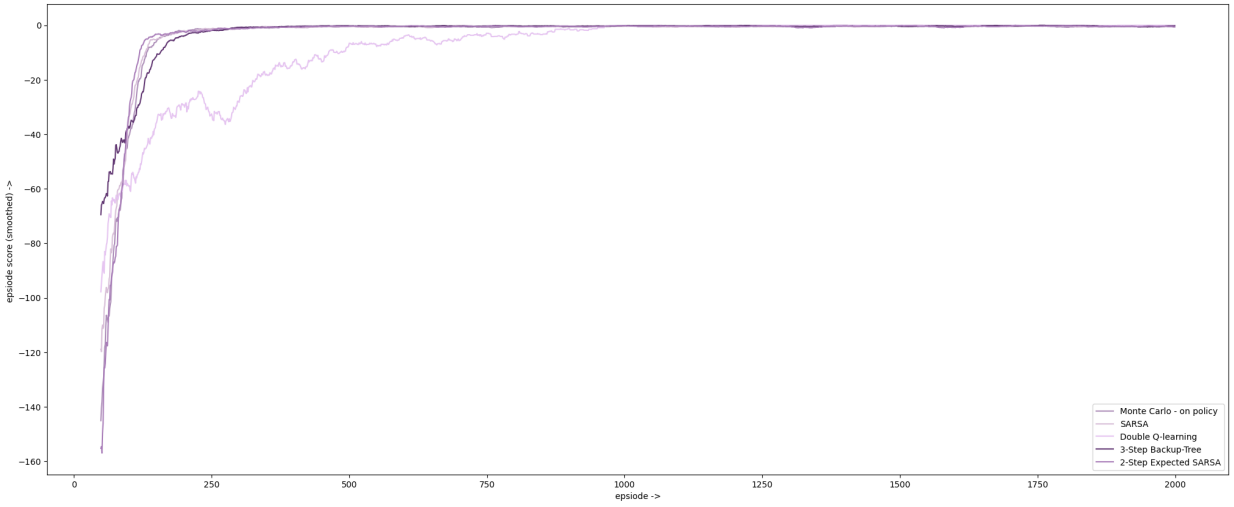
Although at first the reward is much more negative in the *decaying epsilon* figure, it reaches stability soon(decaying ϵ : episode #??), but in the *constant epsilon* figure convergance doesn't happen in 2000 episodes.

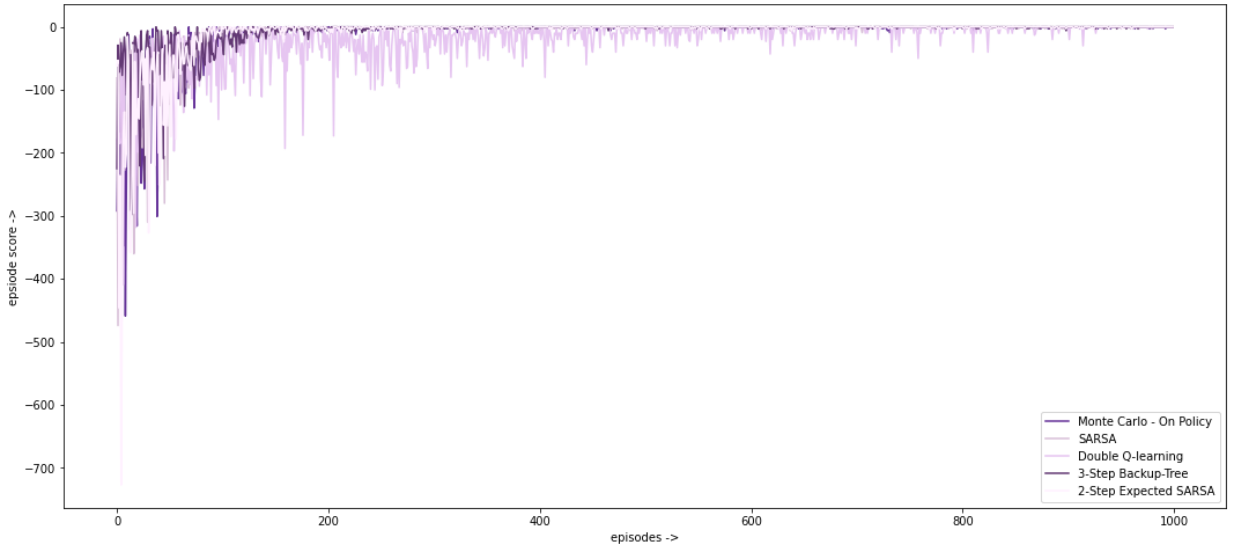
In general, it is obviouse that the regret is much less when using decaying epsilon.

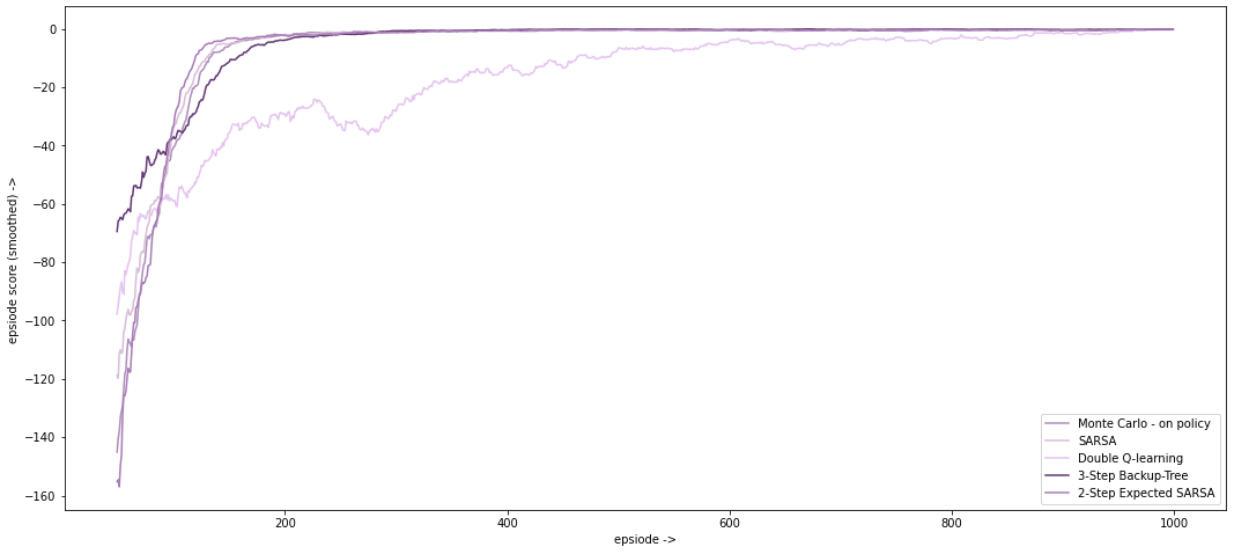
In *decaying epsilon* case, we reach the average reward of 0 which is the best reward we can get.

Conclusion


epsiode score ->
episodes ->
Monte Carlo - On Policy
SARSA
Double Q-learning
3-Step Backup-Tree
2-Step Expected SARSA


episode score (smoothed) ->
epsiode ->
Monte Carlo - on policy
SARSA
Double Q-learning
3-Step Backup-Tree
2-Step Expected SARSA


epsiode score ->
episodes ->
Monte Carlo - On Policy
SARSA
Double Q-learning
3-Step Backup-Tree
2-Step Expected SARSA



As can be clearly seen from the figures, the convergence rate of **Q-learning** is very low.

2-step expected SARSA have the highest rate of convergence, but 3-step treebackup has the least amount of regret because it receives less punishment from the environment in it's first 50 episodes.

After 200 episodes, almost all algorithms have reached acceptable stability with the exception of **Q-learning**. Note: Due to the very large size of the negative reawards taken by **Off-Policy**, I could not draw its shape next to other policies.

```
On-Policy Monte Carlo : median(last 100) = 0.0, variance(last 100) = 0.74

Off-Policy Monte Carlo : median(last 100) = -98.5, variance(last 100) = 12364.23

SARSA : median(last 100) = 0.0, variance(last 100) = 1.33

Double Q-learning : median(last 100) = 0.0, variance(last 100) = 0.0

2-Step Expected SARSA : median(last 100) = 0.0, variance(last 100) = 1.02

3-Step TreeBackup : median(last 100) = 0.0, variance(last 100) = 1.44
```

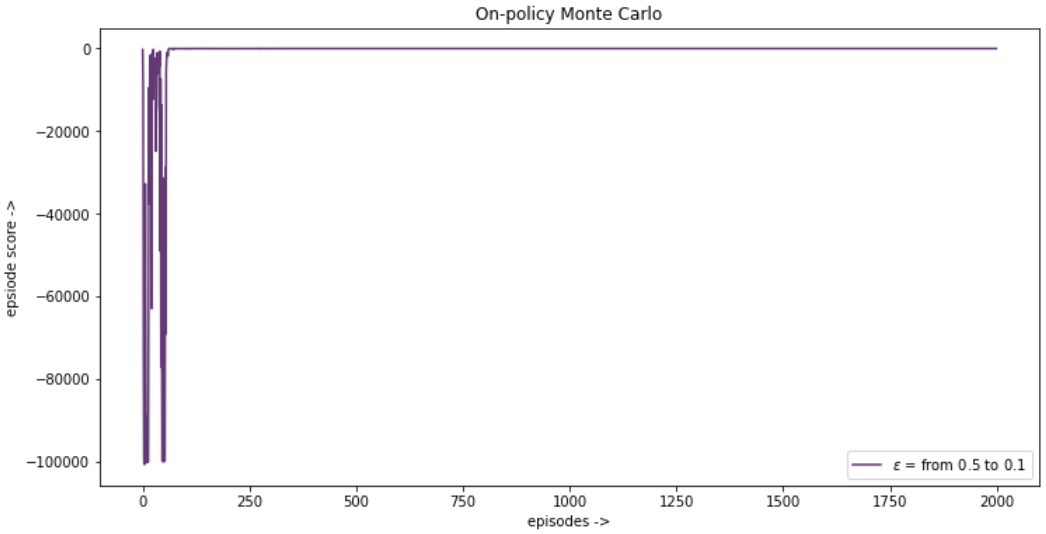
Judging by the numbers above, although the variance of **Double Q-learning** method is very low, if we evaluate the algorithms in terms of regret, **2-step expected SARSA** is the best method.

Bonus

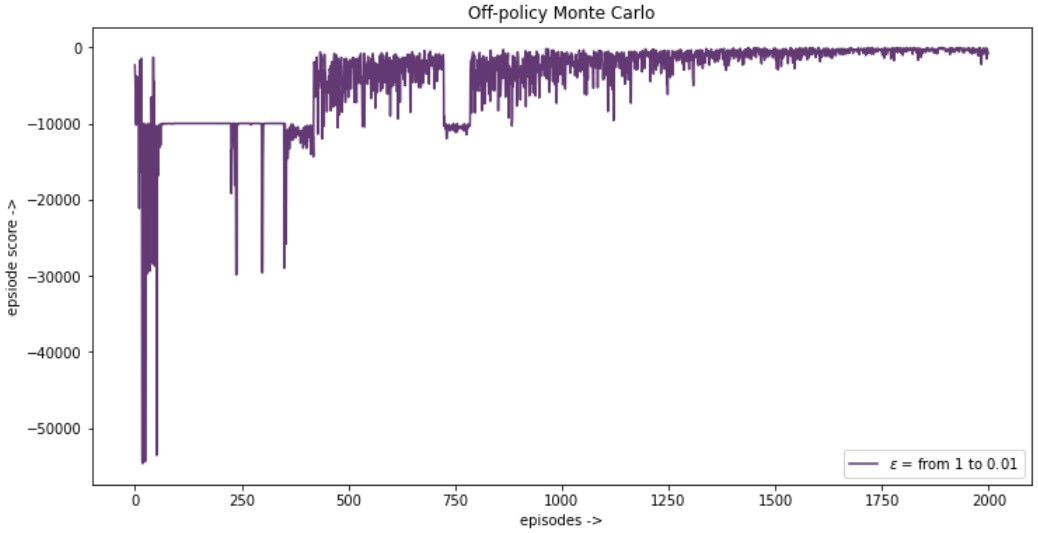
The correct way to converge sooner is to add -1 punishment to each time step before reaching the island.

Monte Carlo :

1. ***On-policy MC - First Visit*** :

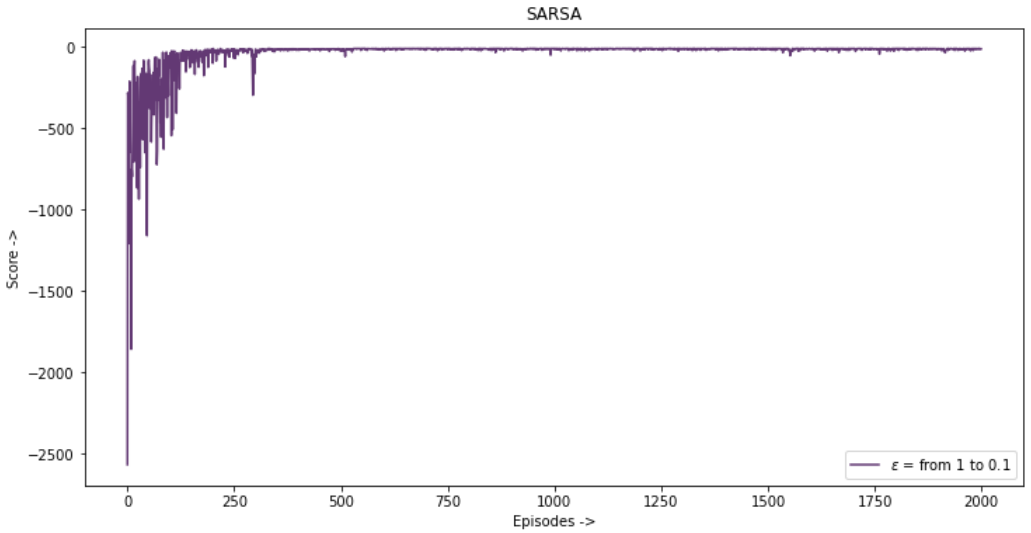


2. **Off-policy MC - Weighted Importance Sampling**:

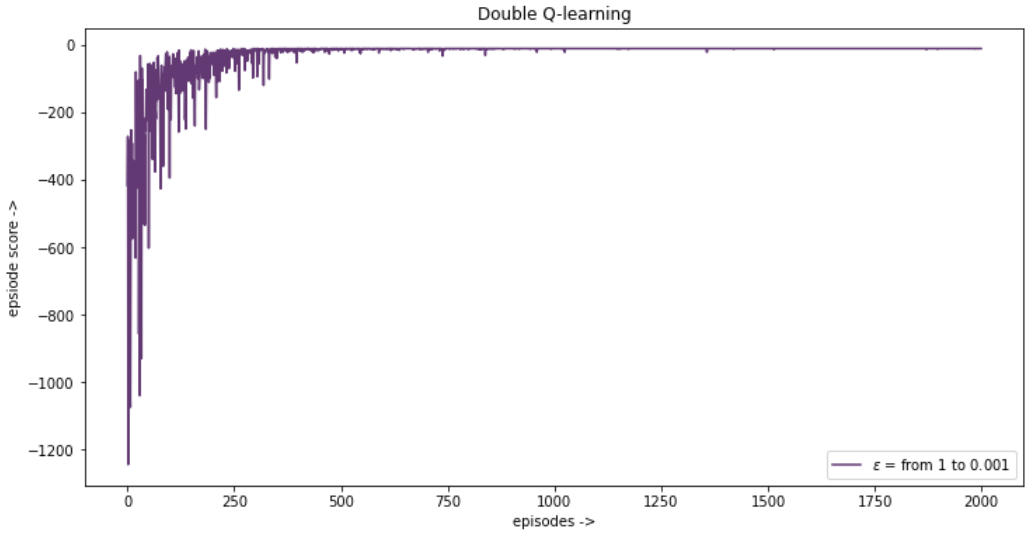


Temporal Difference :

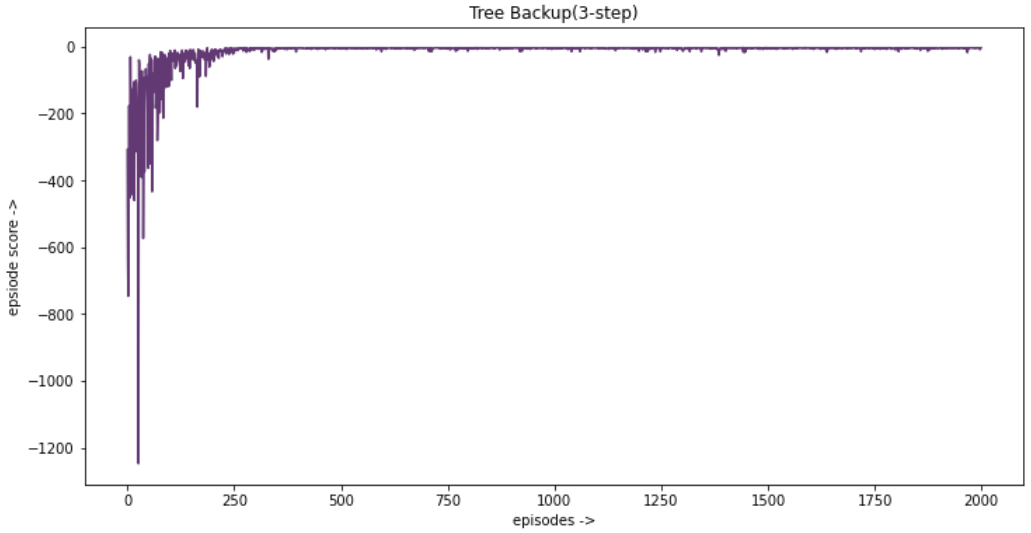
1. **SARSA** :



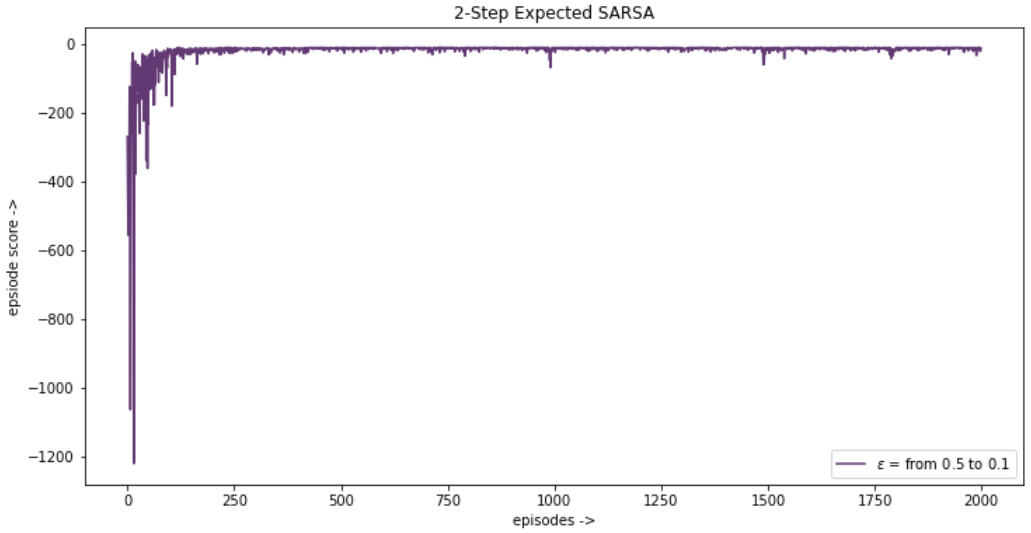
2. **Double Q-learning** :



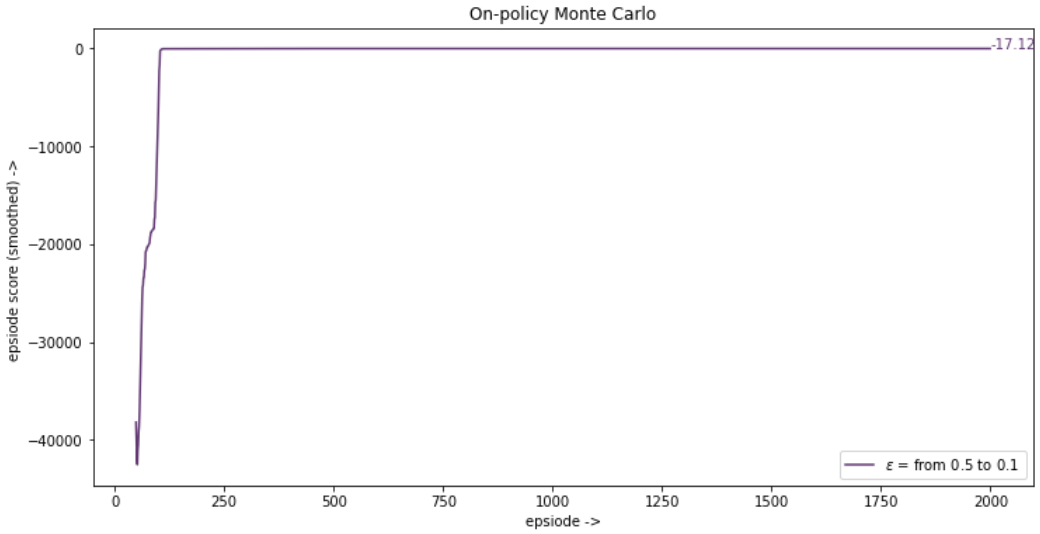
3. **Tree backup** :

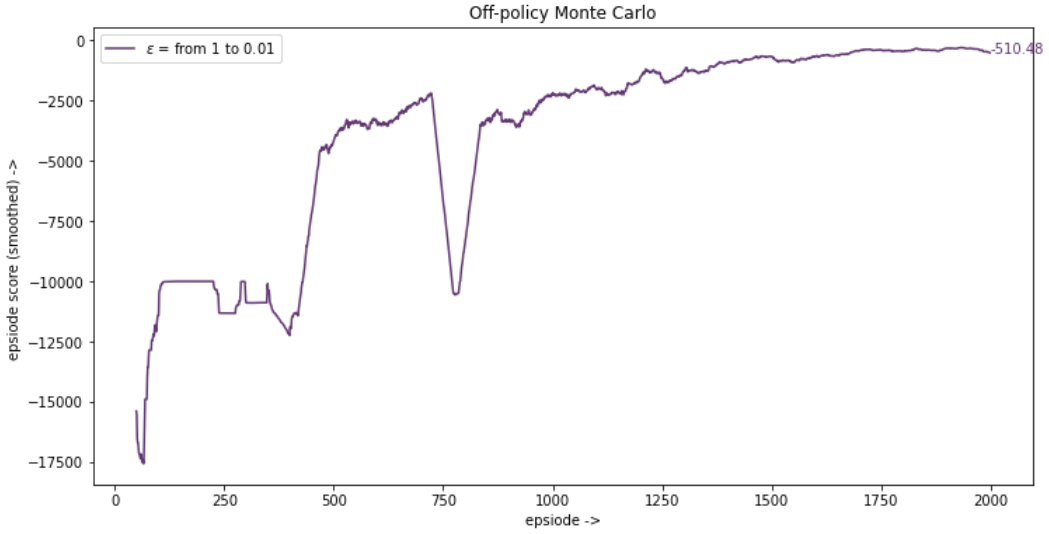


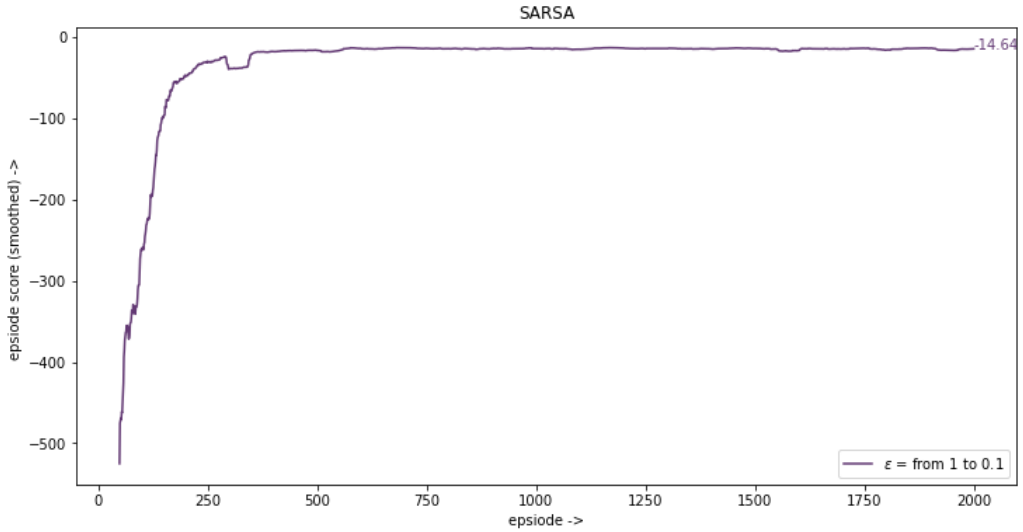
4. **2-Step Expected SARSA**:

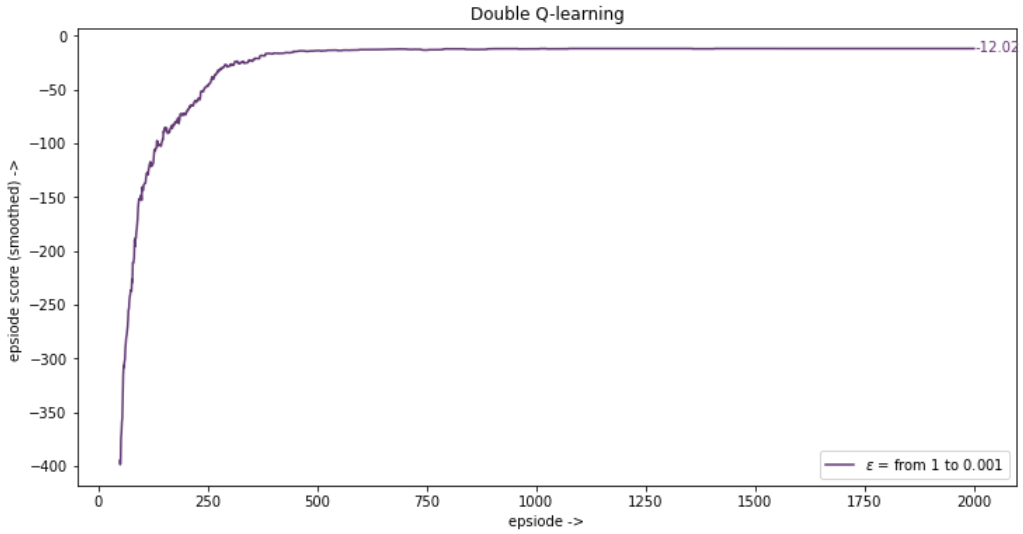


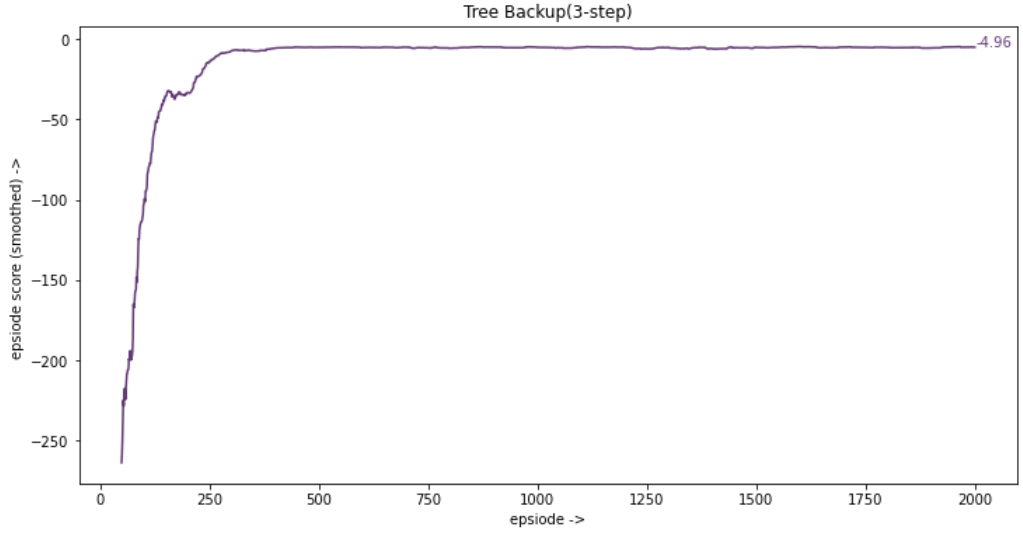
To have better visualization, I also plotted *reward/episode* using sliding window with size 50

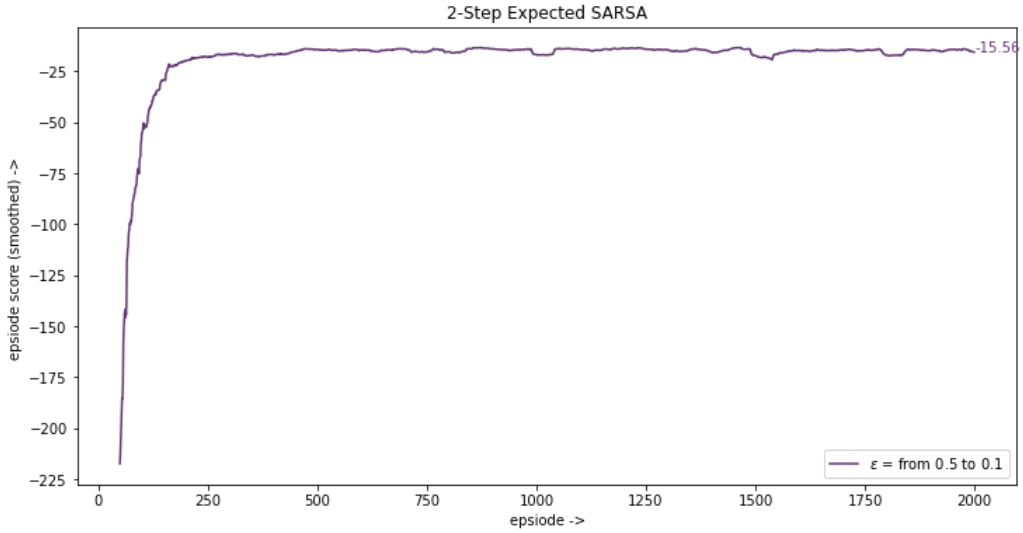

On-policy Monte Carlo
-17.12
epsiode score (smoothed) ->
epsiode ->
ε = from 0.5 to 0.1

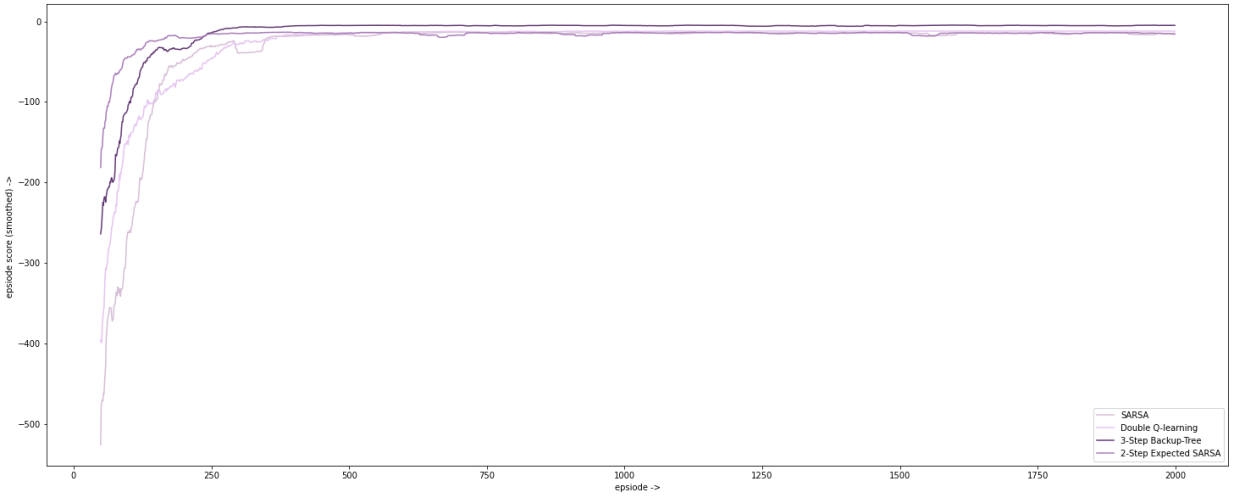

Off-policy Monte Carlo
-510.48
epsiode score (smoothed) ->
epsiode ->
ε = from 1 to 0.01

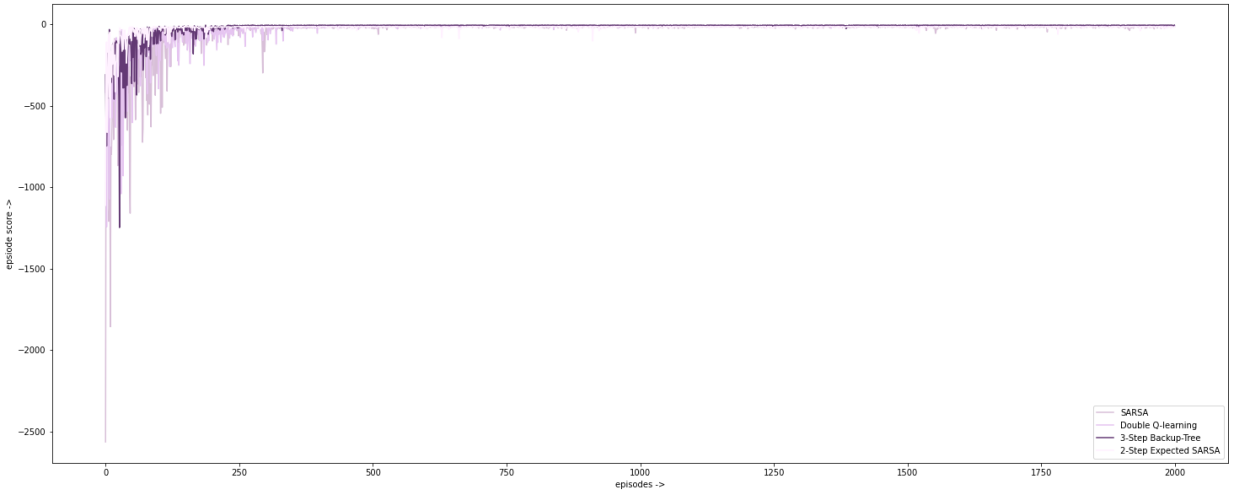

SARSA
-14.64
epsiode score (smoothed) ->
epsiode ->
ε = from 1 to 0.1


Double Q-learning
-12.02
epsiode score (smoothed) ->
epsiode ->
ε = from 1 to 0.001


Tree Backup(3-step)
-4.96
epsiode score (smoothed) ->
epsiode ->


2-Step Expected SARSA
-15.56
epsiode score (smoothed) ->
epsiode ->
ε = from 0.5 to 0.1





2-step expected SARSA have the highest rate of convergence and the least amount of regret because it receives less punishment from the environment in it's first 50 episodes.

Note: Due to the great differences between the Monte Carlo punishment scale and temporal difference methods, MCs are not plotted.

After 250 episodes, **3-Step Tree backup** and **2-Step Expected SARSA** have reached acceptable stability.
Other algorithms reach relative stability after about 500 episodes.

```
On-Policy Monte Carlo : max reward = -14.0, median(last 100) = -18.0, variance(last 100) = 12.34

Off-Policy Monte Carlo : max reward = -39.0, median(last 100) = -357.0, variance(last 100) = 104106.99

SARSA : max reward = -12.0, median(last 100) = -14.0, variance(last 100) = 20.31

Double Q-learning : max reward = -12.0, median(last 100) = -12.0, variance(last 100) = 0.01

2-Step Expected SARSA : max reward = -3.0, median(last 100) = -4.0, variance(last 100) = 2.87

3-Step TreeBackup : max reward = -12.0, median(last 100) = -14.0, variance(last 100) = 15.31
```

Judging by the numbers above, although the variance of **Double Q-learning** method is very low, if we evaluate the algorithms in terms of regret, **2-step expected SARSA** is the best method.