# Chapter 06 - Q-Learning, SARSA

*Author: Gianmarco Scarano*

*gianmarcoscarano@gmail.com*

# 1. Monte-Carlo Method

Recalling the roll-out data generation, we know that at the end we have our Dataset composed by:

$$D = \{s_i, a_i, y_i\}_{i=1}^{N}$$

But now, given $(s, a)$ how do we estimate $Q^{\pi}(s, a)$?

As we said, we can estimate the $Q$ by sampling, executing until termination condition and then average many roll-outs to compute our estimate. This is the Monte-Carlo method.

Of course, the True MC cannot be applied if we are in an infinite horizon MDP, but we can adapt using the following trick:

We have a target for the Monte-Carlo update which is $G$ (our label $y$, the return).

The return will be equal to the sum of rewards until termination condition: $G = \sum_{h=0}^{Term.} \gamma^h r_h$.

While, if we use a function approximator (NN, etc.) we simply run regression (which will converge to the mean of the returns):

$$argmin_Q \sum_{i=1}^{N} (Q(s_i, a_i) - y_i)^2$$

The answer we should ask now is: If the state space is not huge, why should we do approximation if we can rely on a table? Meaning that if the state space is not that large, we can list all the states in a table.

## 1.1 Moving Average

We compute this through the moving average definition, where we want to compute our estimate as a mean and update it with new data coming from the agent's experience.

$$MEAN\ \mu_k := \frac{1}{k} \sum_{0}^{K-1} x_j$$
$$\Rightarrow \frac{1}{k} \left( x_k + \sum_{0}^{k-2} x_j \right)$$
$$\Rightarrow \frac{1}{k} (x_k + (k-1)\mu_{k-1})$$
$$\Rightarrow \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$
$$\Rightarrow \mu_{k-1} + \alpha(x_k - \mu_{k-1})$$

Here, we basically update our previous average $(\mu_{k-1})$ with $\alpha$ which could look like some sort of learning rate.

## 1.2 Tabular Updates

So now, we got a general form to do our updates in Tabular form as follows:

$$p_{k-1} + \alpha(y_k - p_{k-1})$$

Where we update our previous estimate + a LR which multiplies (newValue – previous estimate)

- $\boldsymbol{p_{k-1}}$ = Our current estimate (actually the one we had at step -1)
- $\boldsymbol{\alpha}$ = LR between 0 and 1
- $\boldsymbol{y_k}$ = New value. Our target / label.

### 1.3 Monte-Carlo Updates

Either two ways:

- **Tabular** = $Q(s,a) \longleftarrow Q(s,a) + \alpha(G_i - Q(s,a))$
  - We do this at every termination step
- **Function Approximator** = $argmin_Q \sum_{i=1}^{N}(Q(s_i,a_i) - G_i)^2$
  - We store data and update in batch after a while or do online learning (it means we do this at every datapoint, which is quite expensive and less stable).

Note that $G_i = y_i$ (they are the same thing)

**PROS:**

- Unbiased
- Good convergence properties also with function approximators
- Not very sensitive to initialization

**CONS:**

- Needs some sort of termination condition (not very well defined for Infinite MDPs)
- Depends on many random actions, transitions, rewards (*High variance*!)
- We need the whole returns (rewards, in this case)

## 2. Monte-Carlo vs Temporal Difference

***Monte-Carlo :=***

$$Q^{\pi}(s_t,a_t) = \mathbb{E}[\sum_{h=0}^{\infty}\gamma^h r_h \mid (s_0,a_0) = (s_t,a_t), a_{h+1} = \pi(s_h), s_{h+1} \sim p(\cdot \mid s_h,a_h)]$$

***Temporal Difference :=***

$$Q^{\pi}(s_t,a) = r_t + \gamma \mathbb{E}_{s' \sim p(\cdot \mid s,a)}[V^{\pi}(s')]$$

This is a very important step in order to properly detect the difference between Monte-Carlo approach and the Temporal Difference one. In the Monte-Carlo approach, we do use the actual return, while in the Temporal Difference, we perform **Bootstrapping**.

This approach aims at using the **estimate** of the next state value instead of the TRUE(*MC*) next state value. We iteratively update our estimate towards the current reward and the current estimated return, meaning that we don't have to estimate the whole trajectory since value for the next state is based on an estimate starting from my actual state. We conclude saying that in Temporal Difference Update there is no transition function, since we do perform bootstrapping (sampling).

Continuing the Temporal Difference Update, by recalling the Tabular formula (and also valid for function approximator) we have seen in 1.3, our $G_i$ (which is our new value; $y$) now becomes:

$$Q(s,a) \longleftarrow Q(s,a) + \alpha(r_i + \gamma Q(s',\cdot) - Q(s,a))$$

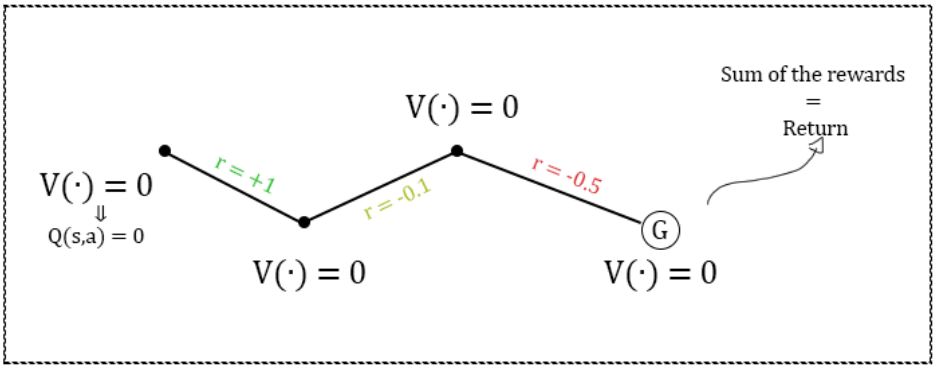
Very simply, the red part is our one-step-update while $Q(s,a)$ is the current estimate.

The red part plus $-Q(s,a)$ is called **TD error** ($\delta$).

We'll now see the meaning of the ($\cdot$) action next to $s'$ thanks to an example.

## 2.1 Example of Temporal Difference vs Monte-Carlo update

$$S = \begin{matrix} A \\ \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ 0 \end{bmatrix} \end{matrix}$$




Starting commenting this image, we have a vector of States and Action (State-Action space) as well as our environment composed by 4 states, rewards etc. $\gamma$ and $\alpha$ are both $0.9$.

In a **Monte-Carlo** update, we said that we do not use the previous estimate. We simply just do the usual Q-function (with sum of discounted rewards etc.).

$$Q(s_0, a_0) = 0 + 0.9(\alpha) \cdot (0.4\textbf{(*)} - 0) = 0.36$$

$$\text{Where } \textbf{(*)} = (1 - 0.1 - 0.5) = 0.4$$

In the **Temporal Difference** method, the scheme is the following:

- Iteration 0:
  - $Q(s_0, a_0) \leftarrow Q(s_0, a_0) + \alpha(r_i + \gamma Q(s_1, a_x) - Q(s_0, a_0)$ (*General formula to follow*)
  - $Q(s_0, a_0) = 0 + 0.9(\alpha) \cdot \left(1 + 0.9 \cdot 0 Q(s_{1,} a_x) - 0\right) = 0.9$
  - $Q(s_1, a_1) = 0 + 0.9(\alpha) \cdot \left(-0.1 + 0.9 \cdot 0 Q(s_{2,} a_x) - 0\right) = -0.09$
- Iteration 1:
  - $Q(s_0, a_0) = 0.9 + 0.9(\alpha) \cdot (1 + 0.9 \cdot (\mathbf{-0.09}) - 0.9) = \mathbf{0.9171}$
  - $Q(s_1, a_1) = -0.09 + 0.9(\alpha) \cdot \left(-0.1 + 0.9 \cdot 0 Q(s_{2,} a_x) - (-0.09)\right) = \mathbf{-0.09}$
  - Etc...

As we can see, in the second iteration, we use the values we computed for the first iteration.

## 2.2 Pro / Cons of Temporal Difference Update

Let's examine the pros and cons of the Temporal Difference Update.

**CONS:**

- Sensitive to initial value
- Biased estimate of $Q^{\pi}$ (since we are using an approximation of $Q$ instead of the true value)

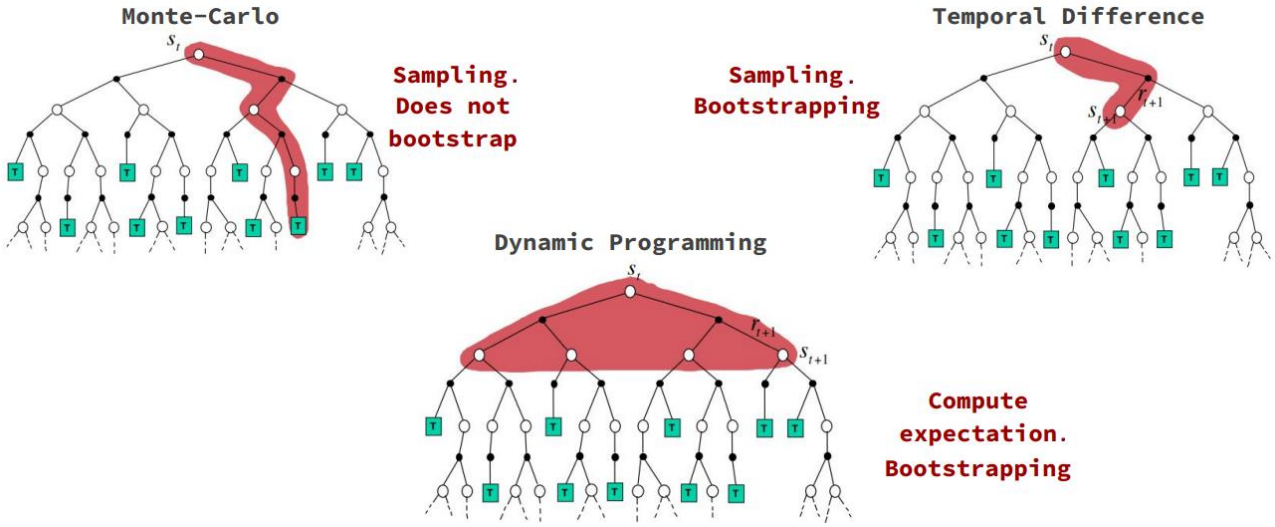
**PROS:**

- Can learn at every step very easily (more efficient than Monte-Carlo approach)
- Depends on just one action instead of many random actions (like in Monte-Carlo)
- Converges but not always

The problem now is that for example, in $(s_2, a_x)$ we have chosen an arbitrary action, while the question is how do we choose that appropriate action?

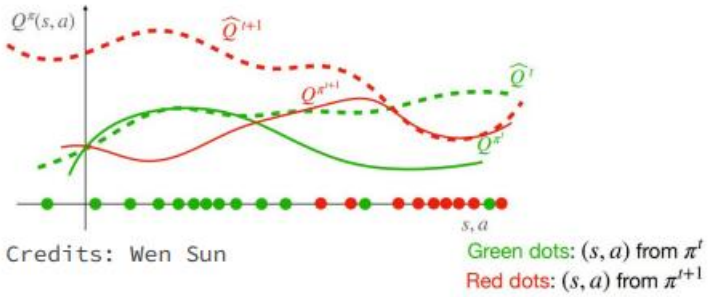
$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_i + \gamma Q(s',\cdot) - Q(s,a))$$

In general, we refer to the action as that little dot $(\cdot)$ next to the next state $s'$. We'll see how to get into this later. For now, let's just put a remark here **(*)**.




## 2.3 $\epsilon$-Greedy exploration

So, if we remember from last chapter, we had the problem of exploration (right image here) where we had an oscillation from the distribution change. For having monotonic improvement in that condition, we needed a strong coverage assumption.




The idea now is that instead of being greedy with respect to $Q$, we try all the actions with some probability and here comes the <u>$\epsilon$-greedy exploration</u>, which states:

- Probability $1-\epsilon$, choose the greedy action (meaning do argmax of Q)
- Probability $\epsilon$, we choose a random action

This strategy handles the exploration-exploitation trade-off. Let's define $m$ as the number of actions, then:

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

The monotonic improvement is given by the fact that for any $\epsilon$-greedy policy $\pi$, the $\epsilon$-greedy policy $\pi'$ obtained by $Q^\pi$ is an improvement, such that $V^{\pi'} \geq V^\pi$ holds.

If we set $\epsilon = \frac{1}{k}$ with $k$ going to infinity ($\infty$), then we might visit all state-action pairs infinitely many times and converge to a greedy policy, which in other words could let us converge to the optimal $Q^*$ in terms of Monte-Carlo update.

## 3. S.A.R.S.A & Q-Learning

Now, going back to the problem of selection of the action **(*)**, a few pages back, we could use two strategies:

- S.A.R.S.A:
  - The target action is selected according to the policy $\pi$ (which can be $\epsilon$-Greedy with respect to Q). This means that we select the target action according to the same policy we execute. This is called **ON-POLICY**.
- Q-Learning:
  - The target action is selected according to a different policy we are executing, so it's greedy with respect to Q. Here we are selecting the action which gives us the maximum value in the next state according to the argmax w.r.t to Q, which is our current best estimate!

This can be easily represented by the pseudo code and a graphic example:

### Sarsa

---

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize $S$
  Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Repeat (for each step of episode):
    Take action $A$, observe $R$, $S'$
    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until $S$ is terminal

### Q-Learning

---

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize $S$
  Repeat (for each step of episode):
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $A$, observe $R$, $S'$
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$;
  until $S$ is terminal

Both pseudocodes are really similar. The only thing that changes is the way we compute the action. In Q-Learning we use the best estimate (argmax), in S.A.R.S.A. we use the action according to the policy.

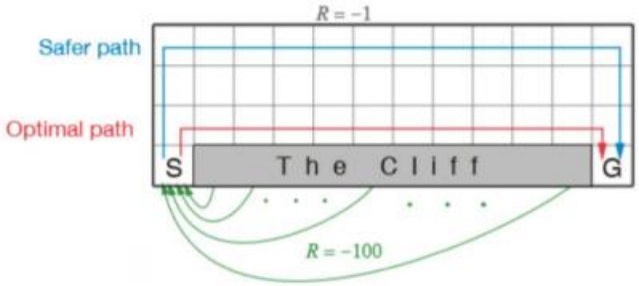
This is a graphic example which perfectly explains the difference between S.A.R.S.A. and Q-Learning:

# Sarsa & Q-Learning: Example

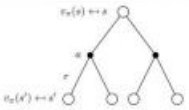
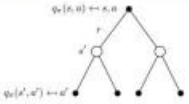
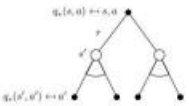
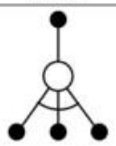
---

**Sarsa:** takes action selection into account and learns the safer path

**Q-Learning:** learns the optimal path independently of the action selection that, at learning time (i.e., while being eps-greedy), makes it fall in the cliff




In Q-Learning we might risk that in the middle of the cliff (since it's eps-greedy), we might take an action which lets us fall into the cliff!

| | *Full Backup (DP)* | *Sample Backup (TD)* |
|---|---|---|
| Bellman Expectation Equation for $v_\pi(s)$ |  Iterative Policy Evaluation |  TD Learning |
| Bellman Expectation Equation for $q_\pi(s,a)$ |  Q-Policy Iteration |  Sarsa |
| Bellman Optimality Equation for $q_*(s,a)$ |  Q-Value Iteration |  Q-Learning |

| *Full Backup (DP)* | *Sample Backup (TD)* |
|---|---|
| Iterative Policy Evaluation | TD Learning |
| $V(s) \leftarrow \mathbb{E}\left[R + \gamma V(S') \mid s\right]$ | $V(S) \overset{\alpha}{\leftarrow} R + \gamma V(S')$ |
| Q-Policy Iteration | Sarsa |
| $Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma Q(S', A') \mid s, a\right]$ | $Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma Q(S', A')$ |
| Q-Value Iteration | Q-Learning |
| $Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') \mid s, a\right]$ | $Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')$ |