**Reinforcement Learning with SARSA — A Good Alternative to Q-Learning Algorithm**

# Intro

The beauty of Machine Learning is that there is no shortage of approaches for tackling complex tasks. For example, **Reinforcement Learning (RL)**practitioners have developed multiple algorithms capable of teaching **intelligent agents** to navigate their environments and perform actions.

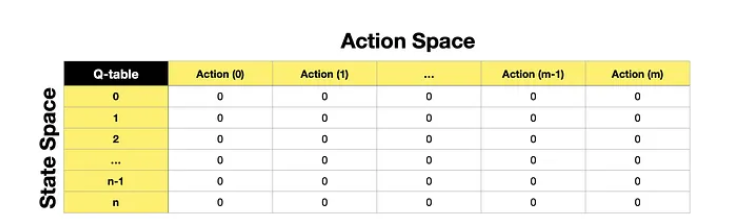
# ****How does SARSA work?****

## Q-table

SARSA is a **value-based method** similar to Q-learning. Hence, it uses a **Q-table**to store values for each **state-action pair**. With value-based strategies, we train the agent **indirectly** by teaching it to identify which states (or state-action pairs) are more valuable.

Typically we start with all values in a Q-table initialised to 0, and we use training to optimise the Q-table. Then our agent can use information stored in a Q-table to choose the best action at each state (i.e., the action with the highest value for each state is the one selected by the agent).

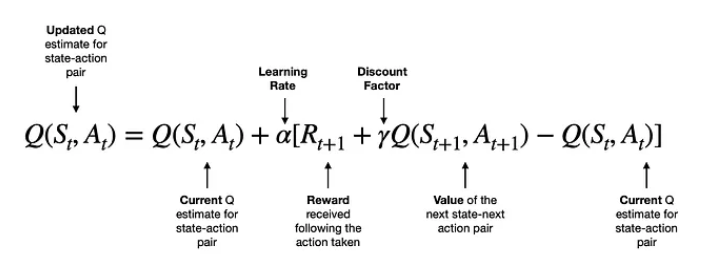
Here is an example of an empty Q-table:



## SARSA algorithm

SARSA is an **on-policy** algorithm, which is one of the areas differentiating it from Q-Learning (off-policy algorithm). **On-policy**means that during training, we use the same policy for the agent to **act** (acting policy) and to **update** the value function (updating policy). Meanwhile, with the **off-policy** approach, we use different policies for acting and updating.

Now let’s take a look at the SARSA algorithm itself:

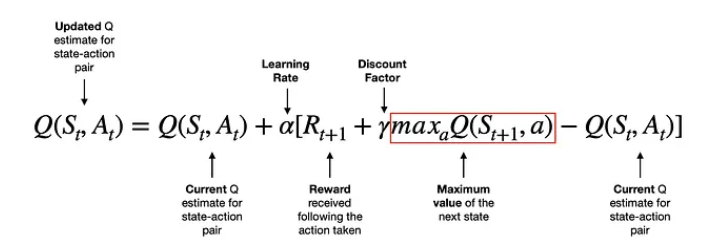


* Q is the **value function**, and the term on the left Q(𝑆𝑡,𝐴𝑡) is the **new value** for the specific state-action pair. Note, S refers to State, and A refers to Action.
* On the right-hand side of the equation, we find the same term Q(𝑆𝑡,𝐴𝑡), which, in this case, is the **current value** for that same state-action pair.
* To update the current value, we take the **reward (**𝑅𝑡+1) following the action taken by the agent, add the **value for the next state-next action pair**𝛾𝑄(𝑆𝑡+1,𝐴𝑡+1) **discounted by gamma**, and subtract the **current value** Q(𝑆𝑡,𝐴𝑡).
* So, the terms in the square brackets produce a positive, zero or negative value, which leads to either increase, no change or a decrease in the new value of Q(𝑆𝑡,𝐴𝑡). Note that we also apply a **learning rate (alpha)** to control the “size” of each update.

Since SARSA uses **Temporal Difference (TD) approach**, the algorithm will keep updating the Q-table after each step until we reach the maximum number of iterations or the solution converges to an optimal one.

## **Comparison to Q-Learning**

If we look at the equation used by the Q-Learning algorithm, we can see that the difference lies in how it selects the value of the next state. I.e., **Q-Learning takes the maximum value** for the next state based on the existing values in the Q-table. Meanwhile, **SARSA takes the value of the next state-next action pair,** as seen above.



***Value-based RL like***[***Q-Learning***](https://towardsdev.com/reinforcement-learning-q-learning-38146880ca49)***,***[***SARSA***](https://arshren.medium.com/reinforcement-learning-sarsa-and-q-learning-e11ebe87dca9)***, or***[***DQN***](https://arshren.medium.com/deep-q-learning-a-deep-reinforcement-learning-algorithm-f1366cf1b53d)***are best suited when the action space is discrete and not too large.***