Reinforcement Learning (RL)
(The basics, Q-Learning, Deep Q-Learning)

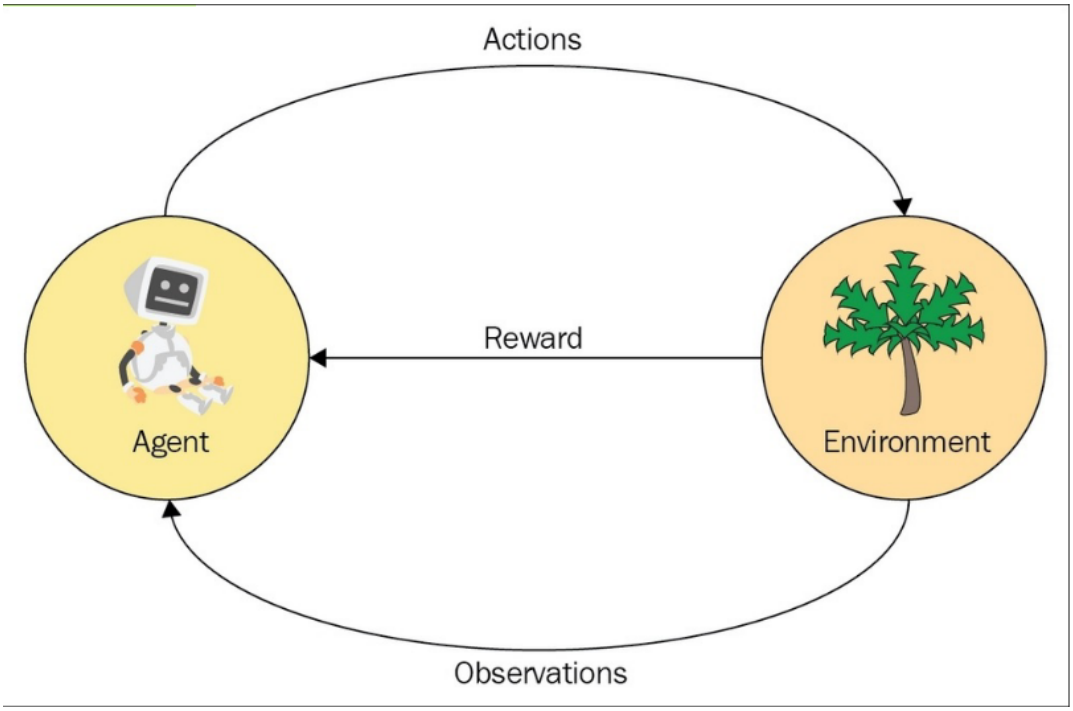
1 - Reinforcement Learning:

In reinforcement learning, unlike supervised and unsupervised learning, you do not need a dataset to learn, you simply put an **agent** in an **environment** that explores and learns what to do, i.e., find out what is right and what is wrong.

The agent begins in the environment with an initial **state** and a set of **actions** known as **action space**. When the agent performs an action in the environment, it will move from one state to the next, and it will receive a **reward** value from the environment in order to learn if the action performed was the correct one. The reward value will aid our agent in learning by preventing him from taking paths that are less rewarding.

In summary, the agent will figure out which state changes will yield the most rewards.

The architecture of reinforcement learning is illustrated in the diagram below:



1- 1 The entities of reinforcement learning:

Therefore, in reinforcement learning, we have the following entities:

- Agent.
- Reward.
- Actions.
- Environment.
- State and also known as observations.

Each entity's definition is summarized below:

Agent: This is the actor in reinforcement learning, he is the one who performs the actions and learns from them to perform well in the environment and receive the maximum rewards.

Actions: Actions are the methods performed by the agent in an environment and allow it to move from one state to another. Each action performed by the agent **results in a reward** from the environment. The decision of which action to choose is made by the **policy**.

Reward: a numerical value received from the environment by the agent in relation to the action performed in a state, the agent's goal is to try to get the maximum of these rewards.

State: Each scenario that the agent encounters in the environment is formally called a state. The agent moves from one state to another **by performing actions**.

Environment: This is the world that contains the agent and that allows the agent to observe the state of this world. When the agent applies an action to the environment, the environment goes from one state to another, and sends the reward value to the agent.

Further definitions in context:

Policy (π): Policy, denoted π (or sometimes π (a|s)), is a mapping between some state S and the selection probabilities of each possible action given that state.
Terminal State: Terminal states mark the end of an episode. There are no possible states after a terminal state has been reached.

Episode: It is to go from the initial state to the final state, the learning is done in several episodes.

Discount Factor: The Discount Factor, usually denoted by γ, is a factor multiplying the expected future reward, and varies in the interval [0, 1]. It controls the importance of future rewards relative to immediate rewards.

Remark: The factor has the purpose of provoking the agent to finish faster and gives importance to immediate rewards rather than future rewards.

Example: In a game, the reduction factor has the desirable effect of pushing players to try to win as soon as possible.

1 – 2 Exploration and Exploitation:

Exploration: The agent must try many different actions in many different states in order to try to learn all the available possibilities and find the path that will maximize its overall reward..

Exploitation: Use the information learned in the exploration stage to choose the right action to get the most rewards.

1 – 3 Value-function and Q-function:

To give some intelligence to the agent, there is a function that represents the quality of the state in which the agent can be. The definition of this function is below.

Value function: the Value function is a measure of the expected global reward assuming that the agent is in state *S.*

To know to what measure a given action is good when we are in a precise state, we use the "Q-function". The definition of this function is below:

Q-value (Q-function): The function that predicts the expected returns from performing an action in a given state. It is also known as **« state-action value function »**.

Important: In reinforcement learning methods, these two functions will be used. Each function has its own set of drawbacks and benefits, for example, the Q-function is useful in Q-Learning and Deep Q-Learning, while the value function is useful in Actor-Critical (A2C) analysis (when the action space is large).

1 – 4 Reinforcement Learning algorithms:

1.4.1 - Q-Learning : It's a reinforcement learning algorithm that starts with an empty table and then stores all the "**Q-values"** of all the possible state-action peers (i.e. what Q-value do we get for each state if we execute an action?). It updates this table using **Bellman's equation**, and the actions are usually chosen using **ε-Greedy policy**. This algorithm is very useful when the state space is small.

The following equation is used to update the "Q-value":

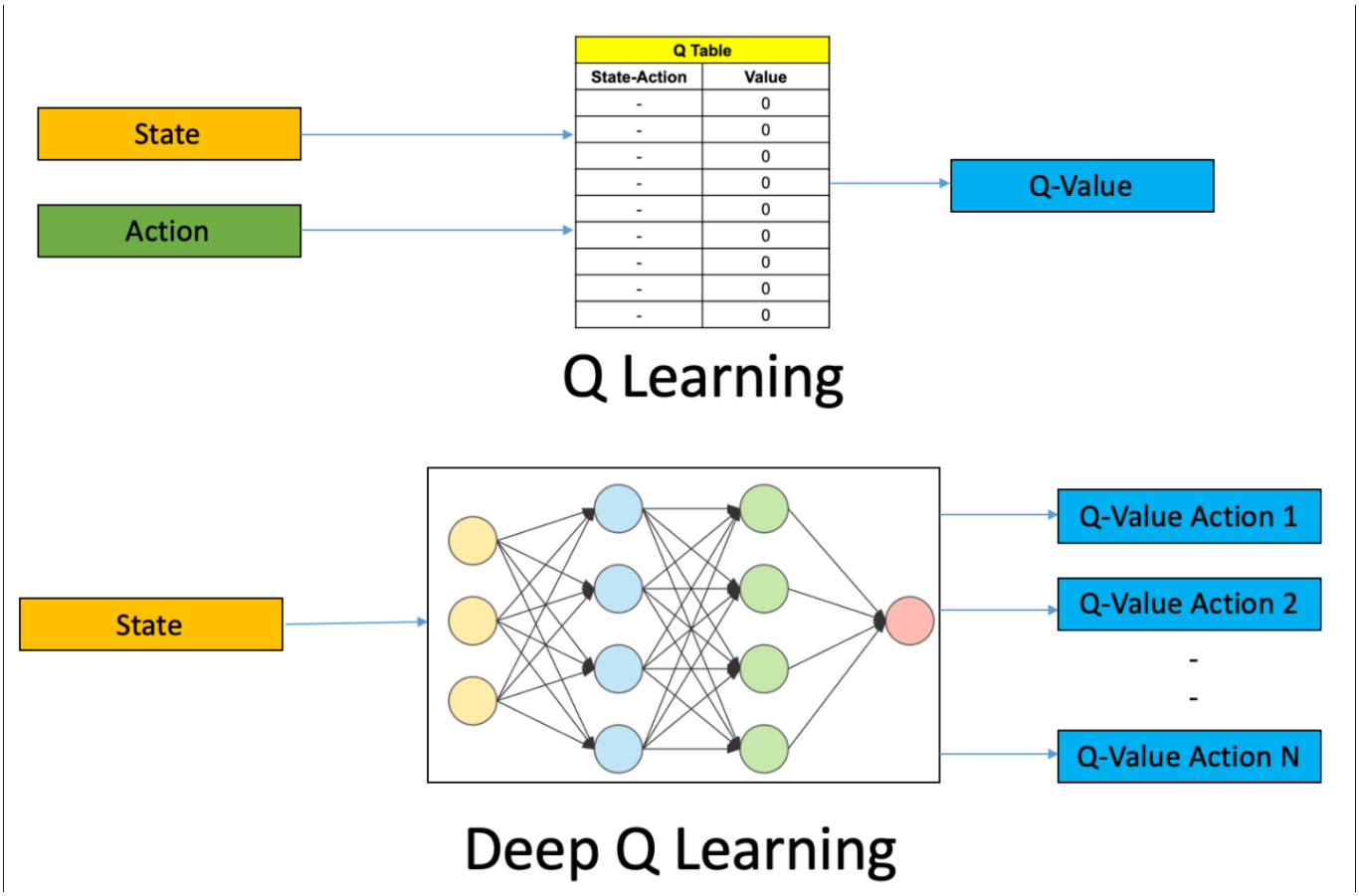
$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

However, this algorithm contains a significant drawback. The size of the “Q-values” array might become enormous when the state space is huge, and the memory cannot handle it. To overcome this limitation and employ a large state space, we use the **Deep Q-Learning** algorithm.

1.4.2 - Deep Q-Learning: The drawbacks of the Q-Learning algorithm in terms of the size of the state space motivated researchers to seek new solutions. The results of this research led to the “Deep Q-Learning” algorithm. The word “Deep” comes from “Deep Learning” based on **neural networks**.

The principle of neural networks is combined with the principle of “Q-Learning” in this algorithm. The states of our environment will be the input to the neural network, and we will get a “Q-value” for each action as an output. The set of ”Q-values” will then be utilized to select an action during the **exploitation** phase.

The figure below illustrates the difference between Q-Learning and Deep Q-Learning while showing the inputs and outputs of our neural network.



The neural network needs data to learn and predict outcomes, and since reinforcement learning does not have a dataset, we will obtain our own training data from the environment, the environmental information we will use as training data is: **state**, **next_state**, **action**, **reward** and a boolean value **(done)** that indicates if we have reached our final state.

Therefore, a set of information is obtained at each transition and this set is called an **experience**. The experience is then stored in a buffer called **"Replay Buffer"**. The learning will start when the "Replay Buffer" reaches a certain size (to have enough data for the learning).

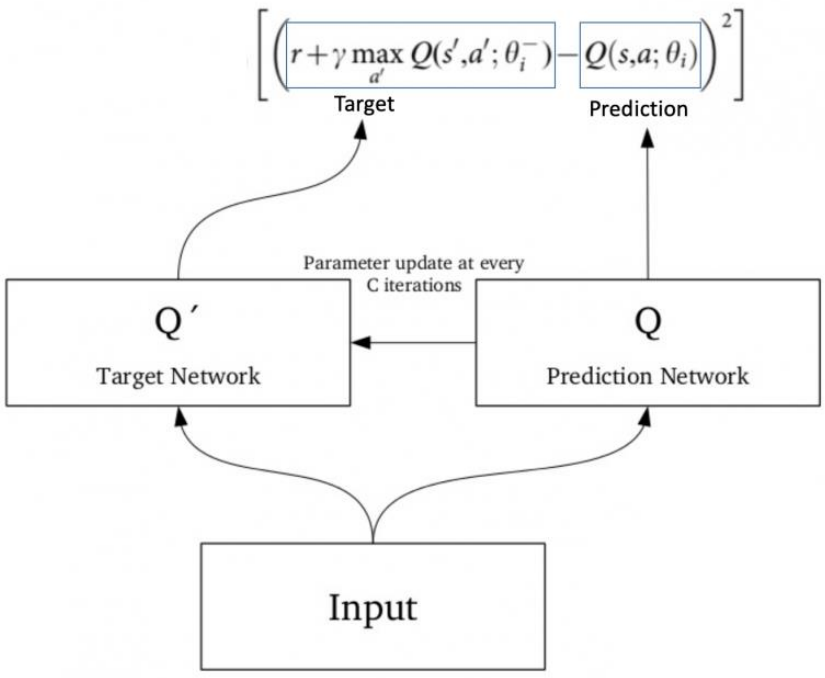
For the correct working of the neural network, the data must be independent, for that, we will use a technique to reduce the temporal correlation in the data by taking random samples of data to be used for training. This technique is known as **« Replay Experience »**.

The learning of the neural network here is a supervised learning, so we need a target (like the *"label"*) and a *"y"* value generated from the network, these values are obtained from the Bellman equation, and then calculate the difference between the *"target"* and the *"y"* value to get the error.

Important: An issue that arises in Deep Q-Learning is that the Bellman equation gives us the value of *Q (s, a)* via *Q (s', a')*. However, there is only one-step between the two states *s* and *s'*. This makes them so similar that it is difficult for a neural network to distinguish them. When we perform parameter updates on the network, in order to bring *Q (s, a)* closer to the expected result.

There is a trick, called **"target network"**, the principle is to keep a copy of our network and that we use it for the *Q-value (s', a')* in the Bellman equation.

The figure below shows us the *"target"* and the prediction value *"y"* in the Bellman equation, as well as the utility of the "target network".



Deep Q-learning tends to overestimate the values of Q, which can affect training performance. The cause of this drawback is the "max" operation of the Bellman equation. Therefore, the researchers proposed a slight change in the equation, which will be as follows:

$$Q(s_t, a_t) = r_t + \gamma \max_a Q'(s_{t+1}, \arg\max_a Q(s_{t+1}, a))$$

Moreover, this new approach is called **« Double Deep Q-Learning »**.

There are other drawbacks in Deep Q-Learning such as the large size of the action space or the continuous actions, which can slow down the convergence of the model. The **"Policy Gradient"** and **"Actor-Critic (A2C)"** algorithms can solve this drawback.

Extra Resources :

https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e

https://pylessons.com/CartPole-reinforcement-learning

https://developers.google.com/machine-learning/glossary/rl

https://cdn.hackernoon.com/hn-images/1*JJ3Dx4O3blc_haCUjv5Y5A.jpeg