

Q-Learning

Approaches to Implement RL

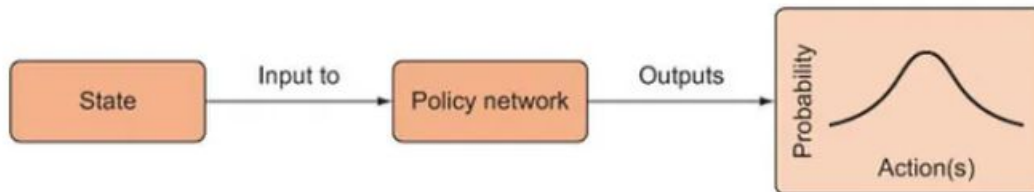
- **Policy-based:**

The main goal of Reinforcement learning is to find the **optimal policy** π^* that will maximize the expected cumulative reward.

- **on-policy/policy gradient**

1. In this approach, the agent tries to apply such a policy that the action performed in each step helps to maximize the future reward without having to learn a value function.
2. Policy function/policy network accepts the state and returns the best action.
3. More precisely, returns the probability distribution over the actions, which can be used to pick the best action.

The idea is to **parameterize the policy**.



$$\pi_{\theta}(s) = \mathbb{P}[A|s; \theta]$$

The policy given a state outputs a probability distribution over actions at that state.

Value-based:

- The value-based approach is about to find **the optimal value function**, that estimates the expected cumulative reward for each state or state-action pair. which is maximum at a state under any *policy*.
- Value-based RL algorithms attempt to learn an optimal value function
Q-learning and Deep Q-Networks (DQNs) are examples of value-based RL algorithms.
- The agent optimizes the expected return (V-value function).

$$V^{\pi}(s) = E \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, \pi \right]$$

- The **optimal expected return** is defined as:

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

Model based vs. Model free approaches

Model-Based: learn the model of the world, then plan using the model. Update and re-plan the model often.

– *Model free approach RL:*

derive the optimal policy without learning the model.

Model, Value, Policy

- The **model-based** algorithms use planning to estimate the optimal policy and create the model.
- In contrast, **model-free** algorithms learn the consequences of their actions through trial and error.
- The **value-based** method trains the value function to learn which state is more valuable and take action.
- **Policy-based** methods train the policy directly to learn which action to take in a given state.
- **Q-learning** is an **Off policy RL algorithm**, which is used for the temporal difference Learning.

Q (quality) value

$$Q^{\pi}(s, a) = E \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi \right]$$

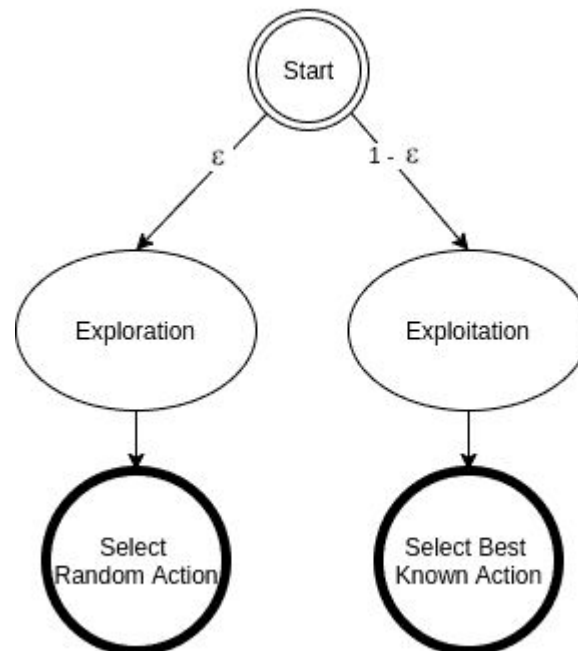
Similarly to the V-function, the optimal Q value is given by:

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$$

The optimal policy can be obtained directly from this optimal value:

$$\pi^*(s) = \arg \max_a Q^*(s, a)$$

- **In Q-learning, we select an action based on its reward.** The agent always chooses the optimal action. Hence, it generates the maximum reward possible for the given state.
- In epsilon-greedy action selection, the agent uses both exploitations to take advantage of prior knowledge and exploration to look for new options:



Epsilon-Greedy Algorithm

- The Epsilon-Greedy Algorithm makes use of the exploration-exploitation tradeoff by instructing the computer to explore (i.e. choose a random option with probability epsilon) and exploit (i.e. choose the option which so far seems to be the best) the remainder of the time.
- Usually, epsilon is set to be around 10%.
- In this way, as time goes on, and the computer is choosing different options, it will get a sense of which choices are returning it with the highest reward.

- However, from time to time it will choose a random action just to make sure that it's not missing anything.
- Using this learning algorithm, the computer can converge to the optimal strategy for whatever situation it's trying to learn.
- We define the “choose” function which generates a random number between 0 and 1.
- If it's greater than epsilon, it directs us to exploit function. Otherwise, it directs us to the explore function.

Challenges

- The outcome of your actions may be uncertain
 - You may not be able to perfectly sense the state of the world
 - The reward may be stochastic
-
- You may have no clue (model) about how the world responds to your actions
 - You may have no clue (model) of how rewards are being paid off
 - The world may change while you try to learn it

- A deterministic Markov decision process is one in which the state transitions are deterministic (an action performed in state x_t always transitions to the same successor state x_{t+1}).
- Alternatively, in a nondeterministic Markov decision process, a probability distribution function defines a set of potential successor states for a given action in a given state.
- If the MDP is non-deterministic, then value iteration requires that we find the action that returns the maximum expected value.
- For example, to find the expected value of the successor state associated with a given action, one must perform that action an infinite number of times, taking the integral over the values of all possible successor states for that action.
- Theoretically, value iteration is possible in the context of non-deterministic MDPs, however, in practice it is computationally impossible to calculate the necessary integrals without added knowledge or some degree of modification.
- **Q-learning solves the problem of having to take the max over a set of integrals.**




What is Q-Learning?

- Q-learning is a model-free, value-based, off-policy algorithm.
- Q-learning finds the Optimal policy by learning the optimal Q-values for each state-action pair.
- Q implies Quality, representing how valuable the selected action is in maximizing future rewards.
- Initially, the agent randomly picks actions.
- But as the agent interacts with the environment, it learns which actions are better, based **on rewards** that it obtains.
- It uses this experience to incrementally **update** the Q values.
- **Temporal Differences(TD)**: used to estimate the expected value of $Q(S_{t+1}, a)$ by using the *current state and action* and *previous state and action*.

Q-function

- The Q-value $Q^\pi(s, a)$ is the expected reward of taking action a in state s and then continuing according to π .

$$Q^\pi(s, a) = \mathbb{E}_{a_{t+i}, s_{t+i}} \left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid s_t = s, a_t = a \right]$$

| | | | |
|---|---|--|---|
| $Q^\pi(s_t, a_t)$ | = | $\underline{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots]$ | s_t, a_t |
|  | |  |  |
| Q-Values for the state given a particular state | | Expected discounted cumulative reward | Given the state and action |

- We define the optimal Q-function $Q^*(s, a)$ as the maximum return that can be obtained starting from observation s , taking action a and following the optimal policy thereafter.

- The optimal Q-function obeys the following *Bellman* optimality equation:

$$Q^*(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a')]$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

New Q-value estimation
Former Q-value estimation
Learning Rate
Immediate Reward
Discounted Estimate optimal Q-value of next state
Former Q-value estimation

$$q^{new}(s, a) = (1 - \alpha) \underbrace{q(s, a)}_{\text{old value}} + \alpha \overbrace{\left(R_{t+1} + \gamma \max_{a'} q(s', a') \right)}^{\text{learned value}}$$

In the beginning, the agent has no idea about the environment.

He is more likely to explore new things than to exploit his knowledge because...he has no knowledge.

Through time steps, the agent will get more and more information about how the environment works and then, the agent is more likely to exploit knowledge than exploring new things.

If we skip this important step, the Q-Value function will converge to a local minimum which in most of the time, is far from the optimal Q-value function.

To handle this, we will have a threshold which will decay every episode using exponential decay formula.

Updating Q value

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

Where:

- $Q(s, a)$: The Q-value for taking action a in state s .
- α : The learning rate, which determines how much new information overrides the old Q-value.
- $R(s, a)$: The immediate reward received after taking action a in state s .
- γ : The discount factor, which weighs future rewards relative to immediate rewards.
- $\max_{a'} Q(s', a')$: The maximum Q-value for the next state s' , across all possible actions a' .

Getting Stuck in Local Optima

- It is very important that the agent does not simply follow the current policy when learning Q. (off-policy learning).
 - The reason is that you may get stuck in a suboptimal solution, that is, there may be other solutions out there that you have never seen.
- Hence it is good to try new things so now and then.
 - For example, we could use something like:

$$P(a | s) \propto e^{\hat{Q}(s,a)/T}$$

- Recall simulated annealing
 - If T is large lots of exploring, if T is small, follow current policy.
 - One can decrease T over time.

Q function is Q-Table

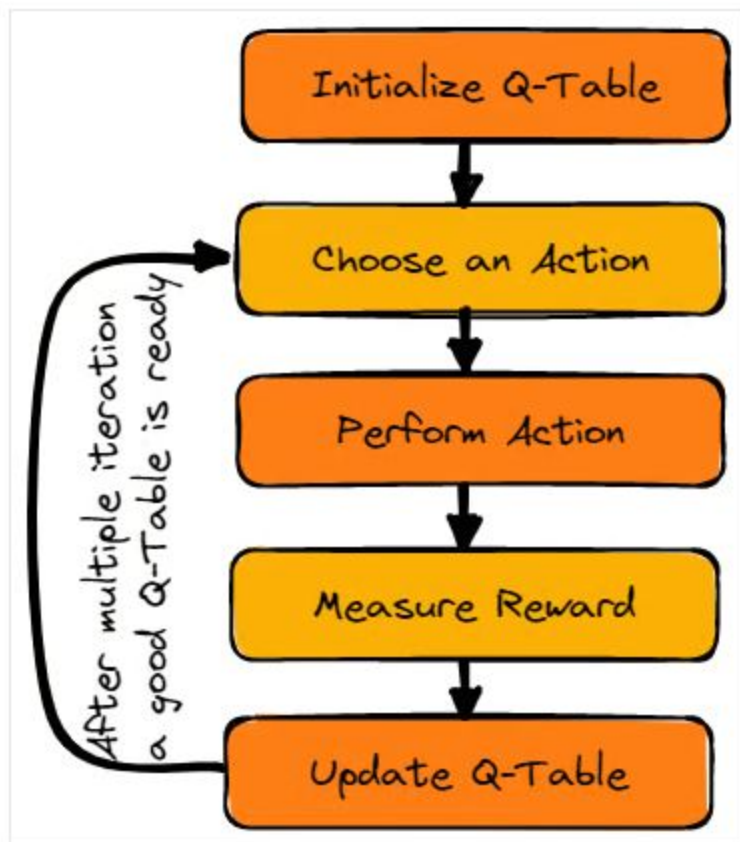
| | | | | |
|-------|-------|-------|-------|-------|
| | a_1 | a_2 | a_3 | a_4 |
| s_1 | 10 | 52 | 15 | -2 |
| s_2 | 14 | 30 | 8 | 7 |
| s_3 | 42 | 0 | -5 | -10 |
| s_4 | -3 | -1 | -7 | -20 |

↑
states
↓

← *actions* →

The reward value is updated during the training such that in steady-state, it should reach the expected value of the reward with the discount factor, which is equivalent to the Q^* value.

In the context of Q-learning, the value of a state is defined to be the maximum Q-value in the given state.



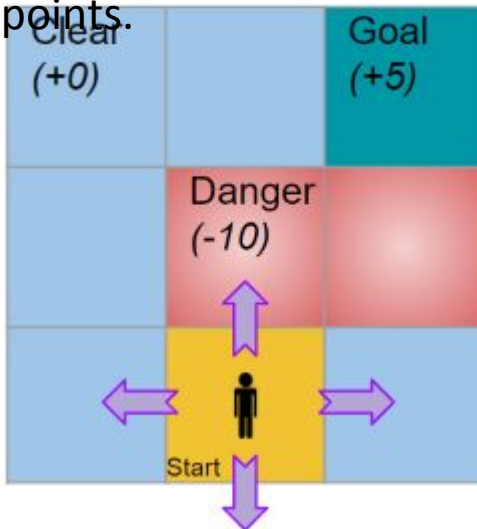
| | | Actions | | | |
|--------|----------|---------------|---------------|----------|---------------|
| | | A_1 | A_2 | ... | A_M |
| States | S_1 | $Q(S_1, A_1)$ | $Q(S_1, A_2)$ | | $Q(S_1, A_M)$ |
| | S_2 | $Q(S_2, A_1)$ | $Q(S_2, A_2)$ | | $Q(S_2, A_M)$ |
| | \vdots | | | \ddots | \vdots |
| | S_N | $Q(S_N, A_1)$ | $Q(S_N, A_2)$ | ... | $Q(S_N, A_M)$ |

- Some squares are Clear while some contain Danger, with rewards of 0 points and -10 points respectively.
- In any square, the player can take four possible actions to move Left, Right, Up, or Down.

Q-table with 9 rows (states) and 4 columns (action).

Initialization

At goal get a reward of 5 points.



| | Left | Right | Up | Down |
|-------|------|-------|----|------|
| (1,1) | 0 | 0 | 0 | 0 |
| (1,2) | 0 | 0 | 0 | 0 |
| (1,3) | 0 | 0 | 0 | 0 |
| (2,1) | 0 | 0 | 0 | 0 |
| (2,2) | 0 | 0 | 0 | 0 |
| (2,3) | 0 | 0 | 0 | 0 |
| (3,1) | 0 | 0 | 0 | 0 |
| (3,2) | 0 | 0 | 0 | 0 |
| (3,3) | 0 | 0 | 0 | 0 |

- 1 Initialise Q-Value (ie. State Action value) estimates with zero value, and pick the initial state.

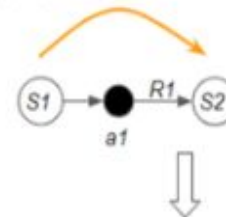
| | a1 | a2 | a3 | a4 |
|----|----|----|----|----|
| S1 | 0 | 0 | 0 | 0 |
| S2 | 0 | 0 | 0 | 0 |
| S3 | 0 | 0 | 0 | 0 |
| S4 | 0 | 0 | 0 | 0 |
| S5 | 0 | 0 | 0 | 0 |

S1
+ initial state

S2
Next state becomes
the Current State

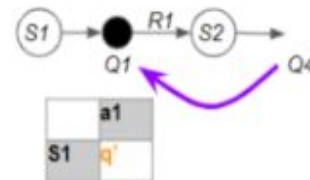
- 2 Agent picks an **action to execute** from the current state using ϵ -greedy policy.

- 3 Agent obtains observation data from the environment (S1, a1, R1, S2)

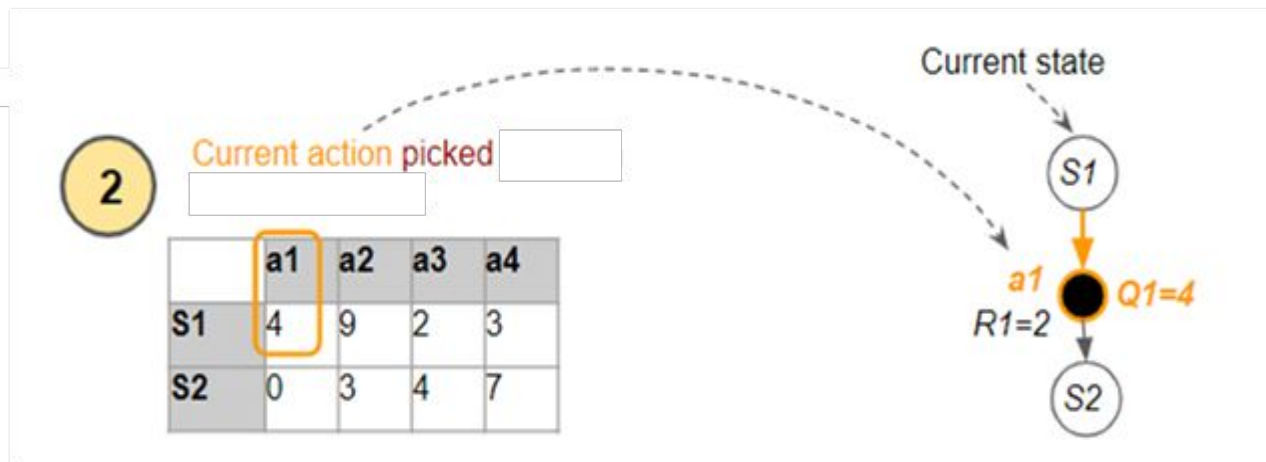


- 4 Update the **current Q-value** using the observed reward and the **target Q-value**. The target Q-value is the action with the max Q-value from the next state.

$$Q(S, A) = Q(S, A) + \alpha(R + \gamma \max Q(S', a) - Q(S, A))$$

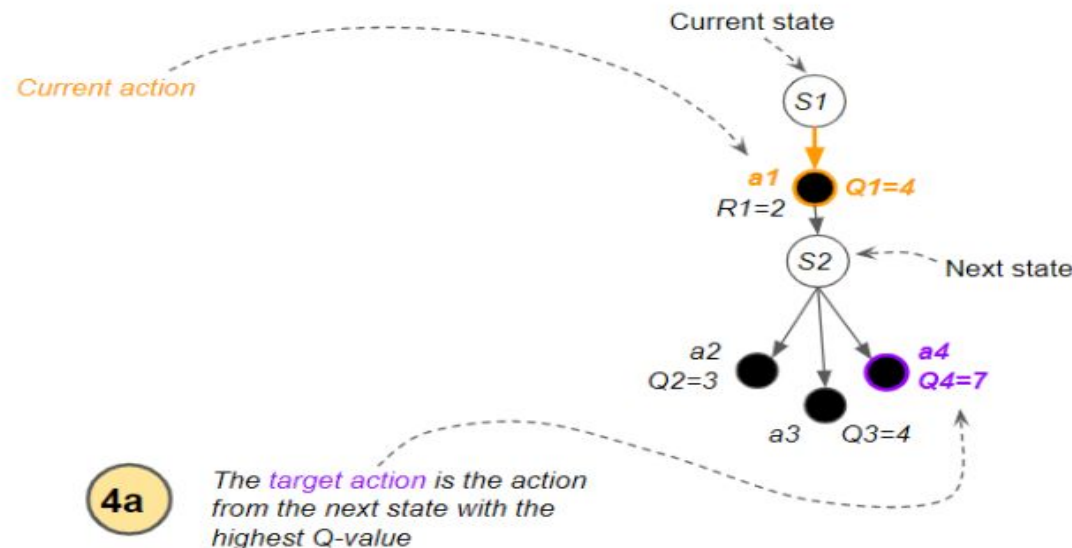


The agent uses the ϵ -greedy policy to pick the current action ($a1$) from the current state ($S1$). This is the action that it passes to the environment to execute, and gets feedback in the form of a reward ($R1$) and the next state ($S2$).



Estimated Q-value ($Q1$) for the current state

The next state has several actions, so which Q-value does it use?
It uses the action (a_4) from the next state which has the highest Q-value (Q_4).
What is critical to note is that it treats this action as a target action to be used only for the update to Q_1 .



| | a1 | a2 | a3 | a4 |
|----|----|----|----|----|
| S1 | 4 | 9 | 2 | 3 |
| S2 | 0 | 3 | 4 | 7 |

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

New
Q-value
estimation

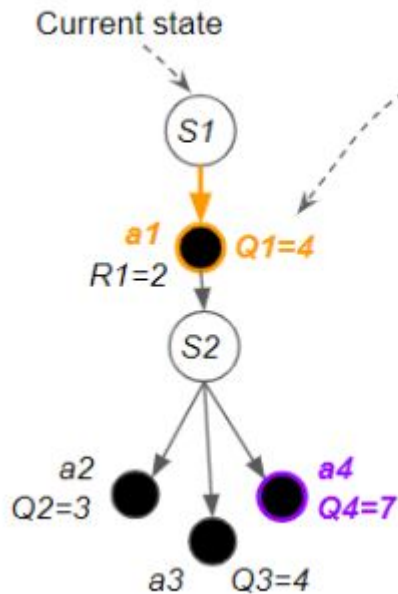
Former
Q-value
estimation

Learning
Rate

Immediate
Reward

Discounted Estimate
optimal Q-value
of next state

Former
Q-value
estimation



4b

The **current Q-value** is updated

$$Q(S, A) = Q(S, A) + \alpha (R + \gamma \max Q(S', a) - Q(S, A))$$

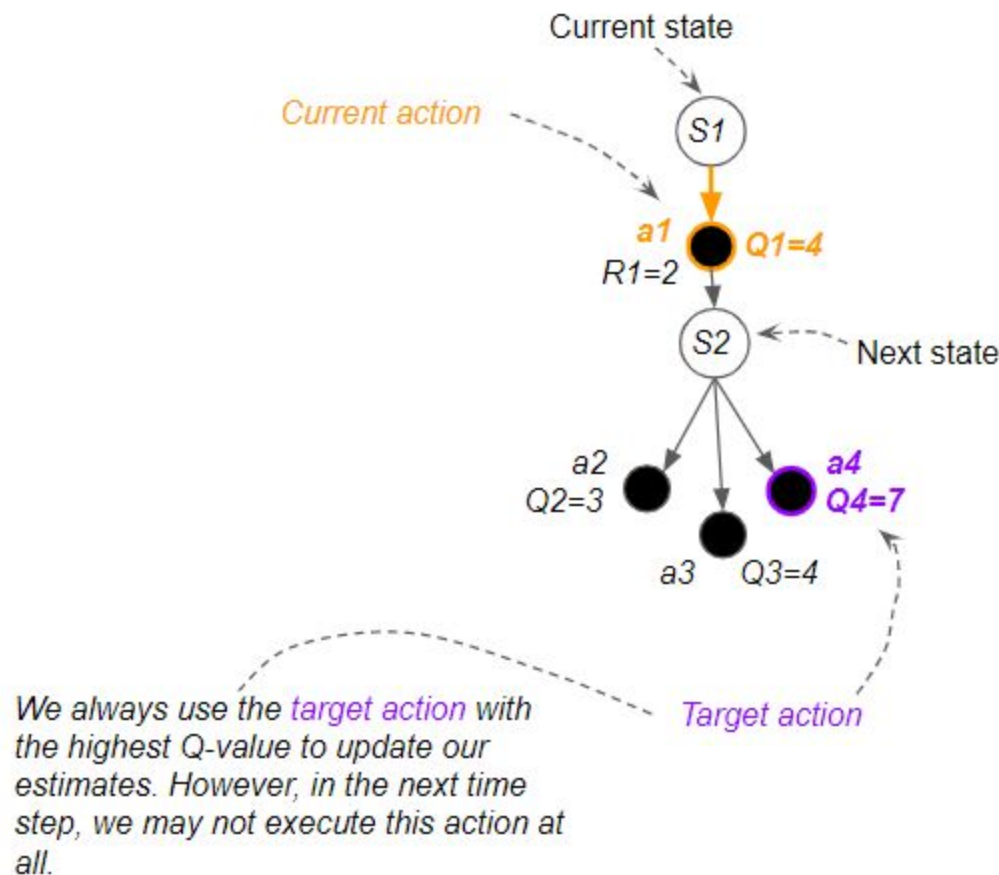
$$Q1 = Q1 + \alpha (R1 + \gamma * Q4 - Q1)$$

$$Q1 = 4 + (2 + 7 - 4) = 9$$

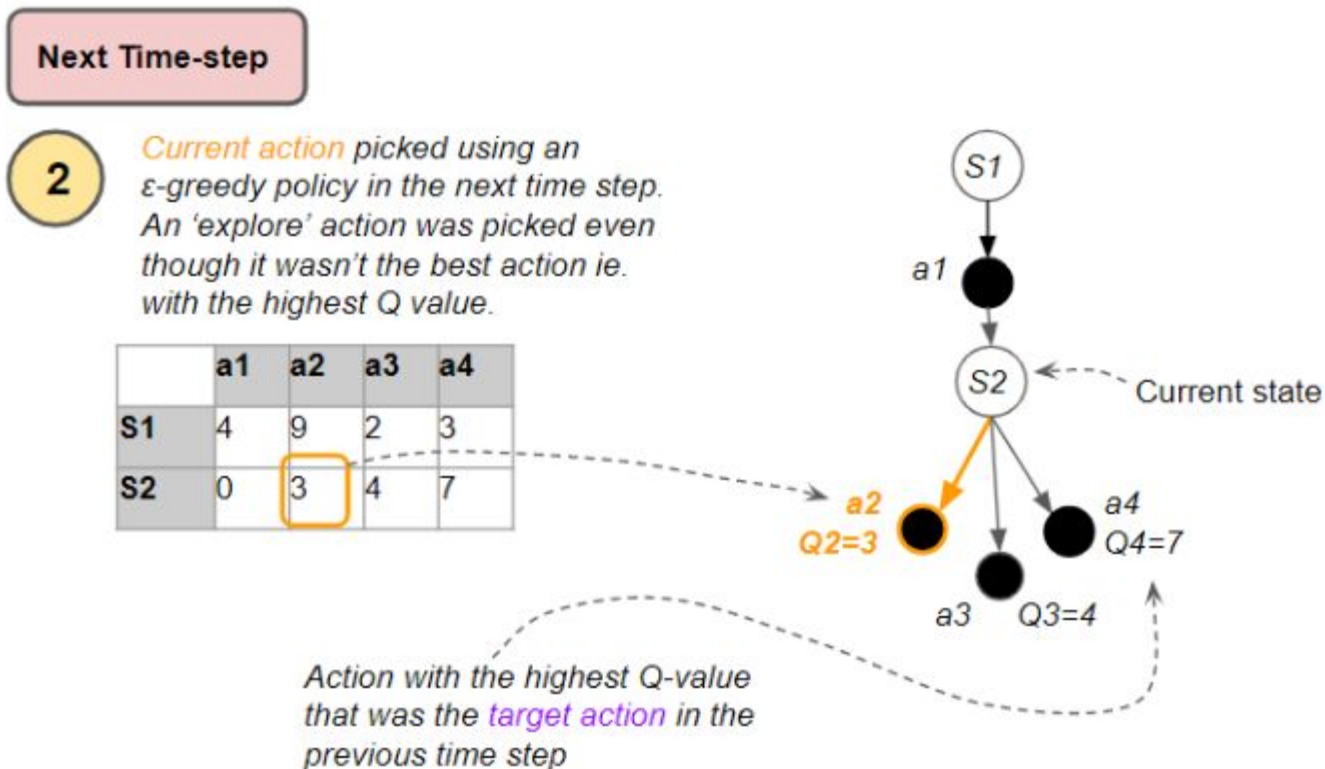
NB: taking $\alpha = \gamma = 1$ for simplicity

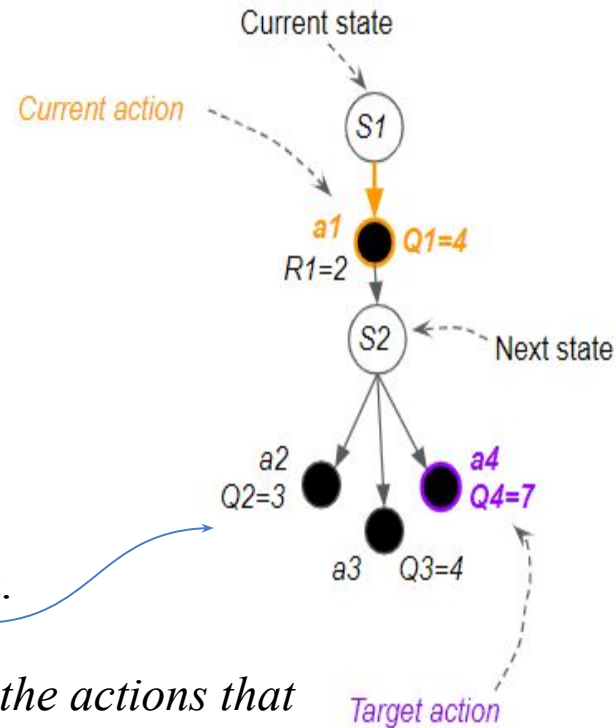
| | a1 | a2 | a3 | a4 |
|----|----|----|----|----|
| S1 | 4 | 9 | 2 | 3 |
| S2 | 0 | 3 | 4 | 7 |

- In other words, there are two actions involved:
- Current action – the action from the current state that is actually executed in the environment, and whose Q-value is updated.
- Target action – has the highest Q-value from the next state, and used to update the current action's Q value.



- Now the next state has become the new current state.
- The agent again uses the ϵ -greedy policy to pick an action.
- The action that it executes (a2) will be different from the target action (a4) used for the Q-value update in the previous time-step.



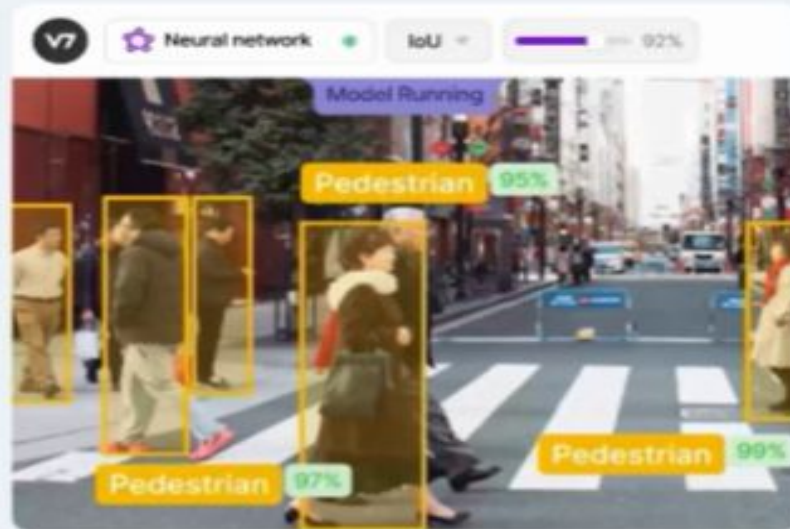


An explore action is chosen, though not best.

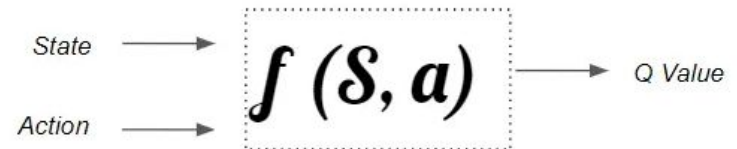
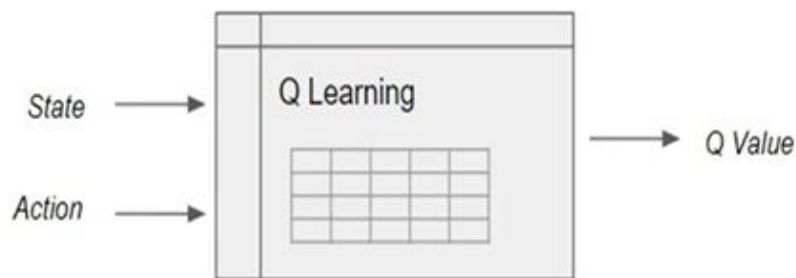
This is known as 'off-policy' learning because the actions that are executed are different from the target actions that are used for learning.

- Start by taking a particular action from a particular state, then follow the policy after that till the end of the episode, and then measure the Return.
- It helps an agent learn to maximize the total reward over time through repeated interactions with the environment, even when the model of that environment is not known.
- And if you did this many, many times, over many episodes, the Q-value is the average Return that you would get.

Label, train, and test computer vision models to **solve any task**



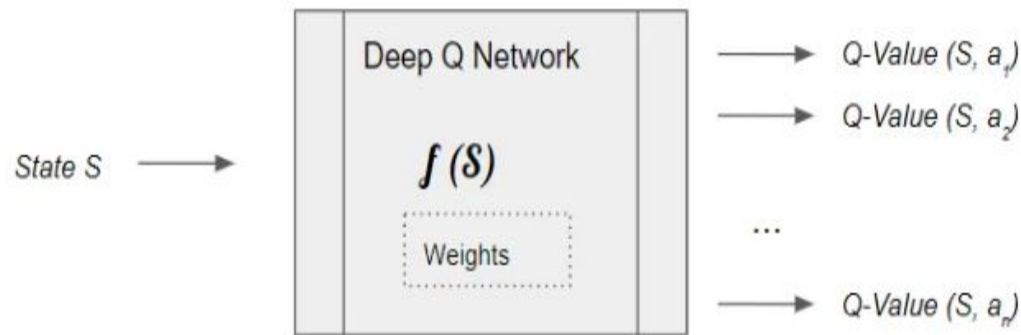
- Reinforcement Learning involves managing state-action pairs and keeping a track of value (reward) attached to an action to determine the optimum policy.
- This method of maintaining a state-action-value table is not possible in real-life scenarios when there are a larger number of possibilities.
- Instead of utilizing a table, we can make use of Neural Networks to predict values for actions in a given state.
- Use a Q-function rather than a Q-table, which achieves the same result of a Q value.



Instead, we train a function approximator, such as a neural network with parameters θ , to estimate the Q-values, i.e. $Q(s,a;\theta)$

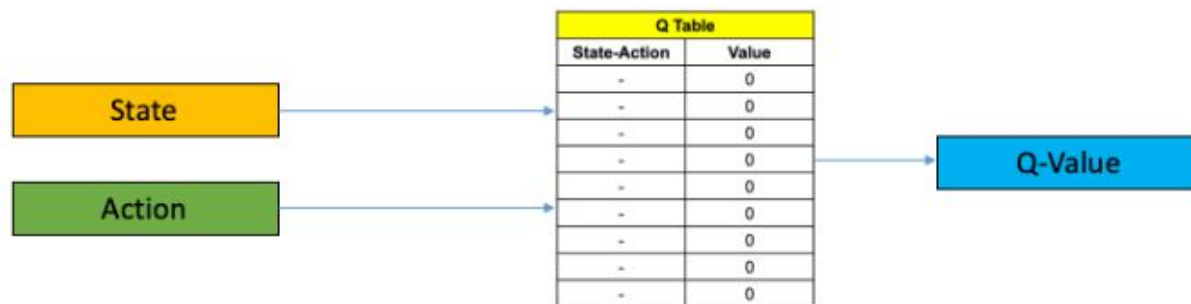
Neural Nets are the best Function Approximators

- Since neural networks are excellent at modeling complex functions, we can use a neural network, which we call a Deep Q Network, to estimate this Q function.
- This function maps a state to the Q values of all the actions that can be taken from that state.

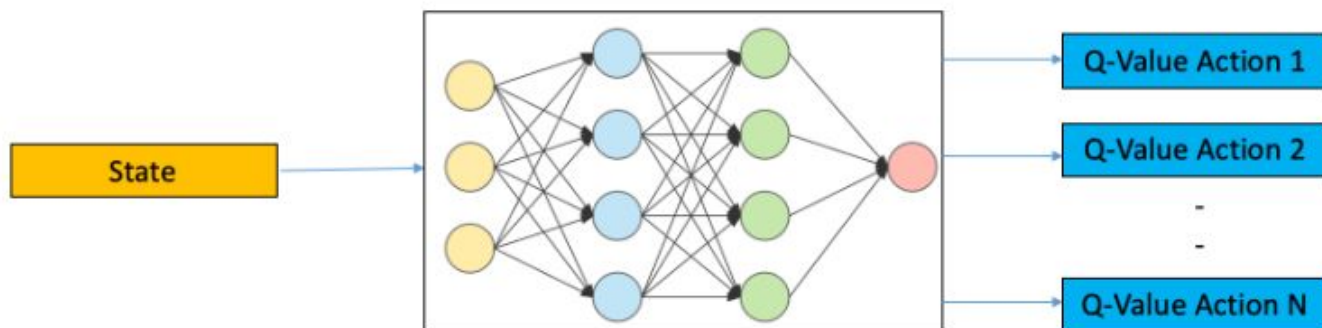


(Image by Author)

It learns the network's parameters (weights) such that it can output the Optimal Q values.



Q Learning



Deep Q Learning