Reinforcement Learning

Penyusun Modul: Chairul Aulia

Editor: Citra Chairunnisa

Need for Reinforcement Learning

A major drawback of machine learning is that a **tremendous amount of data is needed** to train models. The more complex a model, the more data it may require. But this data may not be available to us. It may not exist or we may not have access to it. Further, the data collected might not be reliable. It may have false or missing values or it might be outdated.

All of these problems are overcome by reinforcement learning. In reinforcement learning, we introduce our model to a controlled environment which is modeled after the problem statement to be solved instead of using actual data to solve it.

Reinforcement Learning

Reinforcement learning is probably one of the most relatable scientific approaches that resemble the way humans learn about things. Every day, we, the learner, learn by **interacting with our environment** to know what to do in certain situations, to know about the **consequences of our actions**, and so on.

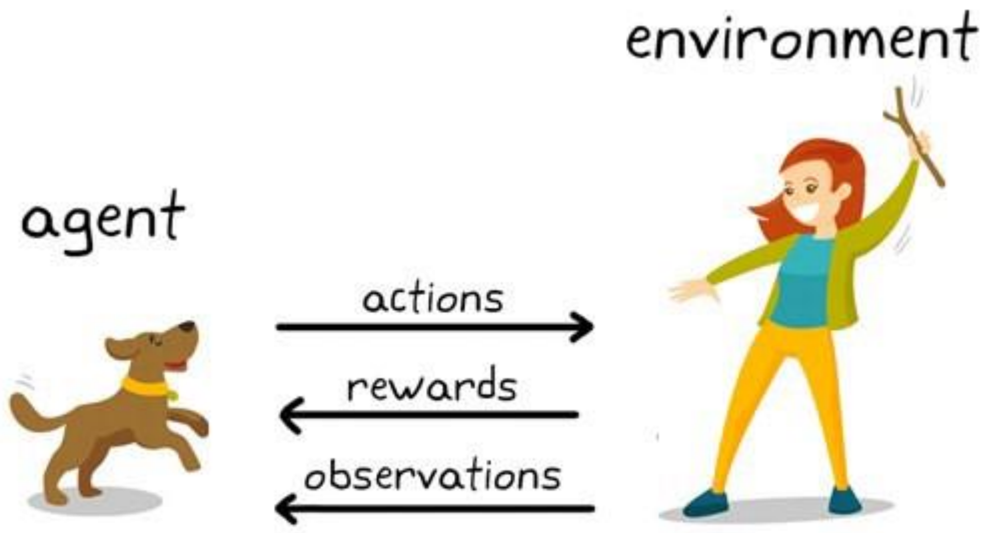
Reinforcement learning is learning what to do in order to **maximize a numerical reward**. This means that the learner should discover **which actions** that **yield the highest reward** in the long run by trying them.

Comparison

Criteria	Supervised ML	Unsupervised ML	Reinforcement ML
Definition	Learns by using labelled data	Trained using unlabelled data without any guidance.	Works on interacting with the environment
Type of data	Labelled data	Unlabelled data	No – predefined data
Type of problems	Regression and classification	Association and Clustering	Exploitation or Exploration
Supervision	Extra supervision	No supervision	No supervision
Algorithms	Linear Regression, Logistic Regression, SVM, KNN etc.	K – Means, C – Means, Apriori	Q – Learning, SARSA
Aim	Calculate outcomes	Discover underlying patterns	Learn a series of action
Application	Risk Evaluation, Forecast Sales	Recommendation System, Anomaly Detection	Self Driving Cars, Gaming, Healthcare

Reinforcement Learning

Let's consider the analogy of teaching a dog. In this scenario, we emulate a situation and the dog tries to respond in different ways. If the dog's response is a desired one, we **reward** it with dog food. If the response isn't the desired one, we simply say "No" as a **penalty**.
Now, every time the dog is exposed to the same situation, the dog executes **a similar action** with even more enthusiasm in expectation of more food. It is basically learning **what to do** from **positive** experiences. Similarly, it will tend to learn **what not to do** when coming face to face with **negative** experiences.



Reinforcement Learning

Being greedy doesn’t always work

There are things that are easy to do for instant gratification, and there are things that provide long term rewards. The goal is to not looking for quick immediate rewards, but instead optimize for maximum rewards over the whole training.

Sequence matters in Reinforcement Learning

The reward agent does not just depend on the current state, but the entire history of states. Unlike supervised and unsupervised learning, time is important here.

Reinforcement Learning

Reinforcement Learning is the science of making optimal decisions using experiences. Breaking it down, the process involves these simple steps:

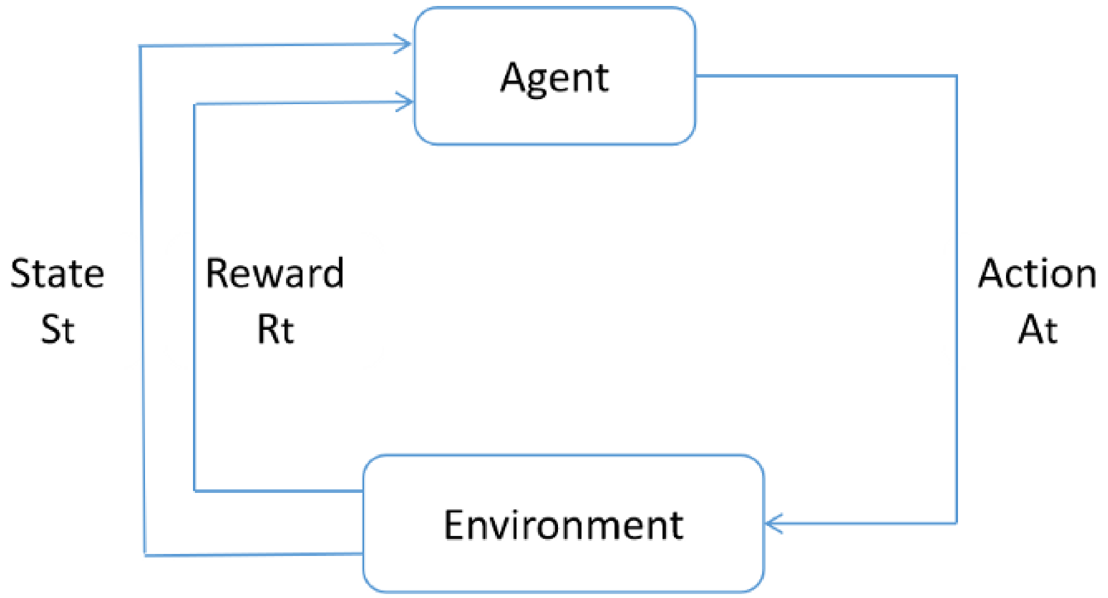
1. *Observation of the environment*
2. *Deciding how to act using some strategy*
3. *Acting accordingly*
4. *Receiving a reward or penalty*
5. *Learning from the experiences and refining our strategy*
6. *Iterate until an optimal strategy is found*

Terminologies in RL

- **Agent** – is the sole decision-maker and learner
- **Environment** – a real or simulation world where an agent learns and decides the actions to be performed
- **Action** – a list of action which an agent can perform
- **State** – the current situation of the agent in the environment
- **Reward** – For each selected action by agent, the environment gives a reward. It’s usually a scalar value and nothing but feedback from the environment. The reward can be positive or negative
- **Policy** – the agent prepares strategy(decision-making) to map situations to actions.

Markov Decision Process

Markov's Decision Process is a Reinforcement Learning policy used to map a current state to an action where the agent continuously interacts with the environment to produce new solutions and receive rewards



The main goal of the Agent is take actions that will maximize your future reward. So the flow is:

1. Take an action;
2. Receive a feedback from environment;
3. Receive the new state;
4. Take a new Action;

Markov Decision Process

Markov's Decision Process (MDP) uses:

- A set of States (S)
- A set of Models
- A set of all possible actions (A)
- A reward function that depends on the state and action R(S, A)
- A policy which is the solution of MDP

The policy of Markov's Decision Process aims to maximize the reward at each state. The Agent interacts with the Environment and takes Action while it is at one State to reach the next future State. We base the action on the maximum Reward returned.

Intermezzo

Before we continue to our algorithm, let’s check the video in the link below to help you visualize the Reinforcement Learning in practice where we have 4 agents playing hide and seek!

https://www.youtube.com/watch?v=kopoLzvh5jY

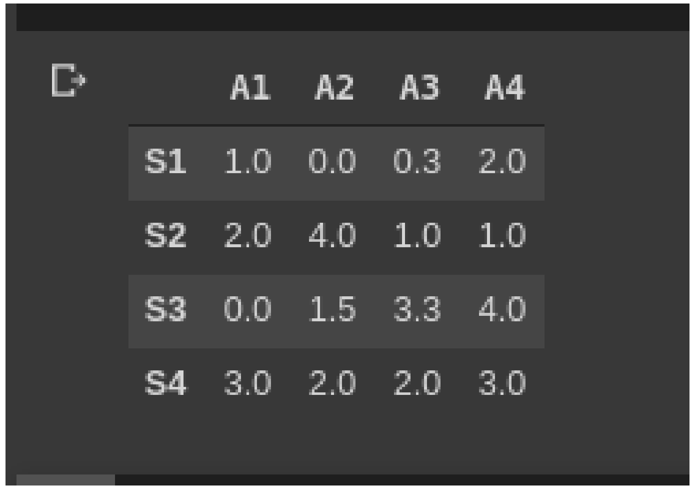
Q-Learning

Q-learning is a model-free, off-policy reinforcement learning that will find the best course of action, given the current state of the agent. Depending on where the agent is in the environment, it will decide the next action to be taken. Some importants terms in Q-Learning:

- States: The State, S, represents the current position of an agent in an environment.
- Action: The Action, A, is the step taken by the agent when it is in a particular state.
- Rewards: For every action, the agent will get a positive or negative reward.
- Episodes: When an agent ends up in a terminating state and can't take a new action.
- Q-Values: Used to determine how good an Action, A, taken at a particular state, S, is. Q (A, S).
- Temporal Difference: A formula used to find the Q-Value by using the value of current state and action and previous state and action.

Q-Table

Q-Learning is a very simple to understand algorithm and very recommended to beginners in Reinforcement Learning, because it's powerful and can be apply in a few lines of code.
Basically in Q-Learning, our we create a table with actions and states, called Q-Table. This table will help our agent to take the best action for the moment.
The table looks like this: (columns represent actions, rows represent states)



	A1	A2	A3	A4
S1	1.0	0.0	0.3	2.0
S2	2.0	4.0	1.0	1.0
S3	0.0	1.5	3.3	4.0
S4	3.0	2.0	2.0	3.0

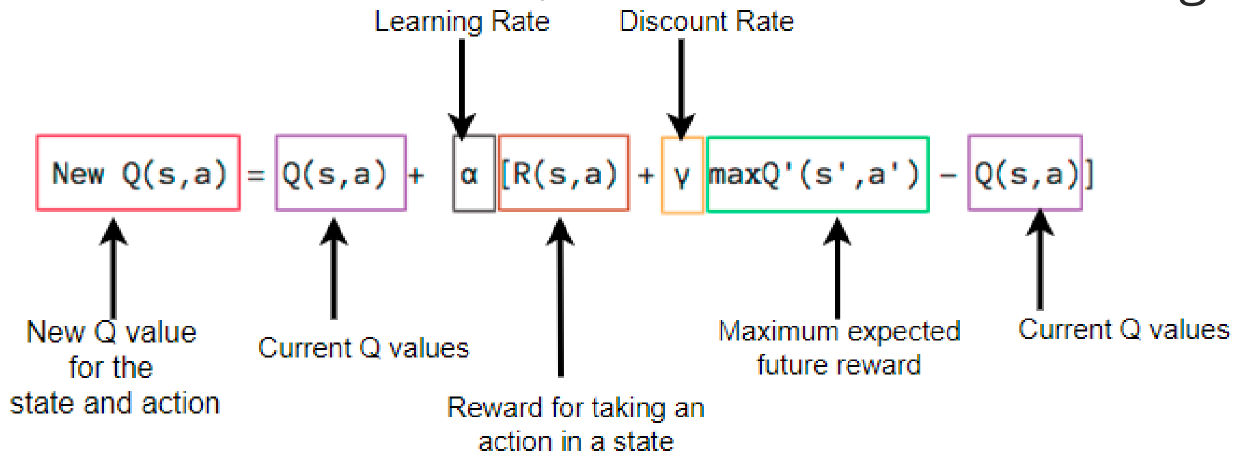
Exploration and Expoitation

In the beginning we start Q table with 0 in all values
The idea is to leave the agent **explore** the environment taking random actions and after, use the rewards received from these actions to populate the table, this is the **Exploration**.
After that, we start the **Exploitation**, where the agent use the table to take actions who will maximize him future reward.

Whenever our agent receive a reward from the environment, that can be negative or positive, we'll use the Bellman Equation to update our Q-Table:

The Bellman Equation

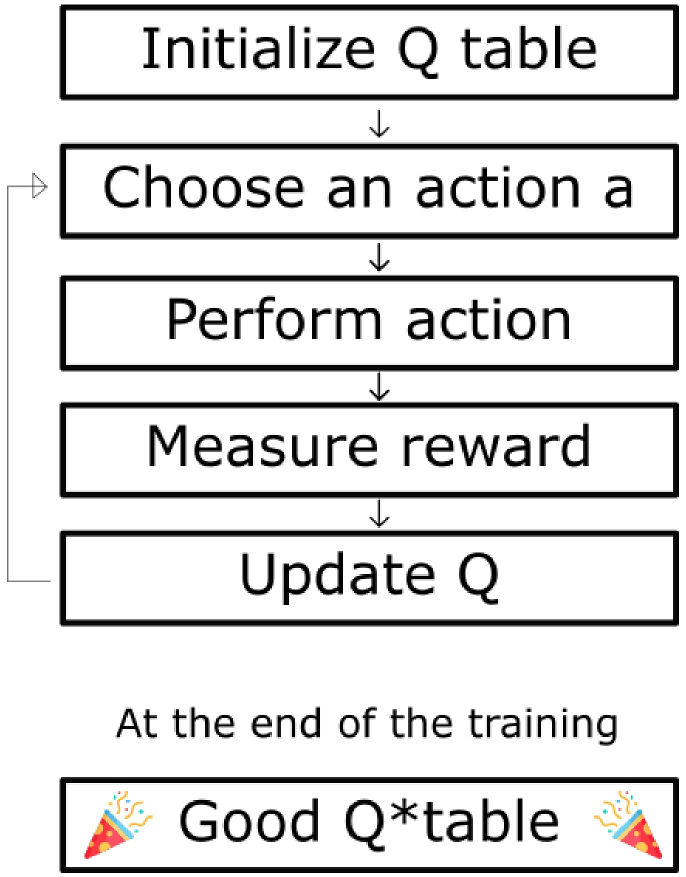
The Bellman Equation is used to determine the value of a particular state and deduce how good it is to be in/take that state. The optimal state will give us the highest optimal value.
The equation is given below. It uses the current state, and the reward associated with that state, along with the maximum expected reward and a discount rate, which determines its importance to the current state, to find the next state of our agent. The learning rate determines how fast or slow, the model will be learning.



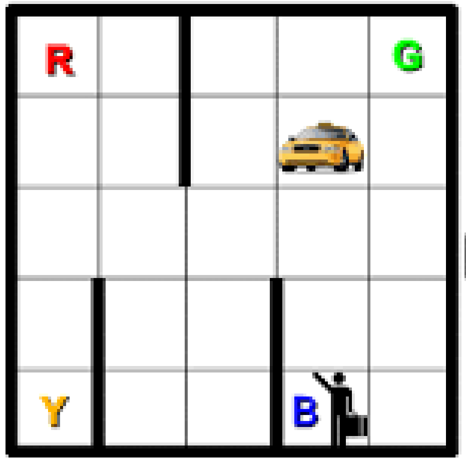
$$\text{New } Q(s,a) = Q(s,a) + \alpha\,[R(s,a) + \gamma \max Q'(s',a') - Q(s,a)]$$

Learning Rate
Discount Rate
New Q value for the state and action
Current Q values
Reward for taking an action in a state
Maximum expected future reward
Current Q values

Q-Learning Algorithm Steps



Example: Taxi Problem



4 designated locations in the grid world indicated by R(ed), B(lue), G(reen), and Y(ellow). When the episode starts, the taxi and passenger start off at a random location. The taxi needs to pick up and drop off the passenger.
Actions: There are 6 discrete deterministic actions:

0: move south	3: move west
1: move north	4: pickup passenger
2: move east	5: dropoff passenger

There are 500 discrete **states** (25 taxi positions x 5 possible locations of the passenger (including when the passenger is in the taxi) x 4 destination locations)
Rewards: -1 for execute each action, +20 for delievering the passenger, -10 for "pickup" and "dropoff" illegally.
Color code:

- blue: location where the passenger needs to be picked up
- magenta: destination
- yellow: empty taxi
- green: full taxi

Exploration Exploitation Trade-off

Suppose the reward is to feel the delicious food in a restaurant and the punishment is to taste the unsavory food.

Exploration means you need to randomly taste all the possible menu and you will get to try all the best serves in that restaurant eventually. But if there is enormous amount of menu available, the accumulation of feeling satisfied might reduce due to the amount of randomly tasting the not so favorite ones.

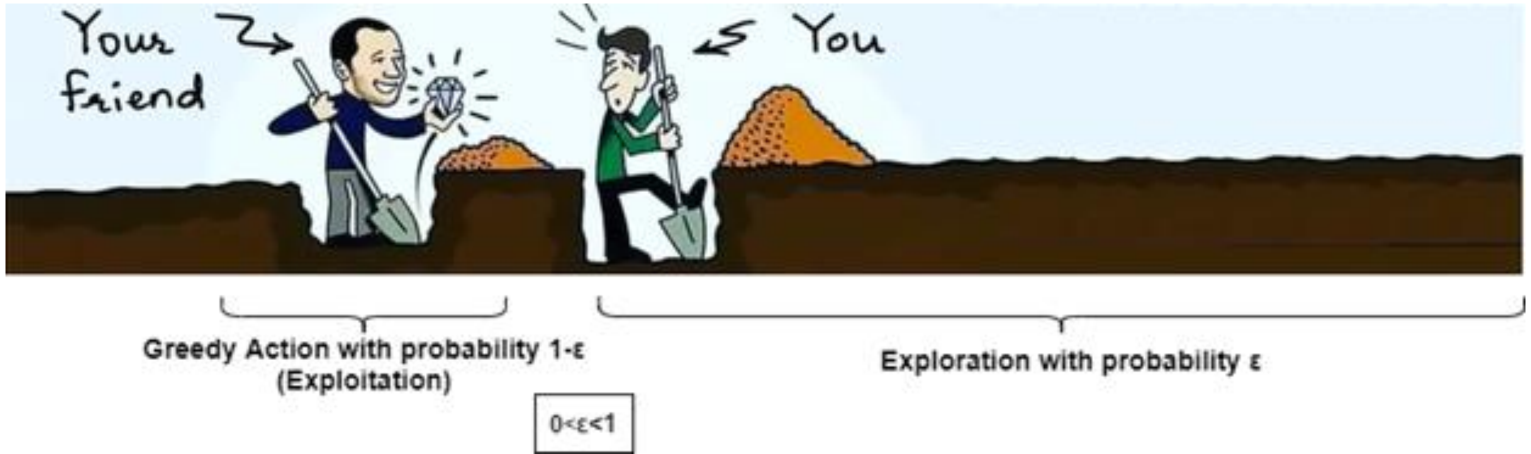
Exploitation means that you already have the info of some of your favorite food, and you just order those you know the taste repeatedly. You might reduce the unsatisfaction feeling of trying the unwanted food, but you will not find the more delicious food other than what you already know, and you might end up not having the highest reward.

And this also applies to reinforcement learning. In order to overcome the **Exploration-Exploitation Dilemma**, we use the **Epsilon Greedy Policy**.

Epsilon Greedy Policy

$$A_t \leftarrow \begin{cases} argmax\ Q_t(a) & with\ probability\ 1-\epsilon \\ a \sim Uniform(\{a_1 \ldots a_k\}) & with\ probability\ \epsilon \end{cases}$$

The Action that the agent selects at time step t, will be a greedy action (exploit) with probability (1-epsilon) or may be a random action (explore) with probability of epsilon.


Your friend
You
Greedy Action with probability 1-ε (Exploitation)
Exploration with probability ε
0<ε<1

Most Common Applications of RL

- Robotics for Industrial Automation
- Text summarization engines, dialogue agents (text, speech), gameplays
- Autonomous Self Driving Cars
- AI Toolkits, Manufacturing, Automotive, Healthcare, and Bots
- Aircraft Control and Robot Motion Control

You can see that the most popular domains are gaming and robotics

Exercise

Let's see the implementation of Q-Learning in Taxi cab problem
We'll be using OpenAI Gym library which is a collection of test problems — environments — that you can use to work out your reinforcement learning algorithms.

Of course there is no dataset since RL is learning by reward not data!