**6 Reinforcement Learning Algorithms Explained**

**Introduction to reinforcement learning terminologies, basics, and concepts (model-free, model-based, online, offline RL)**



Photo by [Ryan Fields](https://unsplash.com/@rfieldss?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

Machine Learning (ML) is split into three branches: Supervised Learning, Unsupervised Learning, and Reinforcement Learning.

* **Supervised Learning (SL)**: Concerned…
* **Unsupervised Learning (UL)**: Concerned with discovering patterns in the data without pre-existing labels
* **Reinforcement Learning (RL)**: Concerned with how agents take actions in an environment to maximize cumulative reward

In layman's terms, Reinforcement Learning is akin to a baby learning and discovering the world, where the baby is likely to perform an action if there is a reward (positive reinforcement) and less likely to perform an action if there is punishment (negative reinforcement). This is also the main difference between Reinforcement Learning from Supervised and Unsupervised Learning, where the latter learns from a static dataset, whereas the former learns from exploration.

This article will touch on the terminologies and basic components of Reinforcement Learning, and the different types of Reinforcement Learning (Model-free, Model-based, Online Learning, and Offline Learning). This article ends off with algorithms to illustrate the different types of Reinforcement Learning.

Note: The equations are based on the textbook Artificial Intelligence: A Modern Approach (Fourth Edition, Global Edition) by Stuart J. Russell and Peter Norvig with minor changes to keep the mathematical equation format consistent.

**Table of Contents**

* [Reinforcement Learning Terminologies](https://medium.com/p/237a79dbd8e/#a72c)
* [Basics: Markov Decision Process (MDP)](https://medium.com/p/237a79dbd8e/#7922)
* [Model-free vs. Model-based Reinforcement Learning](https://medium.com/p/237a79dbd8e/#652b)
* [Offline Learning vs. Online Learning](https://medium.com/p/237a79dbd8e/#eabf)
* [Algorithm: Direct Utility Estimation](https://medium.com/p/237a79dbd8e/#bab1)
* [Algorithm: Adaptive Dynamic Programming (ADP)](https://medium.com/p/237a79dbd8e/#95b3)
* [Algorithm: Temporal-Difference Learning (TD Learning)](https://medium.com/p/237a79dbd8e/#8eb7)
* [Algorithm: Exploration](https://medium.com/p/237a79dbd8e/#e32b)
* [Algorithm: Q-Learning](https://medium.com/p/237a79dbd8e/#3ac8)
* [Algorithm: SARSA](https://medium.com/p/237a79dbd8e/#54c5)

**Reinforcement Learning Terminologies**

Before diving into the different types of Reinforcement Learning and Algorithms, we should familiarize ourselves with the components of Reinforcement Learning.

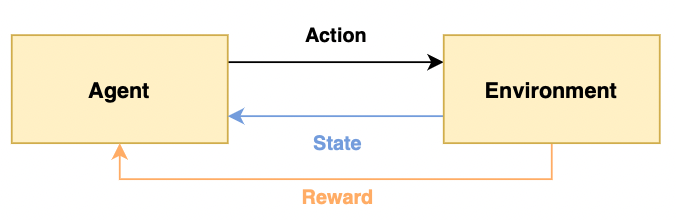


Fig 1: Illustration of Reinforcement Learning Terminologies — Image by author

* **Agent**: The program that receives percepts from the environment and performs actions
* **Environment**: The real or virtual environment that the agent is in
* **State (S)**: The state that an agent can be in
* **Action (A)**: The action that an agent can take when in a given state
* **Reward (R)**: The reward of taking an action (dependent on the action), the reward of being in a state (dependent on the state), or the reward of taking an action in a given state (dependent on the action and state)

In the example of a baby exploring the world, the baby (agent) is in the real world (environment) and can be crying, feeling happy, or hungry (state). The baby can therefore choose to eat or sleep (action) and the baby is fulfilled if the baby gets to eat when he/she is hungry (reward).

As mentioned at the start of the article, Reinforcement Learning involves exploration, and the output of Reinforcement Learning is an optimal policy. A **policy** describes the action to take at every state; akin to an instruction manual. For example, the policy can be to eat when the baby is hungry, otherwise, the baby should sleep. This also contrasts Supervised Learning where the output is only a single decision or prediction, which is less complex than a policy.

Finally, the **goal** of Reinforcement Learning is to maximize the total cumulative reward by optimizing the actions taken. Same as the baby, don’t we all want to reap the maximum cumulative benefits from life? ;)

**Basics: Markov Decision Process (MDP)**

As Reinforcement Learning involves making a series of optimal actions, it is considered a **sequential decision problem** and can be modelled using Markov Decision Process.

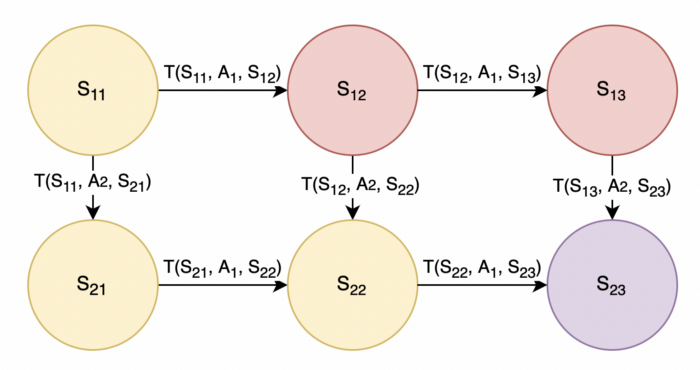


Fig 2: Example of MDP — Image by author

Following the previous section, the states (denoted by **S**) are modeled as circles, and actions (denoted by **A**) allow the agent to transition between states. In Fig 2, there is also a transition probability (denoted by **T**) where T(S11, A1, S12) is the probability of transitioning to state S12 after taking action A1 at state S11. We can think of action A1 as going rightwards and action A2 as going downwards. For simplicity, we can assume the transition probability is 1, such that taking action A1 will guarantee a move rightwards, and taking action A2 will guarantee a move downwards.

Referencing Fig 2, let the goal to be to end at state S23 starting at state S11, and yellow states are good (reward +1), red states are bad (reward -1), and purple is the goal state (reward +100). We want the agent to learn that the optimal action or route is to go Down-Right-Right by taking actions A2-A1-A1 and reap a total reward of +1+1+1+100. Taking it one step further, using the time value of money, we apply a discount factor gamma, on the rewards since a reward now is better than a reward later.

Putting it all together, the mathematical formula for the expected utility by executing actions A2-A1-A1 starting in state S11 is as follows,

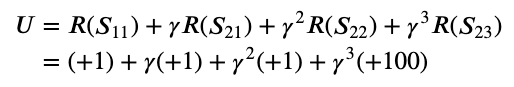


Fig 3: Expected utility starting in state S11 — Image by author

The above example is a simple illustration, there are variations such as,

* The transition probability may not be 1, there is a need to factor in uncertainty in actions such as taking certain actions may not always guarantee a successful move rightwards or downwards. Therefore, we need to take an expected value over this uncertainty
* The optimal action may not be known yet, therefore the generic representation would be to represent an action as a policy from the state, denoted by **π(S)**
* The reward may not be based on the yellow/red/purple state, it could be based on a combination of the previous state, action, and next state, denoted by **R(S1, π(S1), S2)**
* The problem may not be solved within 4 steps, it could take an infinite amount of steps to reach the goal state

Considering these variations, the more general equation that determines expected utility **U(s)** at a given state **s** following policy **π** is as such,

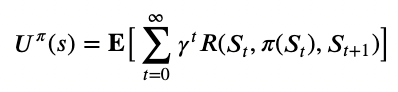


Fig 4: Expected utility by executing policy starting in state s (Equation 16.2) — Image by author

To put Fig 4 into words, the expected utility of a state is the expected sum of discounted rewards.

It follows that a state's utility is related to its neighbours' utility; the utility of a state is the expected reward for the transition plus the discounted utility of the next state, assuming optimal action is chosen. In coding terms, this is considered recursion. Mathematically, it refers to the equation below,

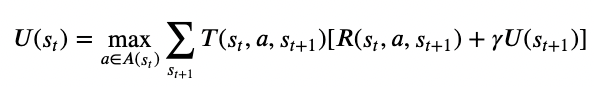


Fig 5: Utility of a state following optimal policy (Equation 16.5) — Image by author

In Fig 5, this is the famous **Bellman equation** that solves for the maximum utility and derives the optimal policy. The optimal policy is the action to take in a state such that it will lead to maximum current utility plus the discounted utility of the next state, taking into account the transition probabilities, summed across all possible next states.

Bringing back the MDP problem in Fig 2, the optimal policy is such that if the agent is in states S11, S12, or S13, the agent should move downwards by taking action A2. Whereas if the agent is in state S21 or S22, the agent should move rightwards by taking action A1. The optimal policy is derived by solving the Bellman equation, to execute the action that reaps the maximum current and discounted future rewards.

* **Extra**: In textbooks, MDP is represented using (S, A, T, R) which represents a set of states, actions, the transition function, and the reward function respectively.
* **Extra**: MDP assumes that the environment is fully observable, if the agent does not know what current state it is in, we would use **Partially Observable MDP (POMDP)** instead!
* **Extra**: The Bellman equation, in Fig 5, can be used to solve for the optimal policy using **Value Iteration** or **Policy Iteration**, which is an iterative method to pass the utility values from a future state to the current state.

Reinforcement Learning is similar to solving an MDP, but now the transition probabilities and reward function are unknown, and the agent has to perform actions to learn

**Model-free vs. Model-based Reinforcement Learning**

The MDP example in the previous section is Model-based Reinforcement Learning. Formally, **Model-based Reinforcement Learning** has components transition probability T(s1, a, s2) and reward function R(s1, a, s2), which are unknown and represent the problem to be solved.

* Model-based methods are useful for simulation.
* Examples of Model-based RL include **Value Iteration** and **Policy Iteration** since it uses MDP which has transition probabilities and reward functions.

**Model-free approach**, on the other hand, does not need to know or learn the transition probability to solve the problem. Instead, the agent learns the policy directly.

* Model-free methods are useful for solving real-life problems.
* Examples of Model-free RL include **Q-learning** and **Policy Search** since it learns the policy directly.

**Offline Learning vs. Online Learning**

Offline and Online Learning is also referred to as Passive and Active Learning.

In **Offline (Passive) Learning**, the problem is solved by learning utility functions. Given a fixed policy with unknown transition and reward functions, the agent tries to learn the utility function by executing a series of trials using the policy.

* For example in a self-driving car, given a map and a general direction to follow (**fixed policy**) with faulty controls (**unknown transition probability** — moving forward could result in the car turning a little left or right) and unknown travelling time (**unknown reward function** — assuming reaching destination faster leads to more rewards), the car can do repeated runs to learn what is the average total travelling time (**utility function**).
* Examples of Offline RL include **Value Iteration** and **Policy Iteration** since it uses the Bellman equation (Fig 5) that uses utility functions. Other examples include **Direct Utility Estimation**, **Adaptive Dynamic Programming (ADP)**, and **Temporal-Difference Learning (TD)** which will be elaborated on in later sections.

In **Online (Active) Learning**, the problem is solved by learning to plan or decide. For Model-Based Online RL, there are exploration and exploitation components. In the exploitation stage, the agent behaves like Offline Learning by assuming a fixed policy and learning the utility function. In the exploration stage, the agent performs **Value Iteration** or **Policy Iteration** to update the policy.

* If the policy is updated using **Value Iteration**, the optimal action is extracted using a one-step look-ahead that maximizes utility/value. If the policy is updated using **Policy Iteration**, the optimal policy is available and action can be executed as recommended.
* Taking the same example in the self-driving car, during the exploration stage, the car may learn that the total time taken is faster when travelling on the highway, and choose to drive towards the highway instead of simply going in the general direction (**policy iteration**). In the exploitation stage, the car now travels with a lesser average total time taken (**higher utility**) following the updated policy.
* Examples of Online RL include **Exploration**, **Q-Learning**, and **SARSA** which will be elaborated on in later sections.

Comparing both, Online Learning is preferred when there are too many states and actions such that there are too many transition probabilities. It would be easier to explore and ‘learn as you go’ in Online Learning rather than learn everything at once in Offline Learning. However, it may also be time-consuming in Online Learning due to the trial and error approach in exploration.

**Note**: There is a difference between Online Learning and On-Policy (and Offline Learning with Off-Policy) where the former refers to the learning (policy can be changed or fixed) and the latter refers to the policy (does the series of trials come from one policy or multiple policies). On-Policy and Off-Policy will be explained using algorithms in the last two sections of this article.



Photo by [Crissy Jarvis](https://unsplash.com/@crissyjarvis?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

After understanding the different types of Reinforcement Learning, let’s dive into the algorithms!

**№1. Direct Utility Estimation**

Type: Model-free, Offline Learning

In Direct Utility Estimation, the agent executes a series of trials using the fixed policy, and the utility of a state is the expected total reward from that state onwards or expected **reward-to-go**.

* Take the example of a self-driving car, if the car has a total future reward of +100 when it starts on a grid (1, 1) in one trial. In the same trial, the car revisits that grid, and the total future reward is +300 from that point onwards. In another trial, the car starts from that grid and has a total future reward of +200. The expected reward-to-go from that grid will be the average reward-to-go in all trials and all visits to that grid, in this case (100 + 300 + 200) / 3.

**Pros**: Given infinitely many trials, the sample average of reward will converge to the true expected reward.

**Cons**: The expected reward-to-go is updated at the end of each trial, meaning that the agent learns nothing until the end of the trial, causing Direct Utility Estimation to converge very slowly.

**№2. Adaptive Dynamic Programming (ADP)**

Type: Model-based, Offline Learning

In Adaptive Dynamic Programming (ADP), the agent tries to learn the transition and reward functions through experience. The transition function is learned by counting the number of times it transitioned to the next state taking action from the current state, while the reward function is learned upon entering the state. Given the learned transition and reward function, we can now solve the MDP.

* Take the example of a self-driving car, given 10 trials of attempting to move forward in a given state, if the car ends up moving forward 8 times and moving left 2 times, we learn that the transition probabilities are T(current state, forward, front state) = 0.8 and T(current state, forward, left state) = 0.2.

**Pros**: Since the environment is fully observable, it is easy to learn the transition model simply by counting.

**Cons**: Performance is limited by the agent's ability to learn the transition model. This would result in the problem being intractable for large state spaces since it takes too many trials to learn the transition model, and there are too many equations and unknowns to solve in the MDP.

**№3. Temporal-Difference Learning (TD Learning)**

Type: Model-free, Offline Learning

In Temporal-Difference Learning, the agent learns the utility function and updates the function after every transition with a learning rate.

Fig 6: Utility function update equation (Equation 23.3) — Image by author

Fig 6: Utility function update equation (Equation 23.3) — Image by author

The term temporal difference refers to the difference in utilities between successive states and updates the utility function based on this error signal, scaled by a learning rate as shown in Fig 6. The learning rate can be a fixed parameter or decreasing function of increasing visits to a state, which helps in the convergence of the utility function.

Compared to Direct Utility Estimation which learns after every trial, TD Learning learns after every transition, making it more efficient.

Compared to ADP, TD Learning does not need to learn the transition and reward functions, making it more computationally efficient, but also takes longer to converge.

ADP and TD Learning are Offline RL algorithms, but there exists active ADP and active TD Learning that are part of Online RL algorithms!

**№4. Exploration**

Type: Model-based, Online Learning, Active ADP

Exploration is an **active ADP algorithm**. Similar to the passive ADP algorithm, the agent tries to learn the transition and reward functions through experience, but the active ADP algorithm will learn the outcome for all actions, not just the fixed policy.

There is an additional **exploration function** that determines how ‘curious’ is the agent to take an action outside of the existing policy. The exploration function should increase with utility and decrease with experience.

* For example, if the state has high utility, the exploration function tends to visit that state more often. Exploration function *increase with utility* due to increasing greed.
* For example, if the state is not visited before or visited enough times, the exploration function tends to choose actions outside of existing policy. Conversely, if the state is visited many times, the exploration function is not as curious. Exploration function *decrease with experience* due to decreasing curiosity.

**Pros**: Exploration policy results in rapid convergence toward zero policy loss (optimal policy).

**Cons**: Utility estimate does not converge as fast as policy estimate because the agent will not frequent the low-utility states and hence does not know the exact utilities of those states.

**№5. Q-Learning**

Type: Model-free, Online Learning, Active TD Learning, Off-Policy

Q-Learning is an **active TD Learning** algorithm. The update rule in Fig 6 remains unchanged, but now the utility of a state is represented as the utility of a state-action pair using a **Q-function** instead, hence the name Q-Learning. The difference in the update rule for passive vs. active TD Learning is shown in Fig 7 below.

This notation difference is due to Passive RL having a fixed policy, such that each state will only perform a fixed action and utility simply depends on the state. Whereas in Active RL, the policy will be updated and utility now depends on the state-action pair as each state may perform different actions following different policies.

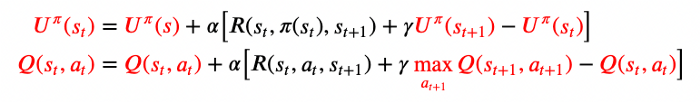


Fig 7: Utility function update equation for Passive TD (top) vs. Active TD (bottom, Equation 23.7) — Image by author

Q-Learning is **Off-Policy**, meaning that the target, or the utility of the next state, maximizes the Q-function over possible actions in the next state (regardless of current policy!). This way, we do not need the actual action in the next state.

**Pros**: Can be applied to complex domains as it is model-free and the agent does not need to learn or apply the transition model.

**Cons**: It does not look into the future and may have difficulty when rewards are sparse. It learns the policy at a slower rate compared to ADP as the local updates do not ensure consistency to Q-values.

**№6. SARSA**

Type: Model-free, Online Learning, Active TD Learning, On-Policy

SARSA is an **active TD Learning** algorithm. The algorithm name SARSA is derived from the components of the algorithm, namely state, action, reward, (next) state, and (next) action. This means that the SARSA algorithm waits for the next action to be taken in the next state before updating the Q-function. In contrast, Q-Learning is a ‘SARS’ algorithm since it does not consider the action in the next state.

Due to this difference, the SARSA algorithm knows the action taken in the next state and does not need to maximize the Q-function over all possible actions in the next state. The difference in the update rule for Q-Learning vs. SARSA is shown in Fig 8 below.

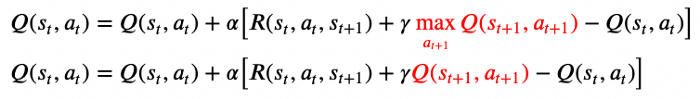


Fig 8: Utility function update equation for Q-Learning (top) vs. SARSA (bottom, Equation 23.8) — Image by author

SARSA is **On-Policy** as the target, or the utility of the next state uses Q-function from the current policy that is running. This way, the actual action in the next state is known.

**Note**: If Q-Learning does not explore other actions and follows the current policy in the next state, then it is identical to SARSA.

**Pros**: On-Policy is appropriate if the overall policy is controlled by another agent or program, such that the agent does not go Off-Policy and try other actions.

**Cons**: SARSA is less flexible than Q-Learning since it does not go Off-Policy to explore other policies. It learns the policy at a slower rate compared to ADP as the local updates do not ensure consistency to Q-values.

In summary, these are the 6 algorithms discussed, categorized into the different types of Reinforcement Learning.

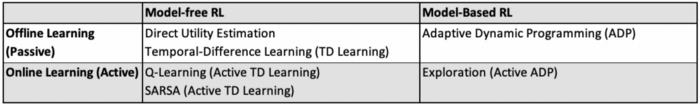


Fig 9: Summary of the 6 Reinforcement Learning Algorithms

These 6 algorithms are the basic algorithms that help form the base understanding of Reinforcement Learning. There are more effective Reinforcement Learning algorithms such as Deep Q Network (DQN), Deep Deterministic Policy Gradient (DDPG), and other algorithms that have more practical applications.

If you have read until the end, kudos to you! I have always found Reinforcement Learning fascinating since it formulates how humans learn and how we impart this knowledge to a robot (and of course in other applications such as self-driving cars, chess, and Alpha Go, to name a few). Hope you have understood more about Reinforcement Learning, the different types of Reinforcement Learning, and the algorithms that illustrate each type of Reinforcement Learning.