Advanced Machine Learning (Semester 1 2023)

Reinforcement Learning

Nina Hernitschek
Centro de Astronomía CITEVA
Universidad de Antofagasta

June 19, 2023

Recap

Machine learning methods we saw so far fall either into the category of **supervised** or **unsupervised** machine learning:

neural networks (NN, CNN, Autoencoder)

Random Forest

SVM

regression

...

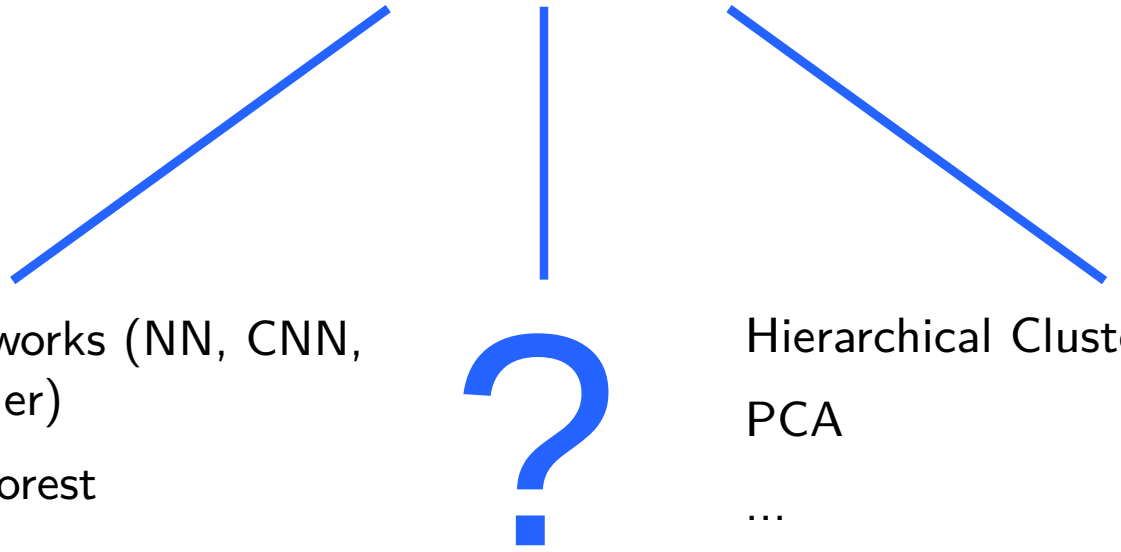
Hierarchical Clustering

PCA

...

Recap

Machine learning methods we saw so far fall either into the category of **supervised** or **unsupervised** machine learning:



Motivation

Consider the **scenario** of teaching new tricks to a cat

Motivation

Consider the **scenario** of teaching new tricks to a cat

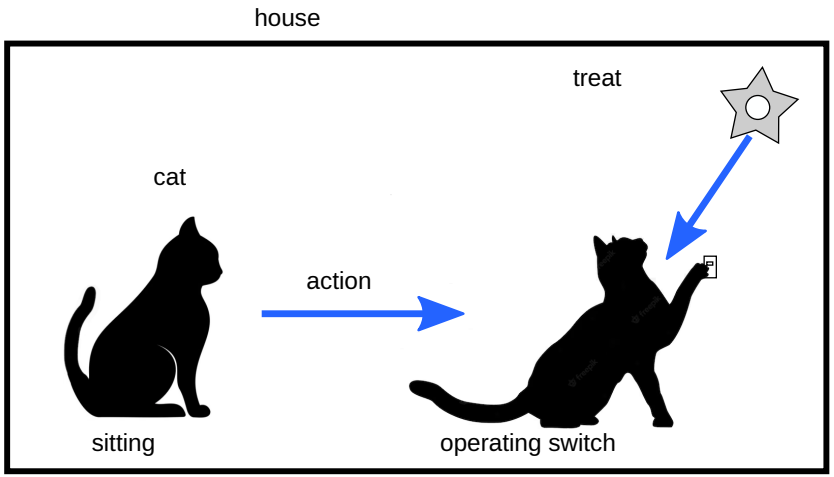
As the cat doesn't understand human language, we can't tell her directly what to do. Instead, we follow a different strategy.

Motivation

Consider the **scenario** of teaching new tricks to a cat

As the cat doesn't understand human language, we can't tell her directly what to do. Instead, we follow a different strategy.

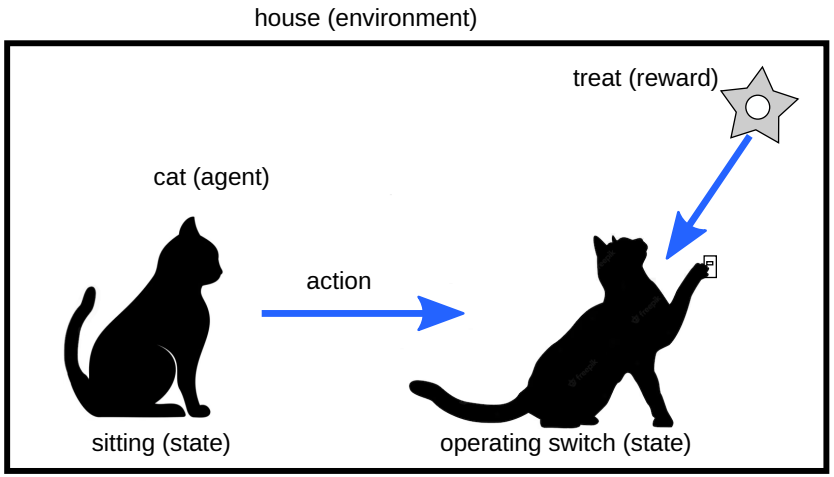
- We emulate a situation, and the cat responds in many different ways.
- If the cat responds in the desired way, we will give her a treat.
- Now whenever the cat is exposed to the same situation, the cat executes a similar action in expectation of getting more treats.
- In that way the cat **learns** what is **rewarded** and **repeats** this behavior.



Motivation

In this case,

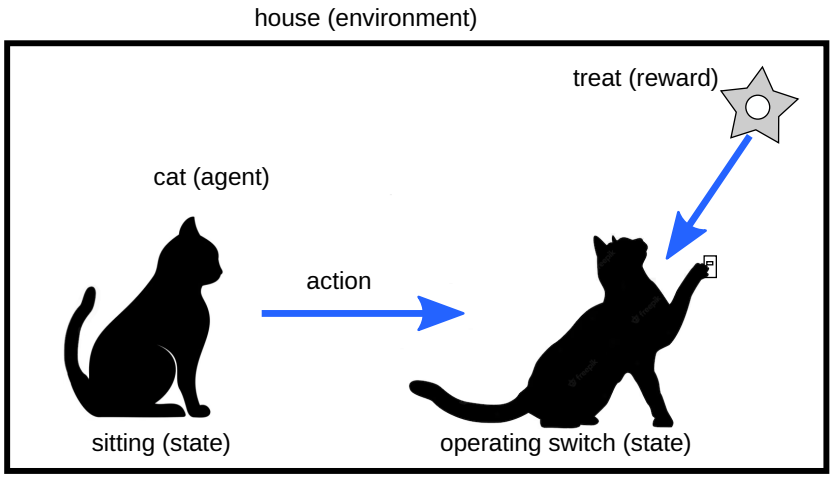
- The cat is an **agent** that is exposed to the **environment** (the house). An example of a **state** could be your cat sitting, and you use a specific word in for the cat to operate the switch.



Motivation

In this case,

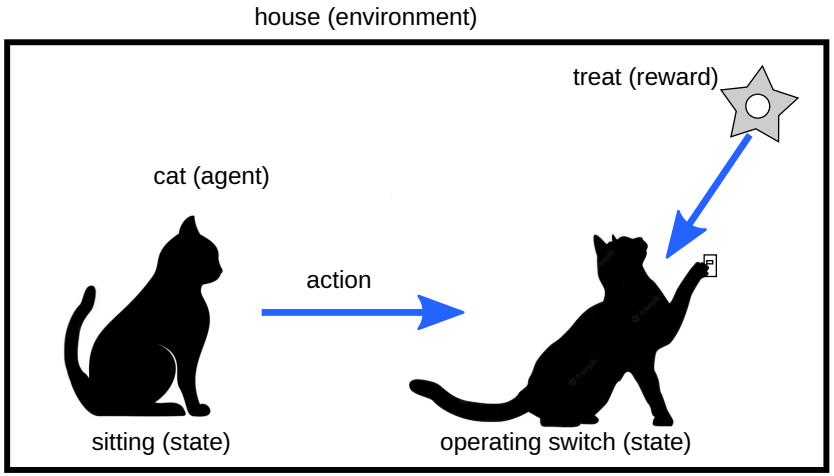
- The cat is an **agent** that is exposed to the **environment** (the house). An example of a **state** could be your cat sitting, and you use a specific word in for the cat to operate the switch.
- The agent reacts by **performing an action** to transition between states. For example, the cat goes from sitting to touching the switch.



Motivation

In this case,

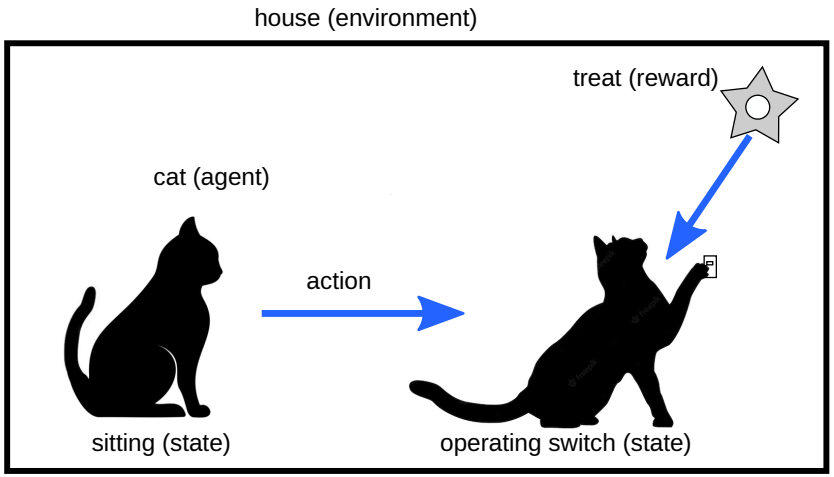
- The cat is an **agent** that is exposed to the **environment** (the house). An example of a **state** could be your cat sitting, and you use a specific word in for the cat to operate the switch.
- The agent reacts by **performing an action** to transition between states. For example, the cat goes from sitting to touching the switch.
- When performing an action, the **policy** is a method of selecting an action given a state in expectation of better outcomes.



Motivation

In this case,

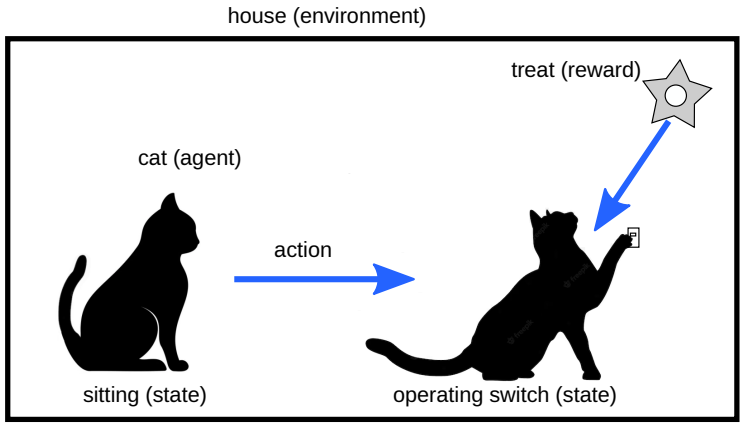
- The cat is an **agent** that is exposed to the **environment** (the house). An example of a **state** could be your cat sitting, and you use a specific word in for the cat to operate the switch.
- The agent reacts by **performing an action** to transition between states. For example, the cat goes from sitting to touching the switch.
- When performing an action, the **policy** is a method of selecting an action given a state in expectation of better outcomes.
- After the transition, they may get a **reward or penalty** in return.



Motivation

In this case,

- The cat is an **agent** that is exposed to the **environment** (the house). An example of a **state** could be your cat sitting, and you use a specific word in for the cat to operate the switch.
- The agent reacts by **performing an action** to transition between states. For example, the cat goes from sitting to touching the switch.
- When performing an action, the **policy** is a method of selecting an action given a state in expectation of better outcomes.
- After the transition, they may get a **reward or penalty** in return.



in machine learning, this is called **reinforcement learning**

Motivation

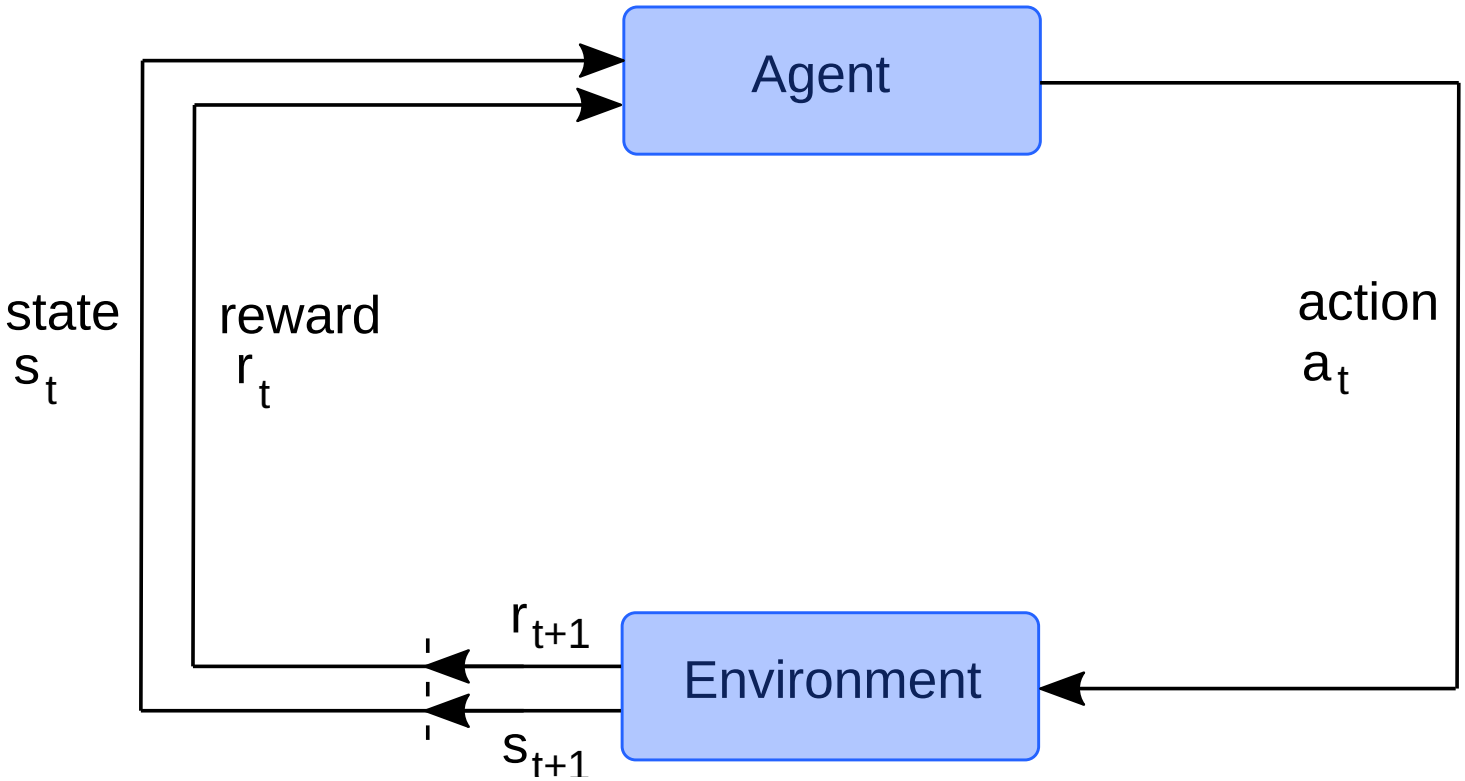
Reinforcement Learning is the closest to human and animal learning in a dynamic environment.

Simplified Definition of Reinforcement Learning

Reinforcement learning is where the learner or the decision maker, called the **agent**, interacts continually with its **environment** by performing actions sequentially at each **discrete time step**. Interaction of the agent with its environment changes the environment's **state**, and as a result, the agent receives a numerical **reward** from the environment.

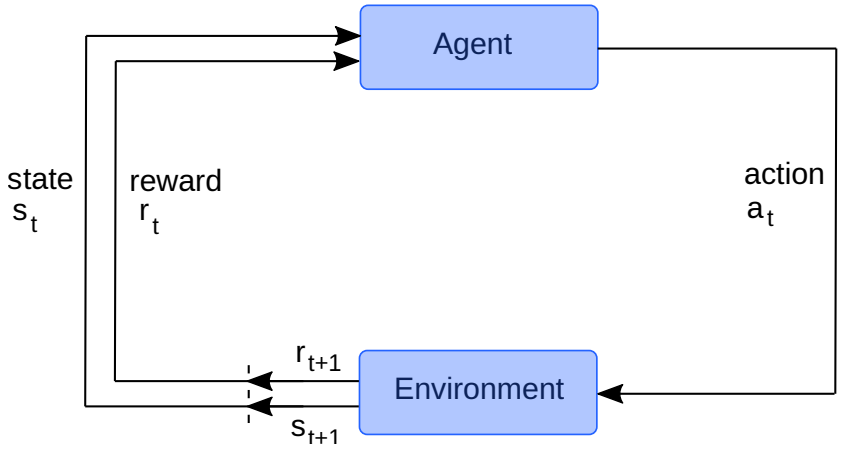
Reinforcement Learning Definitions

The Reinforcement Learning Cycle



Reinforcement Learning Definitions

The elements of Reinforcement Learning are defined as follows (1):

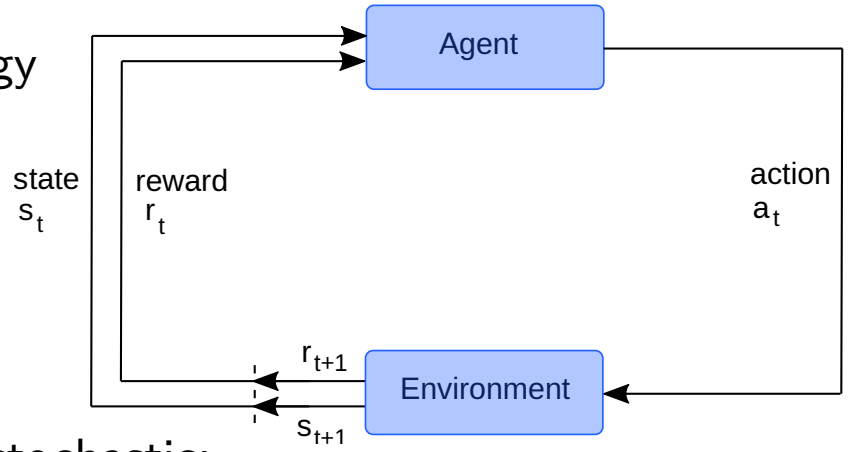


- Agent: An agent takes actions; e.g. a robot navigating a terrain, or a hero navigating a video game.
- State (S): A state is a situation in which the agent finds itself; i.e. an instantaneous configuration that puts the agent in relation to other significant things such as tools, obstacles or prizes.
- Environment, Model: The environment takes the agent's current state and action as input, and outputs the agent's reward and its next state.
- Action (A): A is the set of all possible moves the agent can make. Agents usually choose from a list of discrete, possible actions. In video games, the list might include running right or left, jumping high or low, picking up an item. In the stock markets, the list might include buying, selling or holding.

Reinforcement Learning Definitions

The elements of Reinforcement Learning are defined as follows (2):

- Policy (π): The policy is the strategy the agent employs to determine the next action based on the current state, thus mappings the perceived states of the environment to the actions taken on those states.



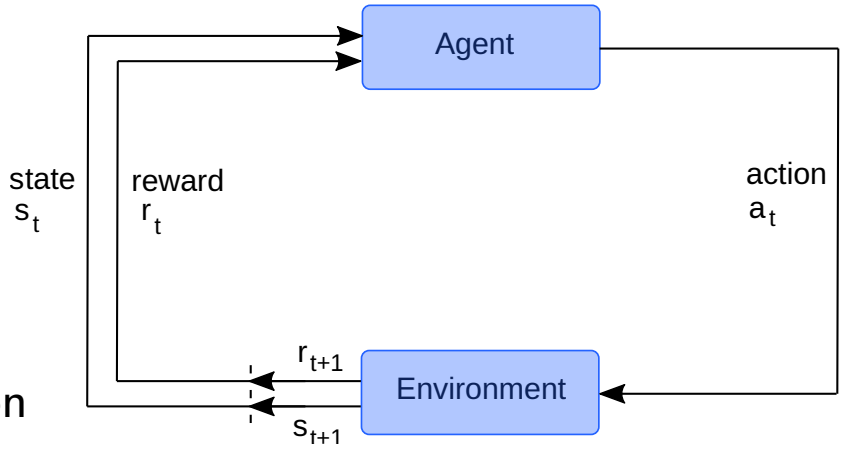
  The policy can be deterministic or stochastic:
  deterministic policy: $a = \pi(s)$, stochastic policy: $\pi(a|s) = P(a_t = a|a_t = s)$
- Reward (R): A reward is the immediate or delayed feedback by the environment (model) which measures the success of an agent's actions in a given state. The agent's main objective is to maximize the total number of rewards. E.g.: in a video game, a hero finding a coin wins points.
- Value Function (V): The value function gives information about agent's expected reward. The goal of estimating values is to achieve more rewards.

Reinforcement Learning Definitions

The elements of Reinforcement Learning are defined as follows (3):

optional:

- Discount factor (γ): A factor that gets multiplied with future rewards, designed to dampen future rewards' effect on the agent's choice of action (making future rewards worth less than immediate rewards).



Reinforcement Learning Definitions

Unlike other forms of machine learning - such as supervised and unsupervised learning - reinforcement learning can only be thought about **sequentially** in terms of state-action pairs that occur one after the other.

Reinforcement is **goal-oriented**, and its aim is to learn sequences of actions by trial-and-error that will lead an agent to achieve its goal, or maximize its objective function.

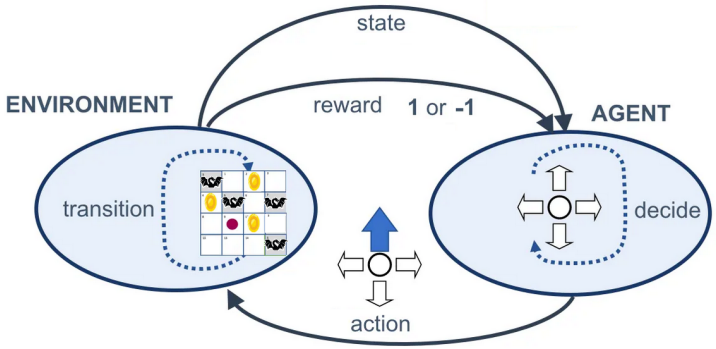
Reinforcement Learning Definitions

Unlike other forms of machine learning - such as supervised and unsupervised learning - reinforcement learning can only be thought about **sequentially** in terms of state-action pairs that occur one after the other.

Reinforcement is **goal-oriented**, and its aim is to learn sequences of actions by trial-and-error that will lead an agent to achieve its goal, or maximize its objective function.

examples:

- In computer games, the goal is to finish the game with the most points, so each additional point obtained will affect the agent's subsequent behavior; e.g. the agent may learn to collect coins and avoid dragons to maximize its score.



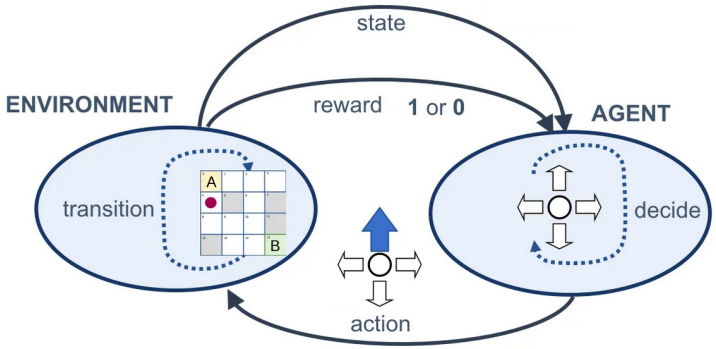
Reinforcement Learning Definitions

Unlike other forms of machine learning - such as supervised and unsupervised learning - reinforcement learning can only be thought about **sequentially** in terms of state-action pairs that occur one after the other.

Reinforcement is **goal-oriented**, and its aim is to learn sequences of actions by trial-and-error that will lead an agent to achieve its goal, or maximize its objective function.

examples:

- In the real world, the goal might be for a robot to travel from point A to point B. Every length unit the robot moves closer to point B is rewarded, whereas every length unit the robot gets further away from B as well as every time the robot gets stuck and has to free itself is penalized.



Reinforcement Learning Definitions

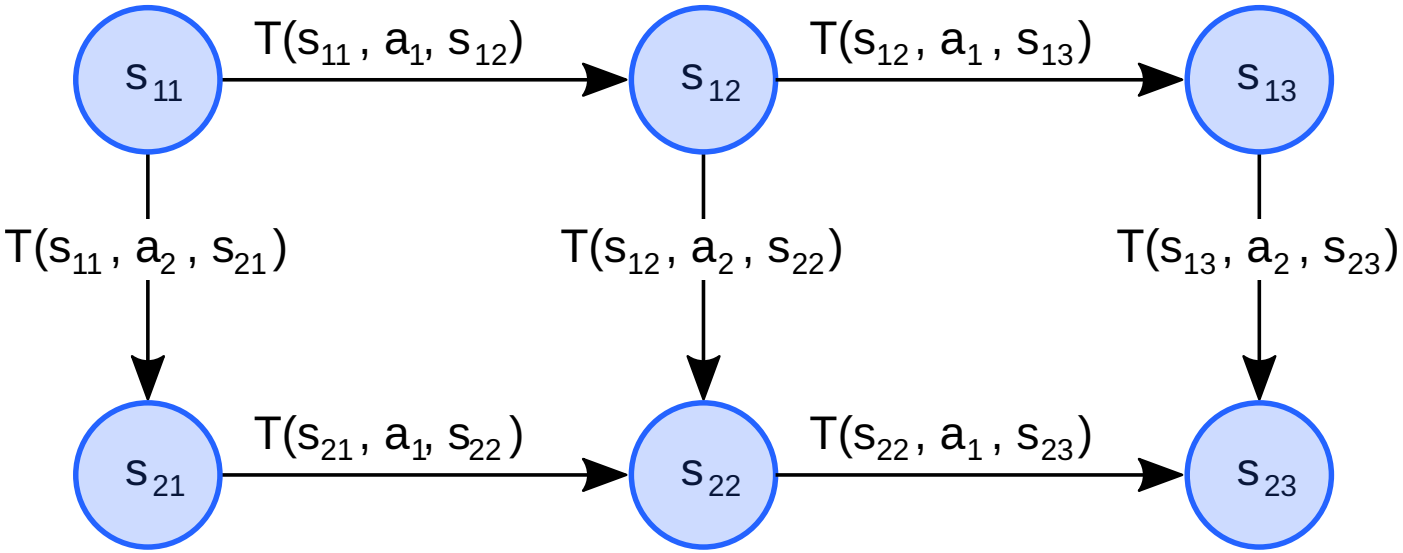
the steps for setting up a Reinforcement Learning algorithm:

- Create the Environment
- Define the Reward
- Create the Agent
- Train and validate the Agent
- Deploy the Policy

Mathematical Description

How can we mathematically formalize the Reinforcement Learning problem?

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



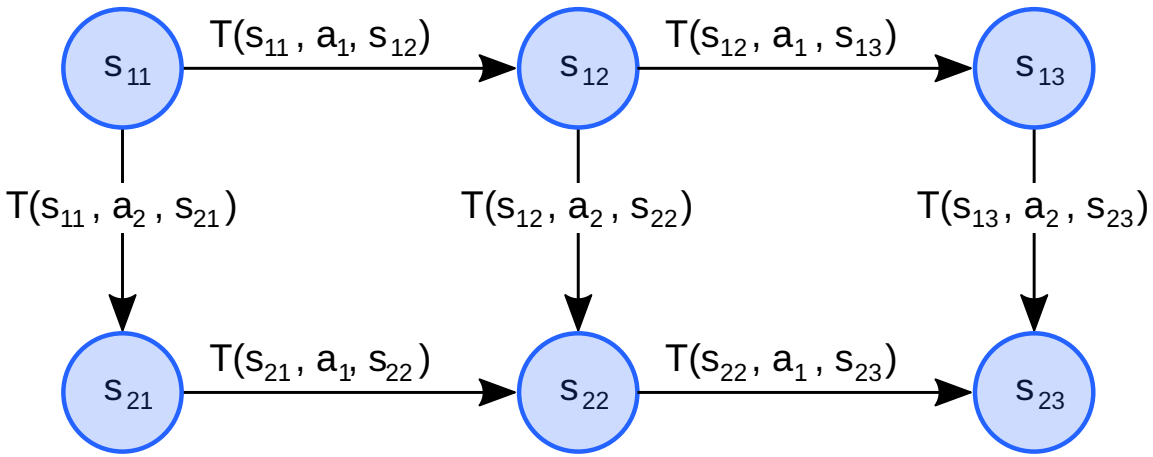
Mathematical Description

Markov Decision Process (MDP):

We can represent the agent state using the Markov state that contains all the required information from the history. The state s_t is a **Markov state** if it follows the Markov property which says that the future is independent of the past and can only be defined with the present (a **memoryless** process):

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, ..., s_t).$$

Reinforcement Learning works on fully observable environments, where the agent can observe the environment and act for the new state.

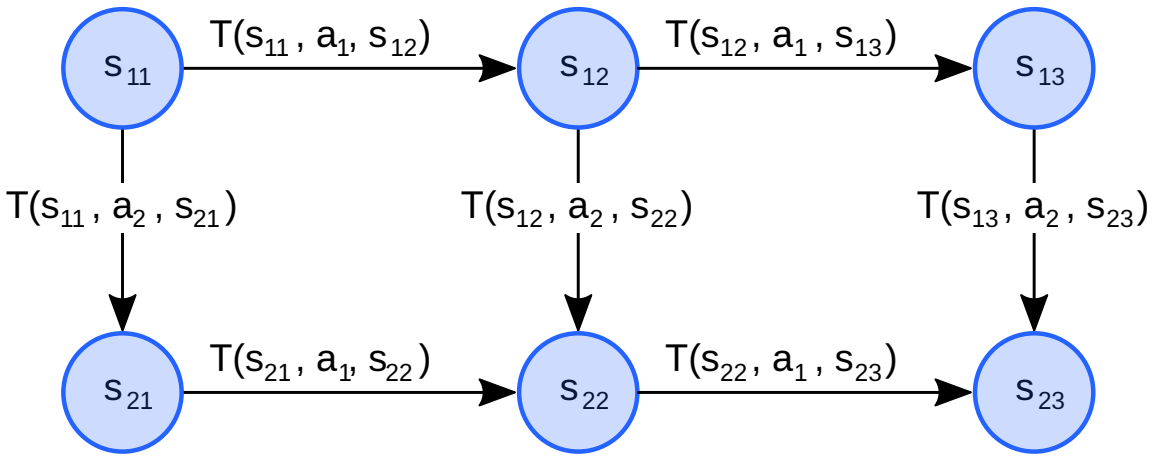


Mathematical Description

Markov Decision Process (MDP):

The states (denoted by $s \in S$) are represented by circles, and actions (denoted by $a \in A$) allow the agent to transition between states. With each state, a reward is associated.

A transition probability T gives the probability of transiting from one state to another state given an action. E.g.: $T(s_{11}, a_1, s_{12})$ is the probability of transitioning to state s_{12} after taking action a_1 at state s_{11}.



Mathematical Description

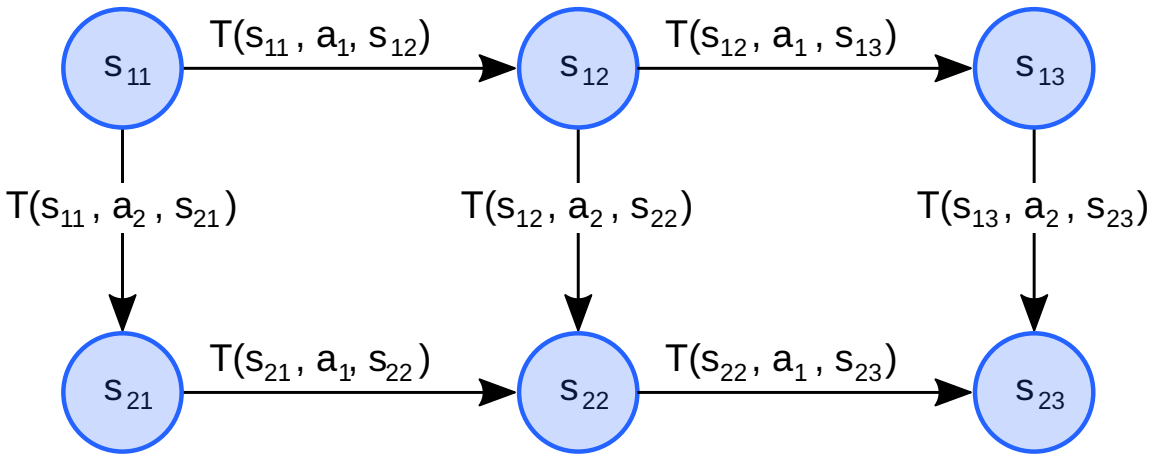
Markov Decision Process (MDP):

S: set of states
A: set of actions
γ: discount factor
T: $S \times A \times S \rightarrow [0, 1]$: transition probability distribution $T(s'|s, a)$

Deterministic policy: $\pi : S \rightarrow A$ mapping from states to actions

Stochastic policy: $\pi : S \times A \rightarrow [0, 1]$ distribution over actions for each state

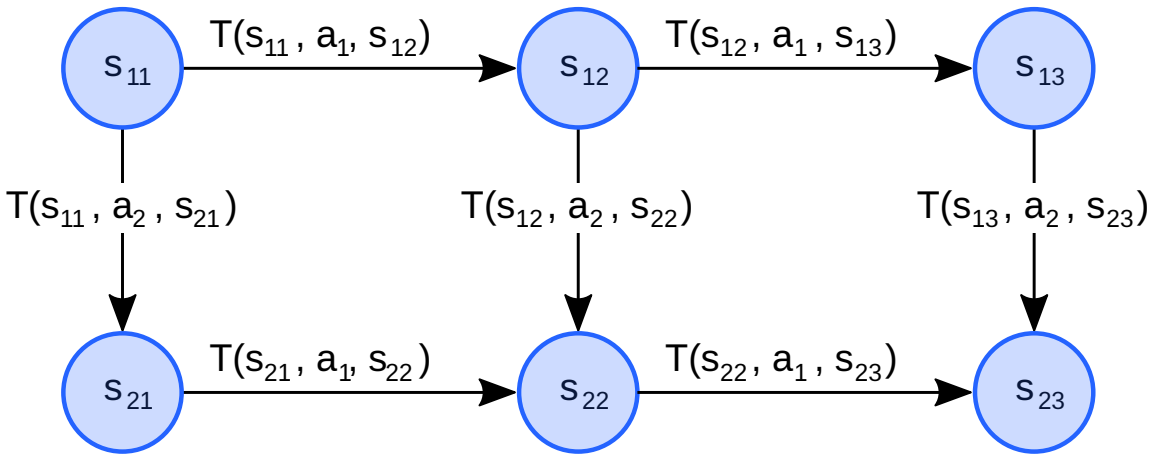
Value of a policy π, expected discount reward if starting in some state and following policy π: $V^{\pi}(s) = R(s) + \gamma \sum_{s \in S} T(s'|s, \pi(s)) V^{\pi}(s')$



Mathematical Description

Markov Decision Process (MDP):

example: Let the goal be to go from state s_{11} to s_{23}. The rewards are: 1 for states s_{11}, s_{21}, s_{22}; -1 for states s_{12}, s_{13}; and s_{23} is the goal state (reward 100).
We want the agent to learn the optimal route. A discount factor γ on the rewards favors earlier rewards over later ones. We assume $T = 1$.

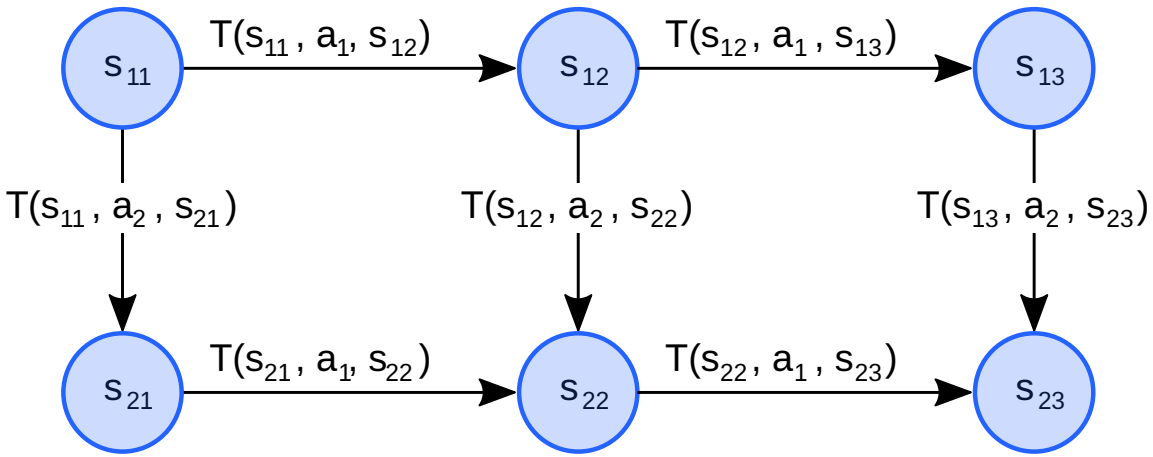


Mathematical Description

Markov Decision Process (MDP):

example: Let the goal be to go from state s_{11} to s_{23}. The rewards are: 1 for states s_{11}, s_{21}, s_{22}; -1 for states s_{12}, s_{13}; and s_{23} is the goal state (reward 100).
We want the agent to learn the optimal route. A discount factor γ on the rewards favors earlier rewards over later ones. We assume $T = 1$.
The **expected utility** for actions $a_2 \rightarrow a_1 \rightarrow a_1$ starting in state s_{11} is:

$$\begin{aligned} U &= R(s_{11}) + \gamma R(s_{21}) + \gamma^2 R(s_{22}) + \gamma^3 R(s_{23}) \\ &= 1 + \gamma(1) + \gamma^2(1) + \gamma^3(100) \end{aligned}$$



Mathematical Description

It follows that a state's utility is related to its neighbors' 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:

$$U(s_t) = \max_{a \in A(s_t)} \sum_{s_{t+1}} T(s_t, a, s_{t+1}) \left[R(s_t, a, s_{t+1}) + \gamma U(s_{t+1}) \right]$$

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.

Mathematical Description

The example is a simplified illustration:

- The transition probability may not be 1, as uncertainties in actions can be factored in. Therefore, we need to take an expected value V over this uncertainty.
- The optimal action may not be known yet (realistically), therefore the generic representation would be to represent an action as a policy from the state, $\pi(S)$.
- The reward may not be based on the state, it could be based on a combination of the previous state, action, and next state, denoted by $R(s_1, \pi(s_1), s_2)$.
- The problem may not be solved within a given number of steps, it could take an infinite amount of steps to reach the goal state.

Mathematical Description

The example is a simplified illustration:

- The transition probability may not be 1, as uncertainties in actions can be factored in. Therefore, we need to take an expected value V over this uncertainty.
- The optimal action may not be known yet (realistically), therefore the generic representation would be to represent an action as a policy from the state, $\pi(S)$.
- The reward may not be based on the state, it could be based on a combination of the previous state, action, and next state, denoted by $R(s_1, \pi(s_1), s_2)$.
- The problem may not be solved within a given number of 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:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1})\right]$$

State-Action Pairs & Complex Probability Distributions of Reward

The **goal** of reinforcement learning is to pick the best known action for any given state, which means the actions have to be ranked, and assigned values relative to one another. Since those actions are state-dependent, what we are really gauging is the value of state-action pairs; i.e. an action taken from a certain state, something you did somewhere.

A simple to demonstrate that the value and meaning of an action is contingent upon the state in which it is taken:

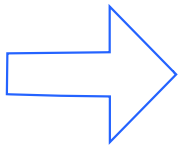
If the action is "driving to the right", it would mean something different whether there is an obstacle or not.

State-Action Pairs & Complex Probability Distributions of Reward

The **goal** of reinforcement learning is to pick the best known action for any given state, which means the actions have to be ranked, and assigned values relative to one another. Since those actions are state-dependent, what we are really gauging is the value of state-action pairs; i.e. an action taken from a certain state, something you did somewhere.

A simple to demonstrate that the value and meaning of an action is contingent upon the state in which it is taken:

If the action is "driving to the right", it would mean something different whether there is an obstacle or not.



We can't predict an action's outcome without knowing the context.

Mathematical Description

Going back to the Markov decision process: How to **find** that policy?

Mathematical Description

Going back to the Markov decision process: How to **find** that policy?

The optimal policy is derived by making use of the Bellman equation:
The Bellman equation can be used to solve for the optimal policy using Policy iteration or Value Iteration, which is an iterative method to pass the utility values from a future state to the current state.

Optimal value function (of optimal policy π^*):

$$V^*(s) = R(s) + \gamma \max_{a \in A} \sum_{s \in S} P(s'|s,a) V^*(s')$$

Mathematical Description

Going back to the Markov decision process: How to **find** that policy?

The optimal policy is derived by making use of the Bellman equation:
The Bellman equation can be used to solve for the optimal policy using Policy iteration or Value Iteration, which is an iterative method to pass the utility values from a future state to the current state.

Optimal value function (of optimal policy π^*):

$$V^*(s) = R(s) + \gamma \max_{a \in A} \sum_{s \in S} P(s'|s,a) V^*(s')$$

Policy Iteration:

$$\forall s \in S : \widehat{V}^{\pi}(s) := R(s) + \gamma \sum_{s \in S} P(s'|s,\pi(s)) \hat{V}^{\pi}(s')$$

Mathematical Description

Going back to the Markov decision process: How to **find** that policy?

The optimal policy is derived by making use of the Bellman equation: The Bellman equation can be used to solve for the optimal policy using Policy iteration or Value Iteration, which is an iterative method to pass the utility values from a future state to the current state.

Optimal value function (of optimal policy π^*):

$$V^*(s) = R(s) + \gamma \max_{a \in A} \sum_{s \in S} P(s'|s,a) V^*(s')$$

Policy Iteration:

$$\forall s \in S : \widehat{V}^{\pi}(s) := R(s) + \gamma \sum_{s \in S} P(s'|s,\pi(s)) \hat{V}^{\pi}(s')$$

Value Iteration: To find the optimal value function, start with any $\widehat{V}^*(s)$ and repeat:

$$\forall s \in S : \widehat{V}^*(s) := R(s) + \gamma \max_{a \in A} \sum_{s \in S} P(s'|s,a) \hat{V}^*(s')$$

Mathematical Description

We want to find the **optimal policy** π^* that maximizes the sum of rewards.

Mathematical Description

We want to find the **optimal policy** π^* that maximizes the sum of rewards.

How to handle the randomness (initial state, transition probability)?

Mathematical Description

We want to find the **optimal policy** π^* that maximizes the sum of rewards.

How to handle the randomness (initial state, transition probability)?

Maximize the **expected sum of rewards**:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi\right]$$

with $s_0 \sim p(s_0)$, $a_t \sim \pi(\cdot|s_t)$, $s_{t+1} \sim p(\cdot|s_t, a_t)$

Mathematical Description

Value function and Q-value function

Following a policy produces sample trajectories (sometimes also called *paths*): $s_0, s_0, r_0, s_1, a_1, r_1, ...$

How good is a state?

Value function and Q-value function

Following a policy produces sample trajectories (sometimes also called *paths*): $s_0, s_0, r_0, s_1, a_1, r_1, ...$

How good is a state?

The **value function** at state s is the expected cumulative reward from following the policy π from state s:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

Mathematical Description

Value function and Q-value function

How good is a state-action pair?

Value function and Q-value function

How good is a state-action pair?

The **Q-value function** at state s and action a is the expected cumulative reward from taking action a in state s and then following the policy π:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^{t} r_t | s_0 = s, a_0 = a, \pi\right]$$

Value function and Q-value function

How good is a state-action pair?

The **Q-value function** at state s and action a is the expected cumulative reward from taking action a in state s and then following the policy π:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

The optimal Q-value function Q^* is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^{*}(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Value function and Q-value function

The optimal Q-value function Q^* satisfies the following Bellman equation:

$$Q^*(s,a) = E_{s' \sim \epsilon}\left[r + \gamma \max_{a'} Q^*(s',a') | s, a\right]$$

intuition: If the optimal state-action pair values for the next time step $Q^*(s',a')$ are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s',a')$.

The optimal policy π^* corresponds to taking the best action in any state as specified by Q^*.

Approaches to Implement Reinforcement Learning

There are mainly two **types** of reinforcement learning:

Positive Reinforcement Learning
gives a positive impact on the action, which is taken by the agent, and it increases the behavior's

- strength
- frequency

This type of reinforcement can sustain the changes for a long time, but too much positive reinforcement may lead to an overload of states that can reduce the consequences.

Approaches to Implement Reinforcement Learning

There are mainly two **types** of reinforcement learning:

Positive Reinforcement Learning gives a positive impact on the action, which is taken by the agent, and it increases the behavior's

- strength
- frequency

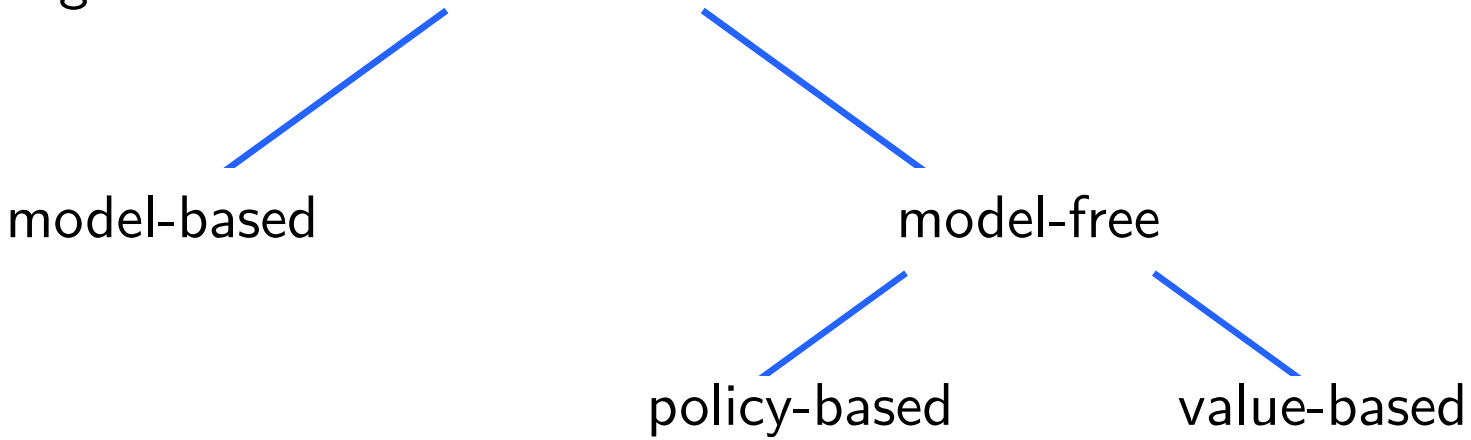
This type of reinforcement can sustain the changes for a long time, but too much positive reinforcement may lead to an overload of states that can reduce the consequences.

Negative Reinforcement Learning increases the tendency that the specific behavior will occur again by avoiding the negative condition.

Depending on situation and behavior, it can be more effective than positive reinforcement, but it provides reinforcement only to meet minimum behavior.

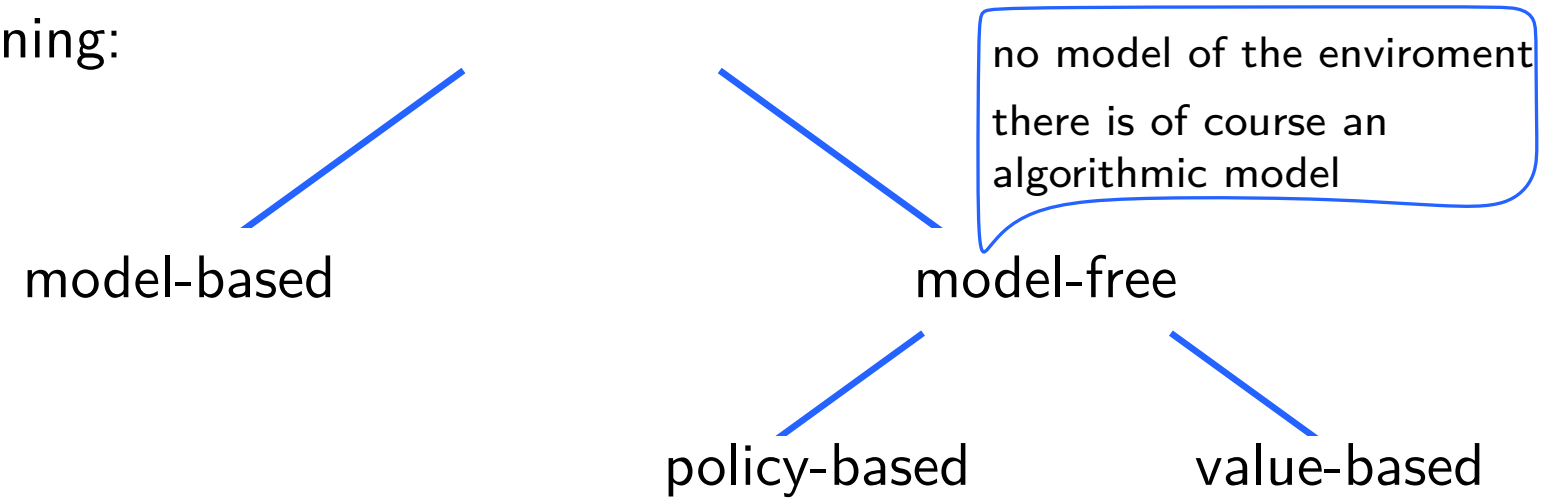
Approaches to Implement Reinforcement Learning

There are mainly the following **approaches** of reinforcement learning:



Approaches to Implement Reinforcement Learning

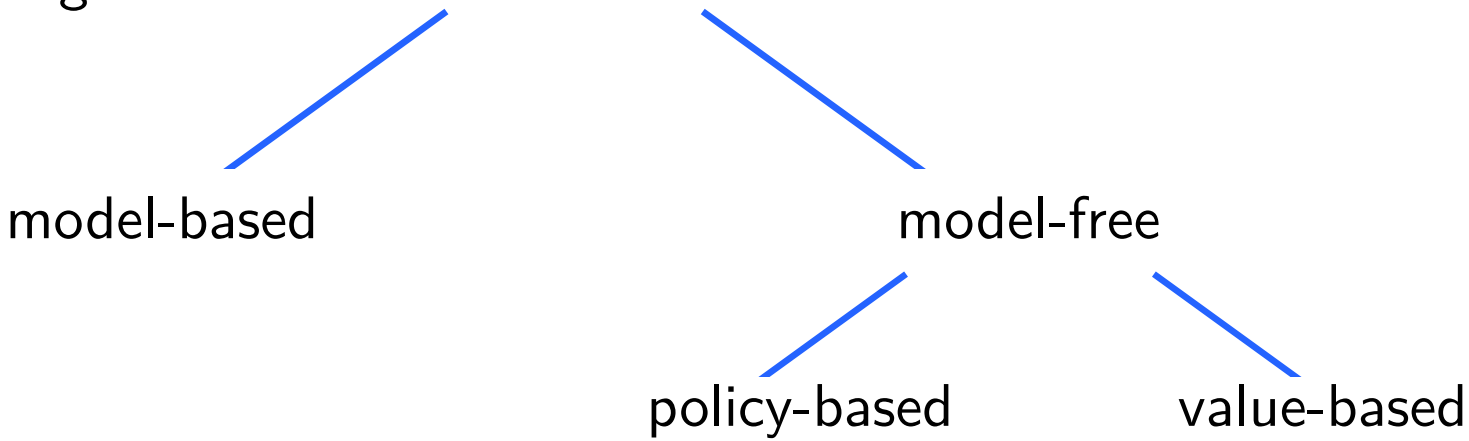
There are mainly the following **approaches** of reinforcement learning:



model-based vs. model-free:
A common problem in Reinforcement Learning is that when being in state s and make an action a, the new state s' is not necessarily known.
In the model-based approach you either have access to the model so the probability distribution over states is known. This allows the agent for planning moves without actually performing any actions.
In the model-free approach no model is given, so the agent isn't trying to explicitly figure out how it works. The agent only collect some experience to derive the optimal policy.

Approaches to Implement Reinforcement Learning

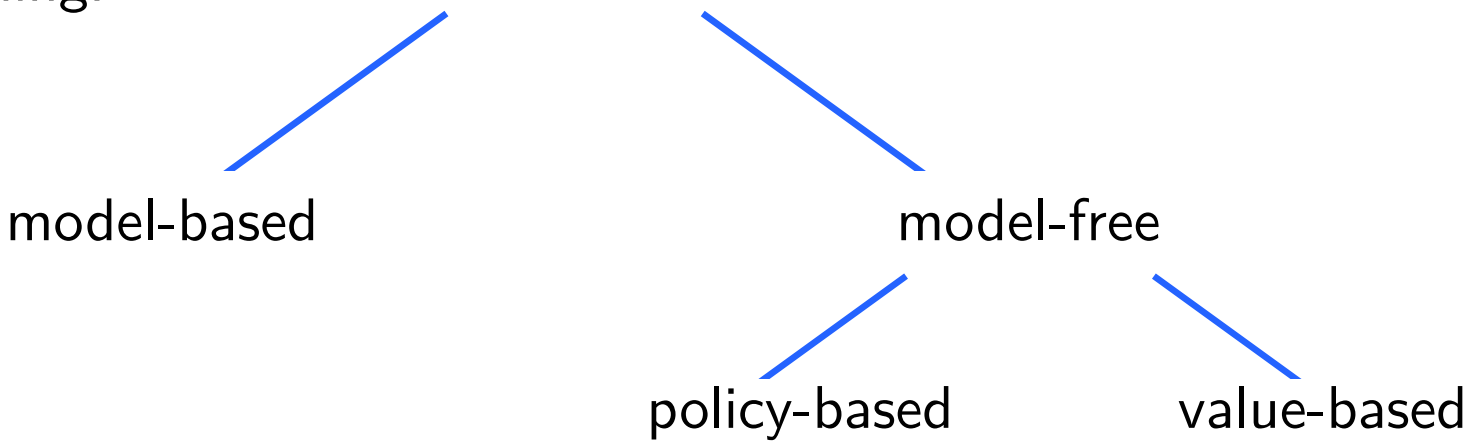
There are mainly the following **approaches** of reinforcement learning:



policy-based vs. value-based:
In the policy-based approach a representation of a policy is built and kept in memory during learning. Goal: find the optimal policy to maximize future rewards without the value function. Two policy-based approaches:

Approaches to Implement Reinforcement Learning

There are mainly the following **approaches** of reinforcement learning:

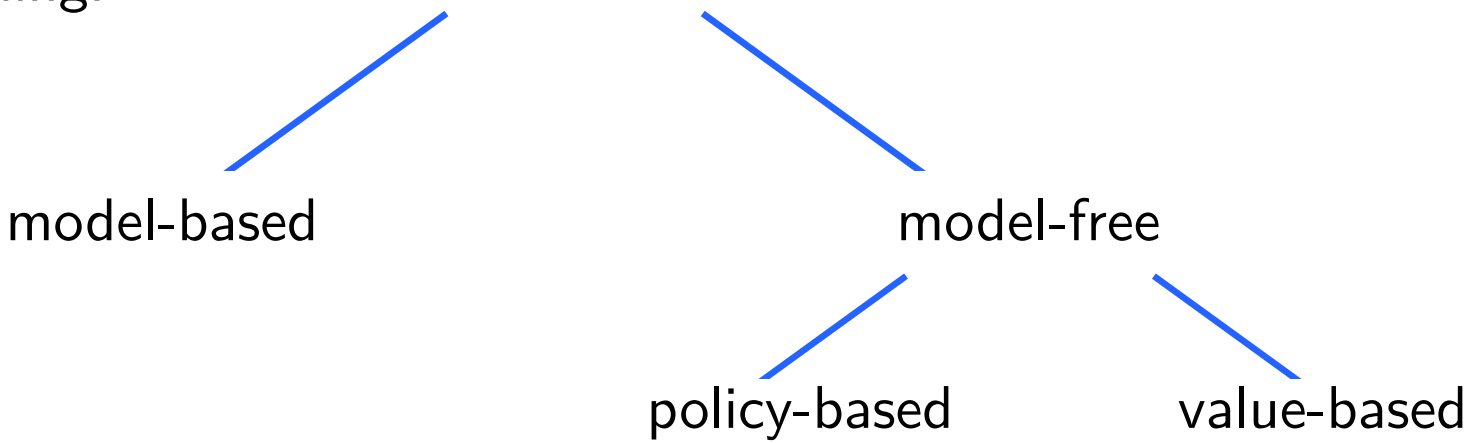


policy-based vs. value-based:
In the policy-based approach a representation of a policy is built and kept in memory during learning. Goal: find the optimal policy to maximize future rewards without the value function. Two policy-based approaches:

- A **deterministic** policy maps states to actions, such that it describes which action to take in each state: $\pi : S \rightarrow A$.

Approaches to Implement Reinforcement Learning

There are mainly the following **approaches** of reinforcement learning:

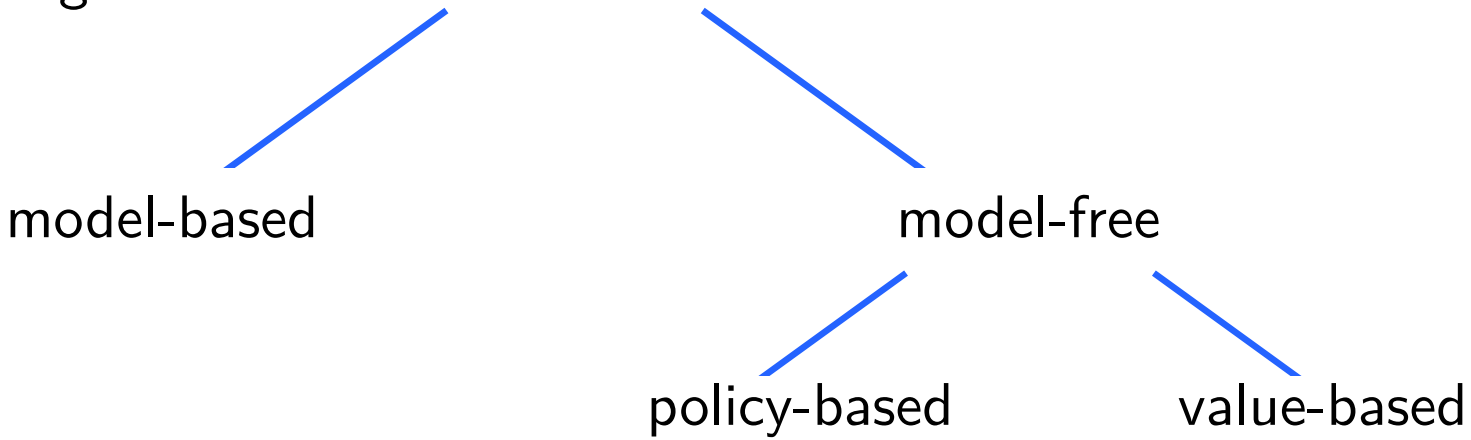


policy-based vs. value-based:
In the policy-based approach a representation of a policy is built and kept in memory during learning. Goal: find the optimal policy to maximize future rewards without the value function. Two policy-based approaches:

- A **deterministic** policy maps states to actions, such that it describes which action to take in each state: $\pi : S \rightarrow A$.
- A **stochastic** policy describes the probability of each action in each state: $\pi : A \times S \rightarrow [0, 1]$ where $\pi(a|s)$ is the probability of taking action a in state s.

Approaches to Implement Reinforcement Learning

There are mainly the following **approaches** of reinforcement learning:



policy-based vs. value-based:
In the value-based approach we don't store any explicit policy, only a value function V. The policy is here implicit and can be derived directly from the value function (pick the action with the best value).
The main goal of this method is to maximize a value function $V(s)$ at a state under any policy. In this approaches, the agent is expecting a long-term return of the current states under policy π.

Approaches to Implement Reinforcement Learning

Offline Learning vs. Online Learning

Offline (Passive) Learning:
Given a fixed policy with unknown transition and reward functions, the agent tries to learn the utility function by executing a series of trials.
example: A self-driving rover is equipped with a map and a general direction to follow (fixed policy), slightly faulty controls (moving forward could turn the rover slightly to the left or right - unknown transition probability) and unknown travelling time (unknown reward function).

Approaches to Implement Reinforcement Learning

Offline Learning vs. Online Learning

Offline (Passive) Learning:
Given a fixed policy with unknown transition and reward functions, the agent tries to learn the utility function by executing a series of trials.
example: A self-driving rover is equipped with a map and a general direction to follow (fixed policy), slightly faulty controls (moving forward could turn the rover slightly to the left or right - unknown transition probability) and unknown travelling time (unknown reward function).

Online (Active) Learning:
The problem is solved by learning to plan or decide. 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.

Approaches to Implement Reinforcement Learning

How to **choose** an approach?

Approaches to Implement Reinforcement Learning

How to **choose** an approach?

The example seen before is model-based Reinforcement Learning. Model-based methods are useful for simulations.

as the Environment is known per definition

Approaches to Implement Reinforcement Learning

How to **choose** an approach?

The example seen before is model-based Reinforcement Learning. Model-based methods are useful for simulations.

as the Environment is known per definition

Model-free methods are useful for solving real-life problems.

as the Environment is not fully known

Approaches to Implement Reinforcement Learning

How to **choose** an approach?

The example seen before is model-based Reinforcement Learning. Model-based methods are useful for simulations.

as the Environment is known per definition

Model-free methods are useful for solving real-life problems.

as the Environment is not fully known

Generally: if no accurate model is provided as part of the problem definition, then model-free approaches are often superior.

Agents that build **predictive models** of the environment: interesting approach, however, model-based agents learning their own models have a problem that inaccuracy in these models can cause instability.

Reinforcement Learning Algorithms

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.

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.

example: The self-driving rover has a total future reward of +100 when it starts on a grid (0, 0) in one trial. In the same trial, the rover revisits that grid, and the total future reward is +300 from that point onwards. In another trial, it starts again from (0, 0) and has a total future reward of +150. The expected reward-to-go from that grid will be the average from all trials and all visits to that grid, in this case (100 + 300 + 150) / 3.

Reinforcement Learning Algorithms

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.

example: The self-driving rover has a total future reward of +100 when it starts on a grid (0, 0) in one trial. In the same trial, the rover revisits that grid, and the total future reward is +300 from that point onwards. In another trial, it starts again from (0, 0) and has a total future reward of +150. The expected reward-to-go from that grid will be the average from all trials and all visits to that grid, in this case (100 + 300 + 150) / 3.

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

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.

example: The self-driving rover has a total future reward of +100 when it starts on a grid (0, 0) in one trial. In the same trial, the rover revisits that grid, and the total future reward is +300 from that point onwards. In another trial, it starts again from (0, 0) and has a total future reward of +150. The expected reward-to-go from that grid will be the average from all trials and all visits to that grid, in this case (100 + 300 + 150) / 3.

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

⊖ 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.

Reinforcement Learning Algorithms

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.

Reinforcement Learning Algorithms

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.

example: The self-driving rover carries out 10 trials of attempting to move forward in a given state. If it 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) $= 0.2$.

Reinforcement Learning Algorithms

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.

example: The self-driving rover carries out 10 trials of attempting to move forward in a given state. If it 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) $= 0.2$.

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

Reinforcement Learning Algorithms

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.

example: The self-driving rover carries out 10 trials of attempting to move forward in a given state. If it 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) $= 0.2$.

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

⊖ Performance is limited by the agent's ability to learn the transition model. For large state spaces it takes too many trials to learn the transition model.

Reinforcement Learning Algorithms

Exploration (type: Model-based, Online Learning, Active ADP)

Exploration is an active ADP algorithm. Similar to passive ADP, 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. An exploration function determines how 'curious' the agent is to take an action outside of the existing policy. The exploration function should increase with utility and decrease with experience.

Reinforcement Learning Algorithms

Exploration (type: Model-based, Online Learning, Active ADP)

Exploration is an active ADP algorithm. Similar to passive ADP, 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. An exploration function determines how 'curious' the agent is 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. If the state was not 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.

Reinforcement Learning Algorithms

Exploration (type: Model-based, Online Learning, Active ADP)

Exploration is an active ADP algorithm. Similar to passive ADP, 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. An exploration function determines how 'curious' the agent is 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. If the state was not 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 policy results in rapid convergence toward zero policy loss (optimal policy).

Reinforcement Learning Algorithms

Exploration (type: Model-based, Online Learning, Active ADP)

Exploration is an active ADP algorithm. Similar to passive ADP, 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. An exploration function determines how 'curious' the agent is 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. If the state was not 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 policy results in rapid convergence toward zero policy loss (optimal policy).

⊖ 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.

Reinforcement Learning Algorithms

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.

Reinforcement Learning Algorithms

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.

The term refers to the difference in utilities between successive states and updates the utility function based on this, scaled by a learning rate α:

$$U^{\pi}(s_t) = U^{\pi}(s) + \alpha\left[R(s_t, \pi(s_t), s_{t+1}) + \gamma U^{\pi}(s_{t+1}) - U^{\pi}(s_t)\right]$$

The learning rate can be a fixed parameter or a function indirect proportional to the number of visits to a state, which helps in the convergence of the utility function.

Reinforcement Learning Algorithms

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.

The term refers to the difference in utilities between successive states and updates the utility function based on this, scaled by a learning rate α:

$$U^{\pi}(s_t) = U^{\pi}(s) + \alpha\left[R(s_t, \pi(s_t), s_{t+1}) + \gamma U^{\pi}(s_{t+1}) - U^{\pi}(s_t)\right]$$

The learning rate can be a fixed parameter or a function indirect proportional to the number of 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, increasing efficiency.

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.

The term refers to the difference in utilities between successive states and updates the utility function based on this, scaled by a learning rate α:

$$U^{\pi}(s_t) = U^{\pi}(s) + \alpha\left[R(s_t, \pi(s_t), s_{t+1}) + \gamma U^{\pi}(s_{t+1}) - U^{\pi}(s_t)\right]$$

The learning rate can be a fixed parameter or a function indirect proportional to the number of 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, increasing efficiency.

⊖ 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.

Reinforcement Learning Algorithms

Temporal-Difference Learning (TD Learning) (type: Model-free, Offline Learning)

With

$$U^{\pi}(s_t) = U^{\pi}(s) + \alpha \overbrace{[\underbrace{R(s_t, \pi(s_t), s_{t+1}) + \gamma U^{\pi}(s_{t+1})}_{\text{TD target}} - U^{\pi}(s_t)]}^{\text{TD error}}$$

the Temporal Difference Equation only needs the next state s_{t+1} to learn incrementally.

TD error is the error in the estimation made a time t, and TD target is an estimate of the return.

TD predicts the combination of immediate reward r_{t+1} and its reward prediction at the next time step $U^{\pi}(s_{t+1})$. The new prediction is compared against what it was expected to be. If they're different, the algorithm calculates the 'temporal difference' to adjust the old prediction toward the new prediction. The predictions gradually become more accurate by constantly striving to bring predictions closer to the actual values.

Reinforcement Learning Algorithms

Temporal-Difference Learning (TD Learning) (type: Model-free, Offline Learning)

real-life example:

You are leaving for home after work and estimate that it will take you 30 minutes to get home. It takes 5 minutes to reach the bus stop. You notice the bus is late, so you reestimate that you will be home in 40 minutes instead. On the bus, you see the bus is stuck behind a slow truck, and you reestimate you will be home after in total 50 minutes. Once you exit the bus, you estimate that you will be home in total 45 minutes instead as you were able to exit at a bus stop closer to your home than usually.
You update your prediction at ever step - the same as TD does. TD asymptotically then converges to the correct prediction.

In the following we will see a variety of Temporal Difference algorithms.

Reinforcement Learning Algorithms

Q-Learning (type: Model-free, Online Learning, Active TD Learning, Off-Policy)

Q-Learning is an active TD Learning algorithm. In Active Reinforcement Learning, the policy is be updated and utility now depends on the state-action pair as each state may perform different actions following different policies. The update rule is unchanged, but now the utility of a state is represented as the utility of a state-action pair using a Q-function.

Reinforcement Learning Algorithms

Q-Learning (type: Model-free, Online Learning, Active TD Learning, Off-Policy)

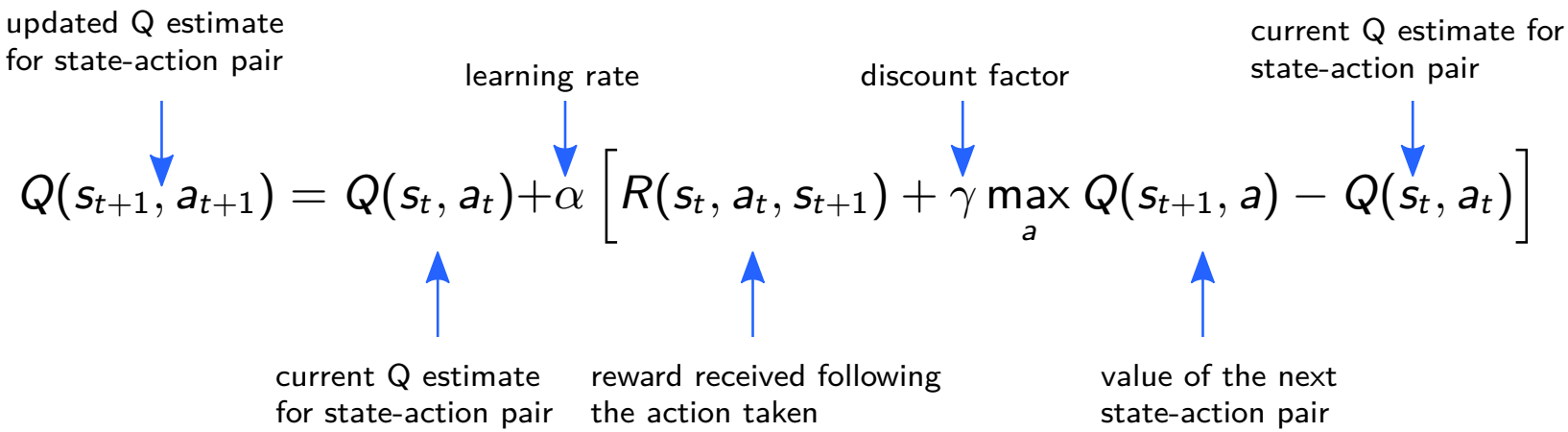
Q-Learning is an active TD Learning algorithm. In Active Reinforcement Learning, the policy is be updated and utility now depends on the state-action pair as each state may perform different actions following different policies. The update rule is unchanged, but now the utility of a state is represented as the utility of a state-action pair using a Q-function.

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha \left[R(s_t, a_t, s_{t+1}) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Reinforcement Learning Algorithms

Q-Learning (type: Model-free, Online Learning, Active TD Learning, Off-Policy)

Q-Learning is an active TD Learning algorithm. In Active Reinforcement Learning, the policy is be updated and utility now depends on the state-action pair as each state may perform different actions following different policies. The update rule is unchanged, but now the utility of a state is represented as the utility of a state-action pair using a Q-function.



Reinforcement Learning Algorithms

Q-Learning (type: Model-free, Online Learning, Active TD Learning, Off-Policy)

Q-Learning is an active TD Learning algorithm. In Active Reinforcement Learning, the policy is be updated and utility now depends on the state-action pair as each state may perform different actions following different policies. The update rule is unchanged, but now the utility of a state is represented as the utility of a state-action pair using a Q-function.

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha \left[R(s_t, a_t, s_{t+1}) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Q-Learning is Off-Policy: the target, or the utility of the next state, maximizes the Q-function over possible actions in the next state (regardless of policy). This way, the actual action in the next state is not needed.

Reinforcement Learning Algorithms

Q-Learning (type: Model-free, Online Learning, Active TD Learning, Off-Policy)

Q-Learning is an active TD Learning algorithm. In Active Reinforcement Learning, the policy is be updated and utility now depends on the state-action pair as each state may perform different actions following different policies. The update rule is unchanged, but now the utility of a state is represented as the utility of a state-action pair using a Q-function.

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha \left[R(s_t, a_t, s_{t+1}) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Q-Learning is Off-Policy: the target, or the utility of the next state, maximizes the Q-function over possible actions in the next state (regardless of policy). This way, the actual action in the next state is not needed.

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

Reinforcement Learning Algorithms

Q-Learning (type: Model-free, Online Learning, Active TD Learning, Off-Policy)

Q-Learning is an active TD Learning algorithm. In Active Reinforcement Learning, the policy is be updated and utility now depends on the state-action pair as each state may perform different actions following different policies. The update rule is unchanged, but now the utility of a state is represented as the utility of a state-action pair using a Q-function.

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha \left[R(s_t, a_t, s_{t+1}) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Q-Learning is Off-Policy: the target, or the utility of the next state, maximizes the Q-function over possible actions in the next state (regardless of policy). This way, the actual action in the next state is not needed.

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

⊖ 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.

Reinforcement Learning Algorithms

Q-Learning (type: Model-free, Online Learning, Active TD Learning, Off-Policy)

What is 'Q' in Q-learning?

The Q stands for **quality**, which means it specifies the quality of an action taken by the agent.

Q-table:
A Q-table or matrix is created while performing the Q-learning. The table follows the state and action pair, i.e., $[s, a]$, and initializes the values to zero. After each action, the table is updated, and the q-values are stored within the table. The RL agent uses this Q-table as a reference table to select the best action based on the q-values.

Reinforcement Learning Algorithms

SARSA (type: Model-free, Online Learning, Active TD Learning, On-Policy)

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. An on-policy method estimates $Q(s, a)$ for the current behavior policy π and for all states s and actions a. Hence we consider transitions from the state-action pair to state-action pair and learn the values of state-action pairs. SARSA uses every element out of the events $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ and hence the name.

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

Reinforcement Learning Algorithms

SARSA (type: Model-free, Online Learning, Active TD Learning, On-Policy)

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. An on-policy method estimates $Q(s, a)$ for the current behavior policy π and for all states s and actions a. Hence we consider transitions from the state-action pair to state-action pair and learn the values of state-action pairs. SARSA uses every element out of the events $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ and hence the name.

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

In SARSA, the Q-value is updated taking into account the action a_{t+1} in the state s_{t+1}. In Q-learning as a SARS algorithm (not considering the next state's action), the action with the highest Q-value in the next state, s_{t+1}, is used to update the Q-table.

Reinforcement Learning Algorithms

SARSA (type: Model-free, Online Learning, Active TD Learning, On-Policy)

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. An on-policy method estimates $Q(s, a)$ for the current behavior policy π and for all states s and actions a. Hence we consider transitions from the state-action pair to state-action pair and learn the values of state-action pairs. SARSA uses every element out of the events $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ and hence the name.

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

In SARSA, the Q-value is updated taking into account the action a_{t+1} in the state s_{t+1}. In Q-learning as a SARS algorithm (not considering the next state's action), the action with the highest Q-value in the next state, s_{t+1}, is used to update the Q-table.

Due to this difference, the SARSA algorithm knows the action for the next state - no need to maximize the Q-function over all possible actions.

Reinforcement Learning Algorithms

SARSA (type: Model-free, Online Learning, Active TD Learning, On-Policy)

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. An on-policy method estimates $Q(s, a)$ for the current behavior policy π and for all states s and actions a. Hence we consider transitions from the state-action pair to state-action pair and learn the values of state-action pairs. SARSA uses every element out of the events $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ and hence the name.

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

⊕ 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.

Reinforcement Learning Algorithms

SARSA (type: Model-free, Online Learning, Active TD Learning, On-Policy)

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. An on-policy method estimates $Q(s, a)$ for the current behavior policy π and for all states s and actions a. Hence we consider transitions from the state-action pair to state-action pair and learn the values of state-action pairs. SARSA uses every element out of the events $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ and hence the name.

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

⊕ 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.

⊖ 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.

Reinforcement Learning Algorithms

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

	Model-free	**Model-based**
offline learning (passive)	Direct Utility Estimation, Temporal Difference Learning (TD Learning)	Adaptive Dynamic Programming (ADP)
online learning (active)	Q-Learning (Active TD Learning), SARSA (Active TD Learning)	Exploration (Active ADP)

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

Domain Selection

Deciding which types of input and feedback your agent should **pay attention to** is a hard problem to solve. This is known as domain selection.

Domain Selection

Deciding which types of input and feedback your agent should **pay attention to** is a hard problem to solve. This is known as domain selection.

Algorithms that are learning how to play video games can mostly ignore this problem, since the environment is man-made and strictly limited. Thus, video games provide the sterile environment of the lab, where ideas about reinforcement learning can be tested.

Domain Selection

Deciding which types of input and feedback your agent should **pay attention to** is a hard problem to solve. This is known as domain selection.

Algorithms that are learning how to play video games can mostly ignore this problem, since the environment is man-made and strictly limited. Thus, video games provide the sterile environment of the lab, where ideas about reinforcement learning can be tested.

Domain selection requires human decisions, usually based on knowledge or theories about the problem to be solved; e.g. selecting the domain of input for an algorithm in a self-driving car might include choosing to include radar sensors in addition to cameras and GPS data.

Characteristics of Reinforcement Learning

There are mainly 5 **characteristics of reinforcement learning**:

1. No supervisor, only a reward: Unlike supervised learning, reinforcement learning does not have any pre-experience. It does not have any given example of outcome or targeted variable. It learns from itself only.

2. Sequential Decision Making: It can be understood as an algorithm or a technique that takes dynamics of the world into decision, it won't stop itself, and delays parts of the problem until it must not be solved.

3. Time as a crucial factor: Time always plays as a crucial factor in reinforcement learning.

4. Feedback is always delayed, not instantaneous.

5. The agent's actions determine the subsequent data it receives.

Characteristics of Reinforcement Learning

There are mainly 5 **characteristics of reinforcement learning**:

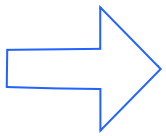
1. No supervisor, only a reward: Unlike supervised learning, reinforcement learning does not have any pre-experience. It does not have any given example of outcome or targeted variable. It learns from itself only.

2. Sequential Decision Making: It can be understood as an algorithm or a technique that takes dynamics of the world into decision, it won't stop itself, and delays parts of the problem until it must not be solved.

3. Time as a crucial factor: Time always plays as a crucial factor in reinforcement learning.

4. Feedback is always delayed, not instantaneous.

5. The agent's actions determine the subsequent data it receives.



Reinforcement learning differs from both supervised and unsupervised learning by how it interprets inputs.

Reinforcement Learning vs. other types of learning

Reinforcement learning differs from both supervised and unsupervised learning by how it **interprets input**.

We can illustrate their difference by describing **what** they learn about a *thing*.

Unsupervised learning: *That thing is like this other thing.* It has similar properties. (The algorithms learn similarities without names, and can spot the inverse and perform anomaly detection by recognizing what is unusual.)

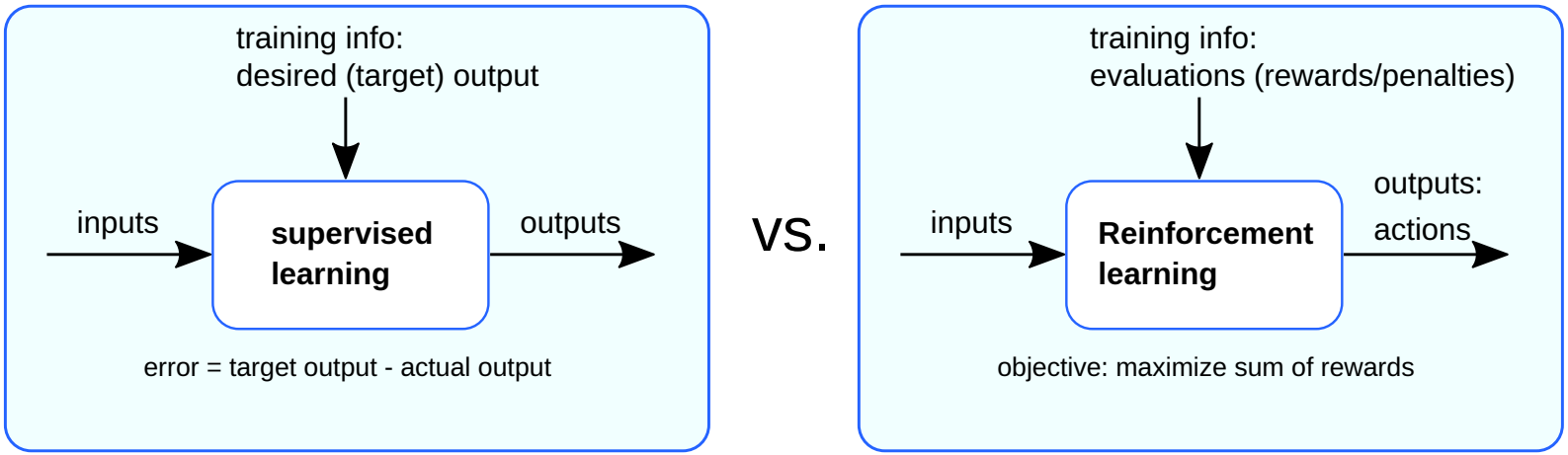
Supervised learning: *That thing is an* `apple`*.* These algorithms learn the correlations between data instances and their labels; that is, they require a labelled dataset. Those labels are used to supervise and correct the algorithm as it makes wrong guesses when predicting labels.

Reinforcement learning: *Eat that thing because it tastes good and will keep you alive longer.* Actions based on short- and long-term rewards. Reinforcement learning can be thought of as supervised learning in an environment of sparse feedback.

Reinforcement Learning vs. other types of learning

Reinforcement learning differs from both supervised and unsupervised learning by how it **interprets input**.

We can illustrate their difference by describing **how** they learn about a *thing*.



Advantages and Disadvantages of Reinforcement Learning

- Reinforcement learning can be used to solve very complex problems that cannot be solved by conventional techniques.
- The model can correct the errors that occurred during the training process.
- Training data is obtained via the direct interaction of the agent with the environment.
- Reinforcement learning can handle environments that are non-deterministic, meaning that the outcomes of actions are not always predictable. This is useful in real-world applications where the environment may change over time or is uncertain.
- Reinforcement learning can be used to solve a wide range of problems, including those that involve decision making, control, and optimization.
- Reinforcement learning is a flexible approach that can be combined with other machine learning techniques, such as deep learning, to improve performance.

Advantages and Disadvantages of Reinforcement Learning

- Reinforcement learning is not preferable to use for solving simple problems.
- Reinforcement learning needs a lot of data and a lot of computation
- Reinforcement learning is highly dependent on the quality of the reward function. If the reward function is poorly designed, the agent may not learn the desired behavior.
- Reinforcement learning can be difficult to debug and interpret. It is not always clear why the agent is behaving in a certain way, which can make it difficult to diagnose and fix problems.

another look at Q Learning

The Bellman equation can be used to solve for the optimal policy using Policy iteration or Value Iteration, which is an iterative method to pass the utility values from a future state to the current state.

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s', a') | s, a\right]$$

Q_i will converge to Q^* as $i \to \infty$, so a policy maps a state to an action.

another look at Q Learning

The Bellman equation can be used to solve for the optimal policy using Policy iteration or Value Iteration, which is an iterative method to pass the utility values from a future state to the current state.

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a')|s,a\right]$$

Q_i will converge to Q^* as $i \to \infty$, so a policy maps a state to an action. To be more specific, Q maps state-action pairs to the highest combination of immediate reward with all future rewards that might be harvested by later actions in the trajectory. We have seen the equation for Q:

$$Q(s_{t+1},a_{t+1}) = Q(s_t,a_t) + \alpha\left[R(s_t,a_t,s_{t+1}) + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right]$$

Internally, the Q-function is encoded by a Q-table, a table where each cell corresponds to a state-action pair value.

another look at Q Learning

The Bellman equation can be used to solve for the optimal policy using Policy iteration or Value Iteration, which is an iterative method to pass the utility values from a future state to the current state.

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a')|s,a\right]$$

Q_i will converge to Q^* as $i \to \infty$, so a policy maps a state to an action. To be more specific, Q maps state-action pairs to the highest combination of immediate reward with all future rewards that might be harvested by later actions in the trajectory. We have seen the equation for Q:

$$Q(s_{t+1},a_{t+1}) = Q(s_t,a_t) + \alpha\left[R(s_t,a_t,s_{t+1}) + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right]$$

Internally, the Q-function is encoded by a Q-table, a table where each cell corresponds to a state-action pair value.

What is the problem with this?

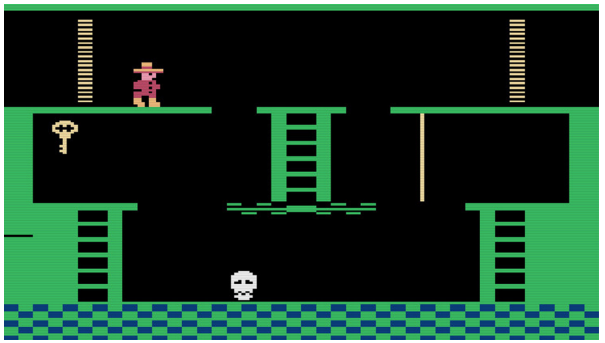
another look at Q Learning

It is **not scalable**. One must compute $Q(s, a)$ for every state-action pair. It can be computationally infeasible to compute the entire state space.

another look at Q Learning

It is **not scalable**. One must compute $Q(s, a)$ for every state-action pair. It can be computationally infeasible to compute the entire state space.

example:
Atari games have RGB images of the size 210×160 pixels, corresponding to an observation space of the shape $(210, 160, 3)$, containing values ranging from 0 to 255. This gives us $256^{210 \times 160 \times 3} = 256^{100800}$ possible observations (for comparison, we have approximately 10^{80} atoms in the observable universe).

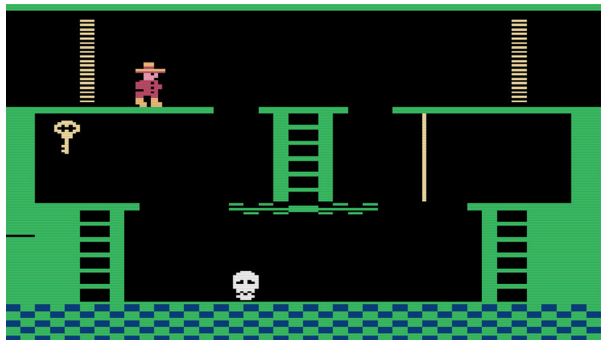


A Reinforcement Learning algorithm would have to deal with a state space that size $\Rightarrow$ creating and updating a Q-table for that environment would not be efficient.

another look at Q Learning

It is **not scalable**. One must compute $Q(s, a)$ for every state-action pair. It can be computationally infeasible to compute the entire state space.

example:
Atari games have RGB images of the size 210×160 pixels, corresponding to an observation space of the shape $(210, 160, 3)$, containing values ranging from 0 to 255. This gives us $256^{210 \times 160 \times 3} = 256^{100800}$ possible observations (for comparison, we have approximately 10^{80} atoms in the observable universe).



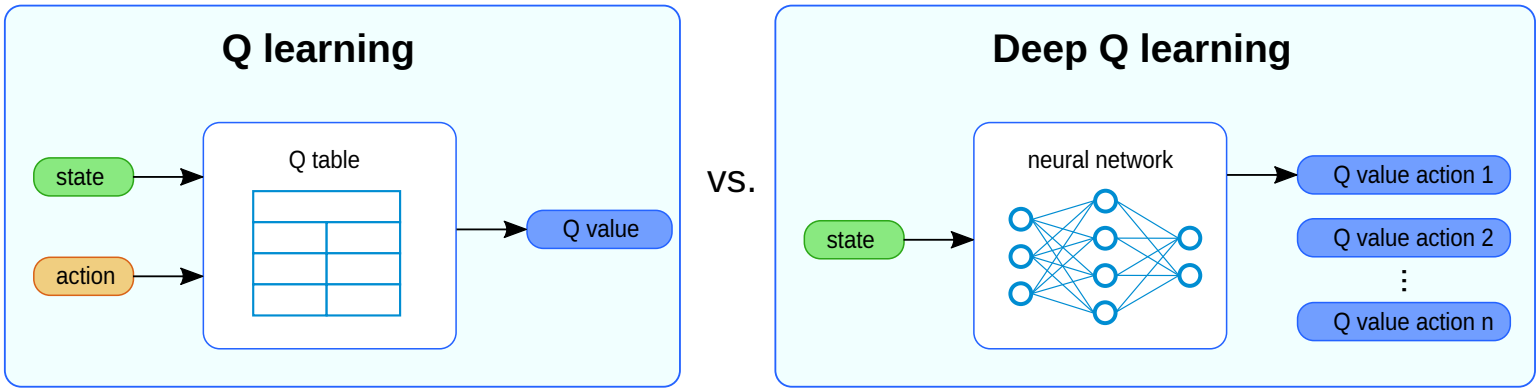
A Reinforcement Learning algorithm would have to deal with a state space that size $\Rightarrow$ creating and updating a Q-table for that environment would not be efficient.

In many decision-making problems, the states s are high-dimensional (e.g.: images from a camera, raw sensor stream from a robot) and cannot be solved by traditional Reinforcement learning algorithms.

Deep Reinforcement Learning

Solution: Use a **function approximator** to estimate $Q(s, a)$ by using a parametrized Q-function $Q_\theta(s, a)$. The function approximator used is a neural network. network.

This **neural network** will approximate, given a state, the different Q-values for each possible action at that state. And that's exactly what Deep Q-Learning does.



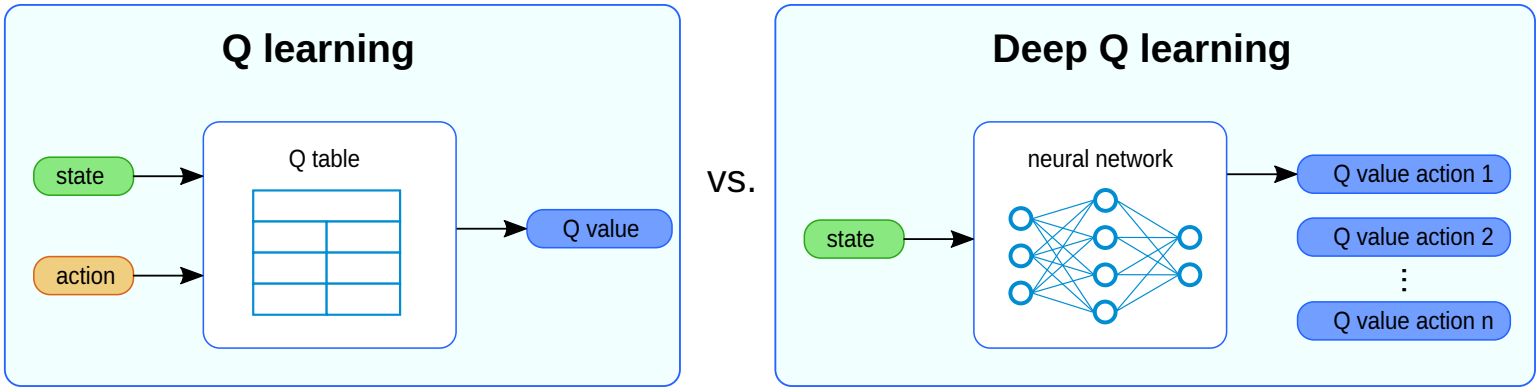
Deep Reinforcement Learning

Deep Learning is one of the best tools that we have today to handle unstructured environments. It can learn from large amounts of data or discover patterns. But this is not decision-making; it is a **recognition** problem. Reinforcement Learning provides the **decision-making**, so software agents can learn to reach their goals in those large unstructured environments.

Sometimes, algorithms (like humans) have to wait to see the fruit of their decisions: a **delayed-return environment**, where it can be difficult to understand which action leads to which outcome over many time steps. Reinforcement learning solves the hard problem of correlating immediate actions with the delayed outcomes they produce.

Reinforcement algorithms that incorporate deep neural networks can beat human experts playing numerous video games. This is less trivial than it sounds!

Deep Reinforcement Learning



Q learning: during training, the Q-value of a state-action pair is updated

updated Q estimate for state-action pair

learning rate

discount factor

current Q estimate for state-action pair

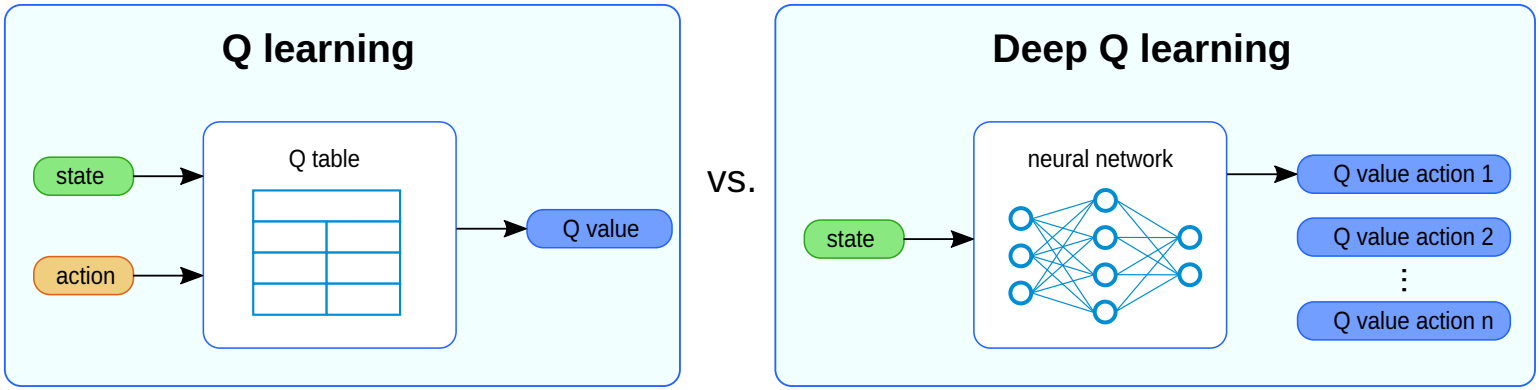
$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha \left[R(s_t, a_t, s_{t+1}) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

current Q estimate for state-action pair

reward received following the action taken

value of the next state-action pair

Deep Reinforcement Learning



Q learning: during training, the Q-value of a state-action pair is updated

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha \left[R(s_t, a_t, s_{t+1}) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Deep Q-Learning: a loss function is created Â´that compares the Q-value prediction and the Q-target and uses gradient descent to update the weights of the Deep Q-Network to approximate the Q-values better.

Q-target: $y_j = r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta)$

Q-loss: $y_j - Q(\phi_j, a_j; \theta)$

another look at Q Learning

The Deep Q-Learning with Experience Replay algorithm has two phases:

Sampling: Perform actions, store experience tuples in replay memory.

Training: Select a small batch of tuples randomly from replay memory, learn from this using a gradient descent update step.

Learning from batches of consecutive samples is **problematic**:

- samples are correlated $\Rightarrow$ inefficient learning
- current Q-network parameters determine next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from the left) $\Rightarrow$ can lead to feedback loops.

another look at Q Learning

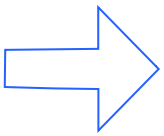
The Deep Q-Learning with Experience Replay algorithm has two phases:

Sampling: Perform actions, store experience tuples in replay memory.

Training: Select a small batch of tuples randomly from replay memory, learn from this using a gradient descent update step.

Address these problems by using **experience replay**:

- continually update a replay memory table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played
- train Q-network on random mini-batches of transitions from the replay memory, instead of consecutive samples



Deep Q-Learning with Experience Replay

Deep Reinforcement Learning

The Deep Q-Learning with Experience Replay algorithm has two phases:

Sampling: Perform actions, store experience tuples in replay memory.

Training: Select a small batch of tuples randomly from replay memory, learn from this using a gradient descent update step.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

sampling

training

Mnih, et al. *Playing atari with deep reinforcement learning.* 2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI).

Deep Reinforcement Learning

The Deep Q-Learning with Experience Replay algorithm has two phases:

Sampling: Perform actions, store experience tuples in replay memory.

Training: Select a small batch of tuples randomly from replay memory, learn from this using a gradient descent update step.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

For each timestep t of the game

Deep Reinforcement Learning

The Deep Q-Learning with Experience Replay algorithm has two phases:

Sampling: Perform actions, store experience tuples in replay memory.

Training: Select a small batch of tuples randomly from replay memory, learn from this using a gradient descent update step.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

With small probability, select a random action (explore), otherwise select greedy action from current policy.

Deep Reinforcement Learning

The Deep Q-Learning with Experience Replay algorithm has two phases:

Sampling: Perform actions, store experience tuples in replay memory.

Training: Select a small batch of tuples randomly from replay memory, learn from this using a gradient descent update step.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

Take the action a_t, and observe the reward r_t and next state s_{t+1}.

Deep Reinforcement Learning

The Deep Q-Learning with Experience Replay algorithm has two phases:

Sampling: Perform actions, store experience tuples in replay memory.

Training: Select a small batch of tuples randomly from replay memory, learn from this using a gradient descent update step.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

Store transition in replay memory

Deep Reinforcement Learning

The Deep Q-Learning with Experience Replay algorithm has two phases:

Sampling: Perform actions, store experience tuples in replay memory.

Training: Select a small batch of tuples randomly from replay memory, learn from this using a gradient descent update step.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

Experience Replay: Sample a random mini-batch of transitions from replay memory, perform a gradient descent step

Deep Reinforcement Learning

Experience Replay in Deep Q-Learning has the following benefits:

- Make more efficient use of the experiences during the training.

Usually, in online reinforcement learning, the agent interacts with the environment, gets experiences (state, action, reward, and next state), learns from them (updates the neural network), and discards them. This is not efficient.
Experience replay helps by using the experiences of the training more efficiently. We use a replay buffer that saves experience samples that we can reuse during the training. This allows the agent to learn from the same experiences multiple times.

Deep Reinforcement Learning

Experience Replay in Deep Q-Learning has the following benefits:

- Make more efficient use of the experiences during the training.
- Avoid forgetting previous experiences and reduce the correlation between experiences.

The problem we get if we give sequential samples of experiences to our neural network is that it tends to forget the previous experiences as it gets new experiences. For instance, if the agent is in the first level and then in the second, which is different, it can forget how to behave and play in the first level.
The solution is to create a Replay Buffer that stores experience tuples while interacting with the environment and then sample a small batch of tuples. This prevents the network from only learning about what it has done immediately before.

Deep Reinforcement Learning

Experience Replay in Deep Q-Learning has the following benefits:

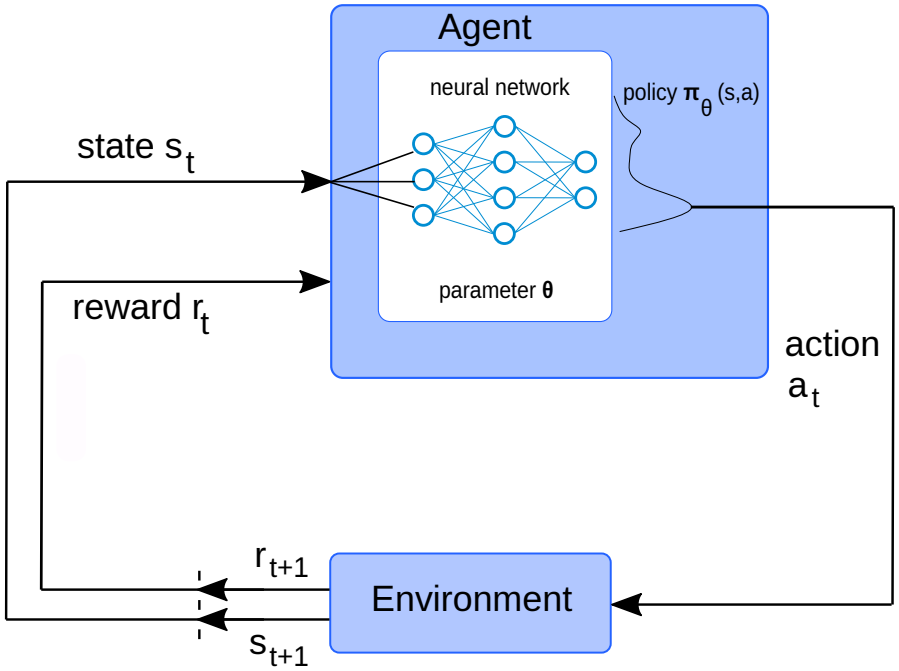
- Make more efficient use of the experiences during the training.
- Avoid forgetting previous experiences and reduce the correlation between experiences.
- Removing correlation.

  By randomly sampling the experiences, we remove correlation in the observation sequences and avoid action values from oscillating or diverging catastrophically.

Deep Reinforcement Learning

Neural network becomes part of the agent

- Input is the state, outputs an action
- Reward adjusts the weights (policy)
- Eventually becomes fine-tuned for optimization

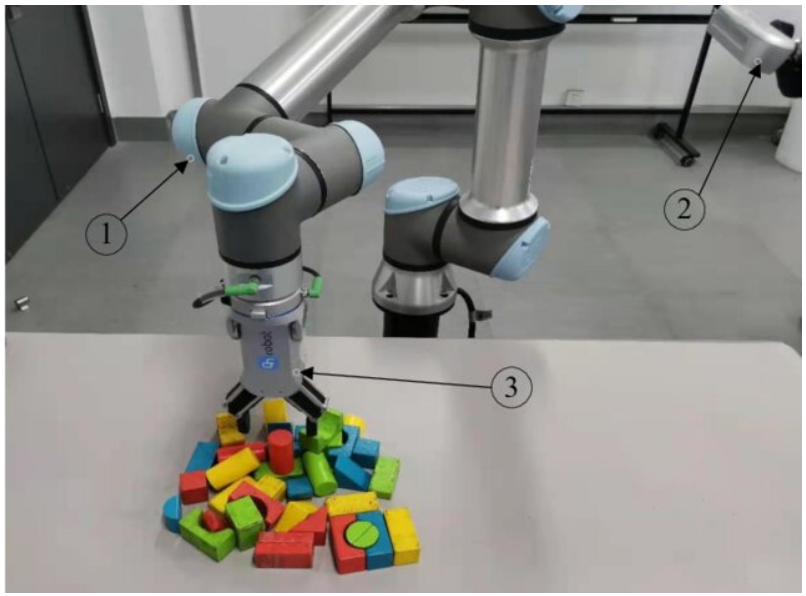


another look at Q Learning

What is (still) a problem with Q-learning?

The Q-function can be very complicated!

example: a robot grasping an object has a very high-dimensional state



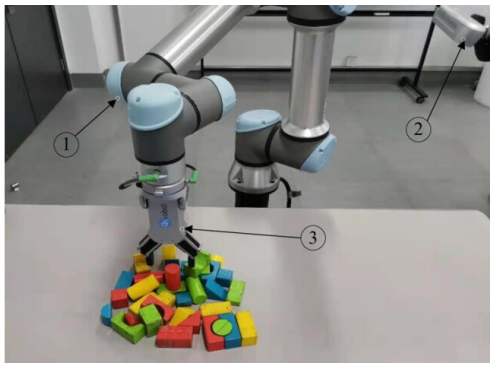
credit: D. Ren et al. (2021), *Fast-Learning Grasping and Pre-Grasping via Clutter Quantization and Q-map Masking*

another look at Q Learning

What is (still) a problem with Q-learning?

The Q-function can be very complicated!

example: a robot grasping an object has a very high-dimensional state
$\Rightarrow$ hard to learn exact the value of every (state, action) pair

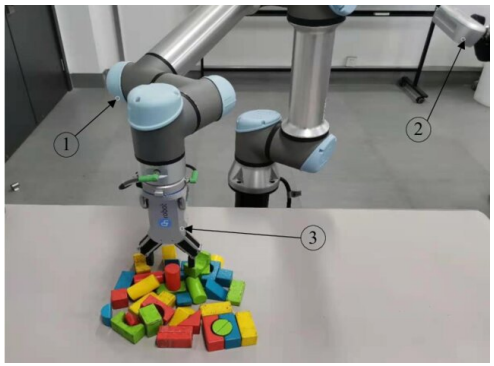


another look at Q Learning

What is (still) a problem with Q-learning?

The Q-function can be very complicated!

example: a robot grasping an object has a very high-dimensional state
$\Rightarrow$ hard to learn exact the value of every (state, action) pair



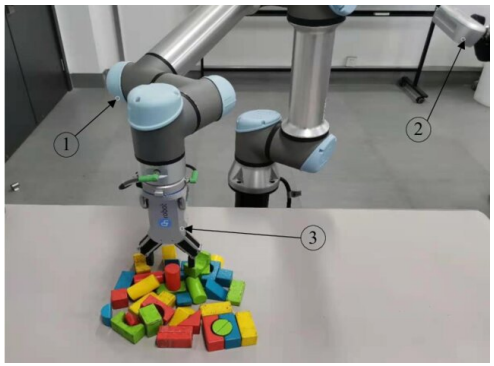
But the policy can be much simpler: carry out the movement.

another look at Q Learning

What is (still) a problem with Q-learning?

The Q-function can be very complicated!

example: a robot grasping an object has a very high-dimensional state
$\Rightarrow$ hard to learn exact the value of every (state, action) pair



But the policy can be much simpler: carry out the movement.

Can we **learn a policy directly**, e.g. finding the best policy from a collection of policies?

another look at Q Learning

Addressing Challenges of Reinforcement Learning

- Policy gradient adjustment method
- Actor-critic method

Policy Gradient Adjustment

We can define a class of parameterized policies:

$$\Pi = \{\pi_\theta, \theta \in \mathcal{R}^m\}$$

Policy Gradient Adjustment

We can define a class of parameterized policies:

$$\Pi = \{\pi_\theta, \theta \in \mathcal{R}^m\}$$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t \leq 0} \gamma^t r_t | \pi_\theta\right]$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

Policy Gradient Adjustment

We can define a class of parameterized policies:

$$\Pi = \{\pi_\theta, \theta \in \mathcal{R}^m\}$$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t \leq 0} \gamma^t r_t | \pi_\theta\right]$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

This can be solved with **gradient ascent** on the policy parameters.

Policy Gradient Adjustment

REINFORCE belongs to a special class of Reinforcement Learning algorithms called Policy Gradient algorithms. A simple implementation of this algorithm would involve creating a Policy: a model that takes a state as input and generates the probability of taking an action as output.

The REINFORCE algorithm:

We can write for the expected reward of a trajectory $\tau = (s_0, a_0, r_0, s_1, ...)$:

$$J(\theta) = \mathbb{E}_{r \sim p(\tau|\theta)}[r(\tau)]$$
$$= \int_{\tau} r(\tau) p(\tau; \theta) \mathrm{d}\tau$$

We differentiate this:

$$\nabla_\theta J(\theta) = \int_{\tau} \nabla\theta p(\tau; \theta) \mathrm{d}\tau$$

This is intractable! Gradient of $\mathbb{E}$ is problematic when $p = p(\theta)$

Policy Gradient Adjustment

However, we can apply a trick:

$$\nabla\theta p(\tau;\theta) = p(\tau;\theta)\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$$

When we inject this back:

$$\begin{aligned}\nabla\theta J(\theta) &= \int_\tau (r(\tau)\nabla\theta \log p(\tau;\theta))p(\tau;\theta)\mathrm{d}\tau \\ &= \mathbb{E}_{r\sim p(\tau;\theta)}\left[r(\tau)\nabla_\tau \log p(\tau;\theta)\right]\end{aligned}$$

This can now be estimated with Monte Carlo sampling.

Policy Gradient Adjustment

Can we compute those quantities **without knowing the transition probabilities**?

We have:

$$p(\tau;\theta) = \prod_{t \leq 0} p(s_{+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Thus:

$$\log p(\tau;\theta) = \sum_{t \leq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$$

When differentiating:

$$\nabla\theta \log p(\tau;\theta) = \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

This does not depend on transition probabilities!
Therefore when sampling a trajectory t, we can estimate $J(\theta)$ with

$$\nabla\theta J(\theta) \approx \sum_{t \geq 0} r(\tau)\nabla\theta \log \pi_\theta(a_t|s_t)$$

Policy Gradient Adjustment

We can interprete the gradient estimator the in the following way:

$$\nabla\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla\theta \log \pi_\theta(a_t|s_t)$$

- If $r(\tau)$ is high, push up the probabilities of the actions seen
- If $r(\tau)$ is low, push down the probabilities of the actions seen

Policy Gradient Adjustment

We can interprete the gradient estimator the in the following way:

$$\nabla\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla\theta \log \pi_\theta(a_t|s_t)$$

- If $r(\tau)$ is high, push up the probabilities of the actions seen
- If $r(\tau)$ is low, push down the probabilities of the actions seen

It might seem simplistic to say that if a trajectory is good then all its actions were good. But in expectation, it averages out!
However, this also suffers from high variance because credit assignment is really hard. Can we help the estimator?

Policy Gradient Adjustment

Variance reduction

Gradient estimator:

$$\nabla\theta J(\theta) \approx \sum t \geq 0 r(\tau) \nabla\theta \log \pi_\theta(a_t|s_t)$$

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state:

$$\nabla\theta J(\theta) \approx \sum t \geq 0 \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Second idea: Use the discount factor γ to ignore delayed effects

$$\nabla\theta J(\theta) \approx \sum t \geq 0 \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Policy Gradient Adjustment

Variance reduction

Gradient estimator:

$$\nabla\theta J(\theta) \approx \sum t \geq 0 r(\tau) \nabla\theta \log \pi_\theta(a_t|s_t)$$

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state:

$$\nabla\theta J(\theta) \approx \sum t \geq 0 \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Second idea: Use the discount factor γ to ignore delayed effects

$$\nabla\theta J(\theta) \approx \sum t \geq 0 \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Problem: The value of a trajectory isn't necessarily meaningful. E.g.: if rewards are all positive, you keep pushing up probabilities of action.

Policy Gradient Adjustment

What is important then? Whether a reward is better or worse than what you expect to get.

Policy Gradient Adjustment

What is important then? Whether a reward is better or worse than what you expect to get.

Idea: Introduce a **baseline function** that is dependent on the state.

The estimator is then:

$$\nabla\theta J(\theta) \approx \sum t \geq 0 \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

baseline function

Policy Gradient Adjustment

How to **choose** the baseline?

$$\nabla\theta J(\theta) \approx \sum t \geq 0 \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

A **simple** baseline: constant moving average of rewards experienced so far from all trajectories

Policy Gradient Adjustment

How to **choose** the baseline?

$$\nabla\theta J(\theta) \approx \sum t \geq 0 \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

A **simple** baseline: constant moving average of rewards experienced so far from all trajectories

A **better** baseline: Want to push up the probability of an action from a state, if this action was better than the expected value from that state. That's Q-function and value function!

Policy Gradient Adjustment

How to **choose** the baseline?

$$\nabla\theta J(\theta) \approx \sum t \geq 0 \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

A **simple** baseline: constant moving average of rewards experienced so far from all trajectories

A **better** baseline: Want to push up the probability of an action from a state, if this action was better than the expected value from that state. That's Q-function and value function!

Intuitively: We are happy with an action a_t in a state s_t if the value $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large. On the contrary, we are unhappy with an action if it is small.

Policy Gradient Adjustment

How to **choose** the baseline?

$$\nabla\theta J(\theta) \approx \sum t \geq 0 \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

A **simple** baseline: constant moving average of rewards experienced so far from all trajectories

A **better** baseline: Want to push up the probability of an action from a state, if this action was better than the expected value from that state. That's Q-function and value function!

Intuitively: We are happy with an action a_t in a state s_t if the value $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large. On the contrary, we are unhappy with an action if it is small.

Using this, we get the estimator:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (Q^{\pi_\theta}(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

The Actor-Critic Algorithm

Problem: We don't know Q or V. Can we learn them?

The Actor-Critic Algorithm

Problem: We don't know Q or V. Can we learn them?

Solution: Yes, we can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

The actor decides which action to take, and the critic tells the actor how good its action was and how it should be adjusted.
This can incorporate special Q-learning algorithms such as experience replay.

The Actor-Critic Algorithm

Problem: We don't know Q or V. Can we learn them?

Solution: Yes, we can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

The actor decides which action to take, and the critic tells the actor how good its action was and how it should be adjusted.
This can incorporate special Q-learning algorithms such as experience replay.

Remark: Using the **advantage function** we can define how much an action was better than expected:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

The Actor-Critic Algorithm

the complete Actor-Critic Algorithm:

initialize policy parameters θ, critic parameters φ
for *iteration 1, 2,...* **do**
  sample m trajectories under the current policy
  $\Delta\theta \leftarrow 0$
  for $i = 1, .., m$ **do**
    for $t = 1, .., T$ **do**
      $A_t = \sum_{t' \geq t} \gamma^{t'-t} r_t^i - V_\varphi(s_t^i)$
      $\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i | s_t^i)$
    end
  end
  $\Delta\varphi \leftarrow \sum_i \sum_t \nabla_\varphi \|A_t^i\|^2$
  $\theta \leftarrow \alpha\Delta\theta$
  $\varphi \leftarrow \beta\Delta\varphi$
end

Applications of Reinforcement Learning

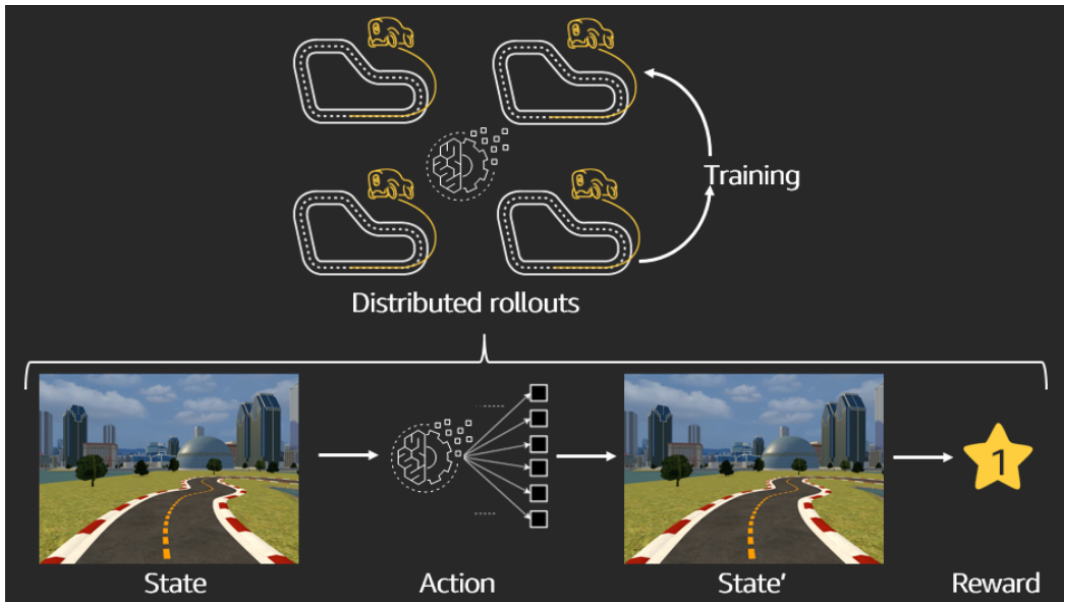
Typical reinforcement learning applications: Where *action must be taken*

- Autonomous cars (including planetary rovers)
- Datacenters cooling
- Traffic light control
- Healthcare
- Image processing
- Robotics (including industrial robots and service robots)
- Marketing
- Gaming

Autonomous driving

Autonomous driving uses Reinforcement Learning with the help of synthetic environments to target the significant problems of Trajectory optimization. The agents learn motion planning, route changing, decision and position of parking and speed control, etc.

example:
DeepTraffic is an open-source environment by MIT that combines the powers of Reinforcement Learning, Deep Learning, and Computer Vision to build algorithms used for autonomous vehicles (such as cars, drones etc.)



Traffic light control

Reinforcement Learning models are trained with the objective of learning a policy using a value function that optimally controls the traffic light based on the current status of the traffic, preventing traffic congestion.
The decision-making needs to be dynamic depending upon the arrival rate of traffic from different directions, which ought to vary at different times of the day. Also, a policy trained for an intersection with n lanes cannot be re-used in an intersection with m lanes.

example:
video sequences from traffic scenes annotated by the software *V7*

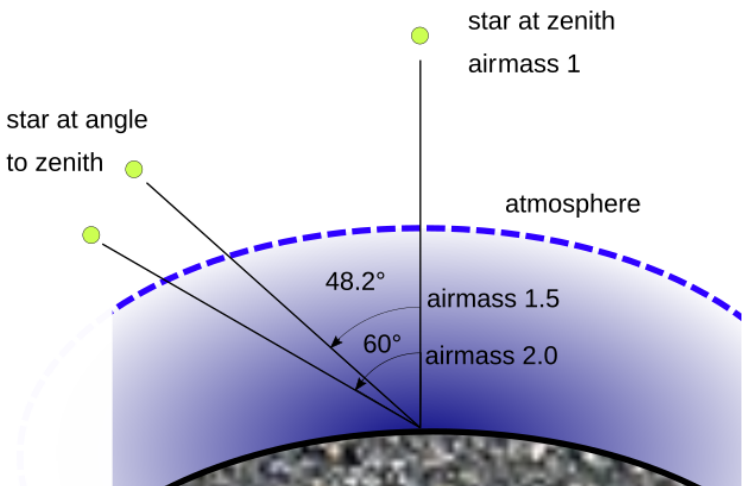
Telescope Scheduling

Survey Strategy Planning Using Reinforcement Learning

Logical approach to a survey:

- obtain high-quality images/data
- observe multiple objects
- time domain astronomy: repeat observations

$\Rightarrow$ often requires scheduling for efficiency and optimization



Airmass is a measure of the amount of air along the line of sight when observing through Earth's atmosphere.

Telescope Scheduling

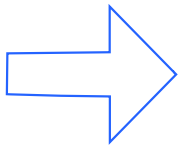
Many surveys so far follow a common pattern:

Sloan Digital Sky Survey (SDSS)

- Over 300 million objects
- Manual planning

Dark Energy Survey (DES)

- 16,000 pointings, image every 2 minutes
- Hand-tuned computerized simulations



Robotic Telescopes, but a lot of manual work

Telescope Scheduling

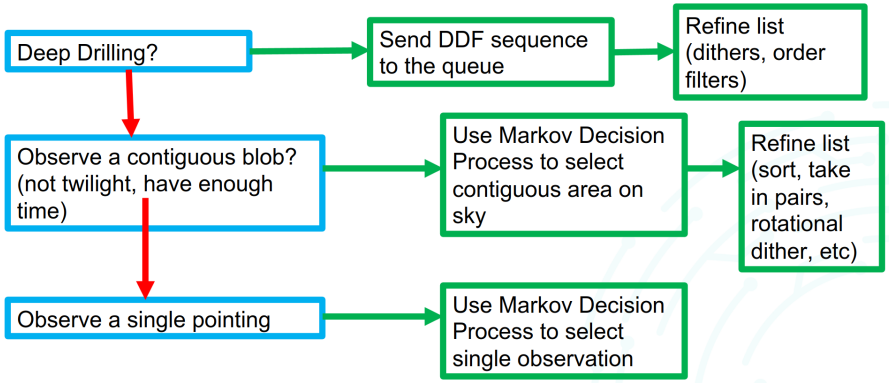
idea:

see sky surveying analog to classic computer games:
surveying the sky is like navigating a 2D plane while collecting points of different values (rewards)

Space Surveyors: A computer game created for Rubin Observatory outreach

Telescope Scheduling for LSST

The Rubin Observatory Scheduler



credit: Peter Yoachim, SCOC-Science Collaborations Workshop December 2020

`https://docushare.lsst.org/docushare/dsweb/Get/Document-36764/scoc2020_session2a.pdf`

Draft version October 12, 2018
Typeset using LaTeX default style in AASTeX62

A Framework for Telescope Schedulers: With Applications to the Large Synoptic Survey Telescope

Elahesadat Naghib,[1] Peter Yoachim,[2] Robert J. Vanderbei,[1] Andrew J. Connolly,[2] and R. Lynne Jones[2]

[1]*Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ 08540, USA*
[2]*Department of Astronomy, University of Washington, Box 351580, U.W., Seattle, WA 98195, USA*

Abstract

How ground-based telescopes schedule their observations in response to competing science priorities and constraints, variations in the weather, and the visibility of a particular part of the sky can significantly impact their efficiency. In this paper we introduce the Feature-Based telescope scheduler that is an automated, proposal-free decision making algorithm that offers *controllability* of the behavior, *adjustability* of the mission, and quick *recoverability* from interruptions for large ground-based telescopes. By framing this scheduler in the context of a coherent mathematical model the functionality and performance of the algorithm is simple to interpret and adapt to a broad range of astronomical applications. This paper presents a generic version of the Feature-Based scheduler, with minimal manual tailoring, to demonstrate its potential and flexibility as a foundation for large ground-based telescope schedulers which can later be adjusted for other instruments. In addition, a modified version of the Feature-Based scheduler for the Large Synoptic Survey Telescope (LSST) is introduced and compared to previous LSST scheduler simulations.

Keywords: Artificial intelligence, autonomous telescope, LSST, reinforcement learning, scheduling, stochastic optimization

Summary

Why use Reinforcement Learning?

- It helps you to find which situation needs an action
- Helps you to discover which action yields the highest reward over the longer period.
- Reinforcement Learning also provides the learning agent with a reward function.
- It also allows it to figure out the best method for obtaining large rewards.

Summary

When **not** to use Reinforcement Learning?

Reinforcement learning cannot be applied in all situations. These are the conditions on when to avoid it:

- When you have enough data to solve the problem with a supervised learning method
- You need to remember that Reinforcement Learning is computing-heavy and time-consuming. in particular when the action space is large.

Summary

Challenges of Reinforcement Learning

- Feature/reward design which should be very involved
- Parameters may affect the speed of learning.
- Realistic environments can have partial observability.
- Too much Reinforcement may lead to an overload of states which can diminish the results.
- Realistic environments can be non-stationary.

Outlook

In the following (and last) session, we will take a look at general Architectures of (large) Machine Learning Projects.