Reinforcement Learning.

Learn how to take actions in an environment. Reward is given only at the final solution.

- Example: learn to play backgammon.
- Final result → win game / lose game.
- Learn to fly a helicopter.
- Learn to move a robot to find the exit of a maze.

Decision maker - agent
Environment

Problem: reward only occurs at the final solution. How to assign credit to all the actions. Credit assignment problem.

A reinforcement learner must generate an internal value for the intermediate states or actions as to how good they are for leading us to the goal.

Solution requires a sequence of actions. A Markov decision process is used to model the agent.
Single State Case: K-Armed Bandit.

K-armed bandit - a slot machine with K levers.
Action - choose and pull one of the levers and win money.
Classification Problem - choose one of K.

$Q(a) \triangleq \text{value of action } a.$

$Q(a) = 0$, for all $a$.

When we try action $a$, we get $r_a > 0$.

Suppose rewards are deterministic (i.e., same reward for each action - i.e., each time we pull the lever).

Explore/exploit,

choose action $a^*$ if $Q(a^*) = \max_a Q(a)$

If rewards are stochastic, we get different rewards for same action.

Amount of reward $p(r_{1|a})$.

Define $Q_r(a)$ as estimate of value of action $a$ at time $t$.

The average of all rewards received when action $a$ was chosen.

Online update:

($\delta$-rule)

$$Q_{t+1}(a) = Q_t(a) + \gamma \left( r_{t+1}(a) - Q_t(a) \right)$$

$r_{t+1}(a)$ reward received after taking action $a$ at time $t+1$.

$Q_{t+1}(a) \text{ converges to } p(r_{1|a}) \text{ as time increases.}$
(3) To extend to full reinforcement learning:

(1) Several states. Several slot machines with different reward probabilities $P(r | s_i, a_j)$ and need to learn $Q(s_i, a_j)$ value of taking action $a_j$ when in state $s_i$.

(2) Actions affect not only the reward but also the next state, and we move from one state to another.

(3) The rewards are delayed and we must estimate immediate values from delayed rewards.
(4) Discrete time, \( t = 0, 1, \ldots \)
state \( s_t \in S \)
action \( a_t \in A(s_t) \)
reward \( r_{t+1} \in \mathbb{R} \)

Markov Decision Process (MDP)
reward and next state are sampled from \( p(r_{t+1}|s_t, a_t) \) and \( p(s_{t+1}|s_t, a_t) \).
Initial state/ often a terminal state (goal)
all actions in terminal state transition to it.

Episode, Traj — sequence of action from initial state to final state
Policy \( \pi \), defines the agent's behavior
\( \pi : S \rightarrow A \)
policy defines action \( a_t = \pi(s_t) \)
value of policy \( \pi \) \( V^\pi(s_t) \) is the expected cumulative reward for following the policy starting at \( s_t \).

Finite-horizon or episodic model, maximize the expected reward for next \( T \) steps
\[
V^\pi(s_t) = \mathbb{E} \left[ r_{t+1} + r_{t+2} + \cdots + r_{t+T} \right] = \mathbb{E} \left[ \sum_{i=t}^{T} r_i \right]
\]
Infinite-horizon
\[
V^\pi(s_t) = \mathbb{E} \left[ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \right] = \mathbb{E} \left[ \sum_{i=t}^{\infty} \gamma^{i-1} r_{t+i} \right]
\]
where \( 0 \leq \gamma < 1 \) discount rate.
For each policy $\pi$, we can compute $V^\pi(s_t)$ and we want to find the optimal policy $\pi^*$ such that

$$V^*(s_t) = \max_{\pi} V^\pi(s_t), \quad \forall s_t$$

In other applications, instead of working with the values of states $V(s_t)$, we work with values of state-action pairs $Q(s_t, a_t)$, i.e., how good is it to perform action $a_t$ in state $s_t$ and obeying optimal policy afterwards?

$Q^*(s_t, a_t)$ is the value, expected cumulative reward of action $a_t$ taken in state $s_t$.

Value of state is equal to the value of the best possible action:

$$V^*(s_t) = \max_{a_t} Q^*(s_t, a_t)$$

$$= \max_{a_t} E \left[ \sum_{i=1}^{\infty} \gamma^{i-1} r_{i+1} \right]$$

$$= \max_{a_t} E \left[ r_{t+1} + \gamma V^*(s_{t+1}) \right]$$

$$V^*(s_t) = \max_{a_t} \left( E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} | s_t, a_t) V^*(s_{t+1}) \right)$$

This is called Bellman's equation.
Similarly,

$$Q^*(s_t, a_t) = E[C(v_{t+1}) | s_t, a_t]$$

$$s_{t+1} \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

Once we have $Q^*(s_t, a_t)$, we define the policy $\pi$ as:

$\pi^*(s_t):$ Choose $a^*_t$

where $Q^*(s_t, a^*_t) = \max_{a_t} Q^*(s_t, a_t)$.

---

**Model-Based Learning**

If we completely know the environmental model parameters $P(r_{t+1} | s_t, a_t)$ and $P(s_{t+1} | s_t, a_t)$,

Then we can solve for the optimal value function and policy by dynamic programming.

The optimal value function is the solution to Bellman's equation:

$$V^*(s_t) = \arg \max_{a_t} \left( E[C(r_{t+1} | s_t, a_t)] + \gamma \sum_{s_{t+1}} P(s_{t+1} | s_t, a_t) V^*(s_{t+1}) \right)$$

This is a standard control/decision theory problem - reinforcement learning was invented by neural network researchers for the case when $P(s_{t+1} | s_t, a_t)$ and $P(r_{t+1} | s_t, a_t)$ are unknown.

The standard control/decision theory problem can be solved by the Value Iteration Algorithm.
Value Iteration.

To find the optimal policy, use the optimal value function.

Iterative algorithm - value iteration - converges to the correct $V^*$ value.

Initialise $V(s)$

For all $s \in S$, all $a \in A$:

\[ Q(s,a) \rightarrow \mathbb{E}[r(s,a) + \gamma \max_{s' \in S} P(s'|s,a)V(s')] \]

\[ V(s) \rightarrow \max_a Q(s,a) \]

Repeat until convergence

Converge if $\max_{s \in S} |V^{(k+1)}(s) - V^{(k)}(s)| < \epsilon$.

Convergence time $O(\frac{1}{\epsilon} |A|^2)$

but often $O(k |S| |A|)$ possible next.

Note: it is possible to compute $\mathbb{E}[r(s,a)]$ easily because $P(r_{t+1}|s_t,a_t)$ is known.

Similarly, we can compute $\sum_{s' \in S} P(s'|s,a)V(s')$ because $P(s'|s,a)$ is known.

If $P(r_{t+1}|s_t,a_t)$ and $P(s'_{t+1}|s_t,a_t)$ are unknown, then this is harder - see temporal difference learning later.
Temporal Difference Learning.

Most interesting realistic application of
reinforcement learning occur when we do not
know the model.

This require exploring the environment to
query the model.

Temporal Difference (TD) algorithm.

Use $\epsilon$-greedy search,
prob $< choose$ best action $- exploit$
prob $> choose$ one of all possible actions
$- explore$

Choose probability
$$p(a|s) = \frac{e^{\frac{Q(s,a)}{T}}}{\sum e^{\frac{Q(s,a)}{T}}}$$

where $Q(s,a)$ is your current estimate of
the $Q$-function.

Initially, $Q(s,a)$ is a bad estimate of the
ture $Q(s,a)$ so start with large $T$
(exploration strategy) and gradually reduce $T$
as we improve our estimate of $Q(s,a)$ and
move to an exploitation strategy.
Deterministic Rewards & Actions.

Special case: only one reward and one action possible.

Then Bellman's equation reduces to:

$$Q(s_t, a_t) = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

**Algorithm:** Explore by strategy above

When in state $s_t$,

choose $a_t$ by one of the stochastic strategies,

give reward $r_{t+1}$ and take $a_{t+1}$,

update $Q(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$

store $Q(s_t, a_t)$ in table.

Initially all $Q(s_t, a_t)$ are 0.

Suppose we only get a non-zero reward at the end of a sequence (i.e., play Backgammon - only get a reward from the action that wins the game).

In intermediate states, the rewards are 0 and so the $Q$'s stay at 0.

At goal state (e.g., win game), we get a reward for the previous state and previous action.

Next time we reach this 'previous state', we will assign a non-zero $Q$ to the state before this.

Eventually this will propagate backwards so that we get non-zero $Q$'s for almost all states and actions.

**Note:** this takes time and many sequences.

Gerry Tesauro trained a computer to play Backgammon by having the machine play a very large number of games against itself.
Non-deterministic Rewards and Action

\[ Q(s_t, a_t) = E[T_{t+1}] + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \]

This is the general case. We can't use the previous strategy because we can receive different rewards and go to different states.

We could explore the space and learn \( P(s_{t+1}|s_t, a_t) \) and \( P(r_{t+1}|s_t, a_t) \) — then solve the control/decision luxury problem. But this is not the best strategy. Instead learning \( \hat{Q}(s_t, a_t) \) with explicitly learning the \( P_s \)

\[ Q \text{-learning} \quad \hat{Q}(s_t, a_t) = Q(s_t, a_t) \]

TD update:  
\[ + \gamma (r_{t+1} + \max a_{t+1} \hat{Q}(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \]

Think of \( r_{t+1} + \max a_{t+1} \hat{Q}(s_{t+1}, a_{t+1}) \) as a sample of instances for each \( (s_t, a_t) \).

This algorithm is guaranteed to converge to the optimal \( Q \). (Proof: Watkins & Dayan)

Extensions:

- Many variants:
  - Apprenticeship learning (Abbeel et al. Stanford): Get an expert to give you a good policy, then improve this by reinforced learning.
  - Learning sequences of moves/actions: e.g., start with an action which move to a neighboring square only, then learn action that move several squares at the same time.