Markov process or Markov chain

> the state sequence satisfies the Markov assumption

- The current state depends on only a finite fixed number of previous states.
- $P(X_t|X_{0:t-1}) = P(X_t|X_{t-1})$

Decision Theory

- Choosing actions based on the desirability of their immediate outcomes.
- The outcome of taking action a in state s_0 is deterministic then $Result(s_0, a)$.
- If the outcome is non-deterministic (stochastic), fully observable, the agent knows the current state s_0.
  - It is represented as a random variable whose values are the possible outcome states.
  - The transition model specifies the probabilities of possbile outcome states.
  - $P(s, a, s')$

MEU, Maximum Expected Utility

> A rational agent should choose the action that maximizes its expcted utility.

- The MEU principle could be seen as defining all of AI.
- $action = argmax_a(EU(a|e))$
  - among this set of actions, give the highest expected utility.
- $EU(a|e) = \sum_{s'}(P(Result(a) = s'|a, e) * U(s'))$

Sequential decision problem

> type of problem or scenario where a decision-maker must make **a series of choices or decisions** over time in order to achieve a desired outcome.

- a sequential decision-making problem or sequential decision-making task

MDP, Markov Decision Process

> a sequential decision problem for a fully observable, stochastic environment environment with a Markovian transition model and additive rewards.

- a formal framework for modeling sequential decision problems.
- including states, actions, transition probabilities, and rewards
- used in reinforcement learning and operations research.

Component	Description
Environment	fully observable, Stochastic
States	a set of states (with an initial state S_0)
Transition model	Markovian
Reward	additive
Agent	Make decisions at time step for actions Interacts with environment through percepts and actions

- A set of states: an initial state S_0, possible terminal state(s)
- A set of actions in each state: $ACTIONS(s)$
- A stochastic Markov transition model: $P(s'|s, a)$
- A reward function: $R(s)$ or $R(s, a, s')$
- In MDP, Additive Reward is defined as a function that assigns a numerical value to each state-action pair or state-transition pair.
    - represents the immediate benefit or cost associated with taking a specific action in a particular state.
- The solution must specify what the agent should do for any state that the agent might reach.
- **Policy**: $\pi(s)$, an action for each state
    - A solution to an MDP.
    - $\pi(s)$ is the action recommended by the policy π for state s
- Q state: $Q(s, a)$, the expected utility of doing action a in state s

RL, Reinforcement Learning

- an agent interacts with an environment
- takes actions to maximize a cumulative reward signal over time.
- learns a policy that balances exploration and exploitation
    - exploration: trying new actions.
    - exploitation: choosing known good actions.

Sample MDP

Environment

- The agent lives in a 4x3 fully observable stochastic environment
- Walls block the agent's path; Begin in the start state, the agent must choose an action at each time step
- Actions available to the agent in each state: Up, Down, Left, or Right
- The interaction with the environment terminates when the agent reaches one of the goal states, marked +1 or -1

4x3 Grid World

	1	2	3	4
3				*+1*
2		🚫		*-1*
1	*START*			

- START: Initial position (1,1)
- 🚫 : Wall (impassable)
- +1: Goal state with positive reward
- -1: Goal state with negative reward

Transition Model

- Stochastic environment with uncertainty
- When agent chooses an action, there's:
  - 0.8 probability of moving in intended direction
  - 0.1 probability of moving perpendicular to intended direction (each side)

Reward Function

- The agent receives rewards each time step:
  - Small "living" reward each step (-0.04 in all states except the terminal states)
  - Big rewards come at the end (+1 for good and -1 for bad terminal states)

Goal

- To maximize sum of rewards from the start state to a terminate state

Additive Reward

- a type of reward structure
- the total reward earned by an agent in an MDP
- calculated by summing up the individual rewards received at each time step as the agent interacts with the environment.

Discounted reward

- a techinque used to calcluate the expected cumulative reward over time.
- applying a discount factor to future rewards, prioritizing more immediate rewards over distant ones.

Utility

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

- The expectation E is with repect to the probability distribution over state sequences determined by s and π.
- γ is the discount factor, $0 \leq \gamma < 1$, which determines the present value of future rewards.
- $R(S_t, \pi(S_t), S_{t+1})$ is the reward received when transitioning from state S_t to state S_{t+1} by taking action $\pi(S_t)$.

Bellman equation

$$U(s) = max_{a \in A(s)} \sum_{s'} P[s'|s,a](R(s,a,s') + \gamma U(s'))$$

Formula Component	Description
$U(s)$	the utility of state s

Formula Component	Description
max_a	the maximum over all possible actions $a \in ACTIONS(s)$
$\sum_{s'}$	the sum over all possible successor states s'
$P[s'\|s,a]$	the transition probability of reaching state s'
$R(s,a,s')$	the reward received after transitioning from state s to state s' by taking action a
γ	the discount factor, $0 \leq \gamma < 1$
$U(s')$	the utility of the successor state s'

- In optimal setting, the neighbor state meets the Bellman equation.
- From s_0 to termination, each step in the sequence should match the Bellman equation.

$$\pi_s^* = argmax_\pi U^\pi(s)$$

Q function

- the expected utility of taking a given action in a given state.
- enables the agent to act optimally simply by choosing
  - $U(s) = max_a Q(s,a)$
- The optimal policy can be extracted from the Q-function.
  - $\pi^*(s) = argmax_a Q(s,a)$

$$Q(s,a) = \sum_{s'} P[s'|s,a](R(s,a,s') + \gamma U(s'))$$
$$= \sum_{s'} P[s'|s,a](R(s,a,s') + \gamma max_{a'} Q(s',a'))$$

Reinforcement Learning

Key components of RL

- Agent
- Environment
- State
- Action

- Reward
- Policy

Application of RL

- Robotics
- Autonomous systems
- Game playing
  - Atari video games from raw visual input
  - Playing poker
  - Alpha Go
- Recommendation systems

RL Algorithms

- Model-based RL
  - often learns a utility function $U(s)$
  - the sum of rewards from state s onward from observing the effects of its actions on the environment.
- Model-free RL
  - Action-utility learning, such as Q-learning
  - Policy learning
- Passive RL
  - the agent's policy is fixed
  - the task is to learn the utilities of states.
- Active RL
  - the agent must figure out what to do through exploration and exploitation.
    - Exploration: trying new actions to discover their effects.
    - Exploitation: leveraging known actions that yield high rewards.

Direct Utility Estimation

- the utility of a state is **the expected total reward** from the state onward.
  - **the expecgted reward-to-go** from that state.
  - each trial provides a sample of this quantity for state visited.
- end of each sequence, the algorighm calculates the observed reward-to-go for each state and updates the estimated utility for the state accordingly.
- In the limit of infinitly many trials, the sample average will converge to the expectation in teh Bellman equation.

- Ignores the connections between states.
  - misses opportunities to learning, it learns nothing until the end of the trial.
  - often converges very slowly.

Adaptive Dynamic Programming, ADP

$$U_i(s) = \sum_{s'} P(s'|s, \pi_i(s))[R(s, \pi_i(s), s') + \gamma U_i(s')]$$

- advantage of the constraints among the utilities of states by learning the transition model
  - that connects them and solves the corresponding MDP
  - using dynamic programming to calculate the utilities of the states.
- These Bellman equations are linear when the policy π_i is fixed
  - can be solved using any linear algebra package.

Learning a transition model

- the environment is fully observable
- the agent can learn the mapping from a state-action pair (s, a) to the resulting state s'.
- The transition model $P(s'|s, a)$ is represented as a table and it is estimated directly from the counts that are accumulated in $N_{s'|s,a}$.
- The counts record how often state s' is reached when executing a in s.
  - $P(s'|s, a) = \frac{N_{s'|s,a}}{\sum_{s''} N_{s''|s,a}}$

Temporal difference, TD

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha[R(s, \pi(s), s') + \gamma U^\pi(s') - U^\pi(s)]$$

- when a transition occurs from state s to state s' via action $\pi(s)$
- α is the learning rate, $0 < \alpha \leq 1$
- no need a transition model to perform updates
- the environment is itself supplies the connection between neighboring states in the form of observed transitions.

Q learning

- **model-free** reinforcement learning algorithm
- the agent learns a Q-function, $Q(s, a)$
- avoids the need for a model by learning an action-utility function $Q(s, a)$ instead of utility function $U(s)$.

- a model-free Q-learning TD update
  - $Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma max_{a'} Q(s',a') - Q(s,a)]$
  - this update is calculated whenever action a is executed in state s leading to state s'
- TD Q-learning agent doesn't need a transition model $P(s'|s,a)$
  - either for learning or for action selection.