> "In life, good deeds often go unrewarded. However, with a large enough sample size, seemingly skewed outcomes will converge to an expected value, and things will eventually turn out in your favour."
>
> — *Kaguya-sama: Love is War*, Chapter 195
> Aka Akasaka

Markov Chains in AI: A Gentle Introduction

1. Introducing Markov Chains

Markov chains are mathematical models that describe a sequence of events where the probability of each event (state) depends only on the state attained in the previous event – this is known as the **Markov property**, or **memorylessness**. In simple terms, the future state depends only on the **present state** and not on how that state was reached. Formally, if s and s' are states, a first-order Markov chain satisfies:

[t(s, s') = P(s_{t+1} = s' \mid s_t = s),]

meaning the chance of transitioning to state *s′* at the next time step $t+1$ given the current state s_t is independent of past states before t. All such transition probabilities for a Markov chain can be organized into a **transition matrix** T, where each entry $T_{ij} = P(\text{next state} = j \mid \text{current state} = i)$. Each row of T sums to 1 (a stochastic matrix), reflecting that from any state i, the probabilities of transitioning to all possible next states j add up to 100%. The set of all possible states is called the **state space** (often denoted S).

To build intuition, consider a few relatable examples:

- **WhatsApp Replies:** Whether you reply to a message might depend mostly on the latest message you saw (e.g. if it's a question or not) and not the entire chat history. This can be seen as a Markov chain where the state is "last message type" and influences your response (next state).
- **Video Game AI:** An NPC's behavior might be modeled with states like "patrolling," "alerted," or "chasing." The choice of next action depends on its current state (if it's alert, it may transition to chasing with some probability).
- **Social Media Trends:** A topic on Twitter could be in states such as "dormant," "trending," or "viral." Each day it might stay trending or fade, depending only on its current status.
- **Dice Rolls and Traffic:** A fair die roll is memoryless by nature (the next roll is independent of the last), and traffic flow at an intersection can be modeled so that the next minute's congestion depends mainly on the current level (assuming sudden accidents or earlier conditions beyond the current state are negligible).

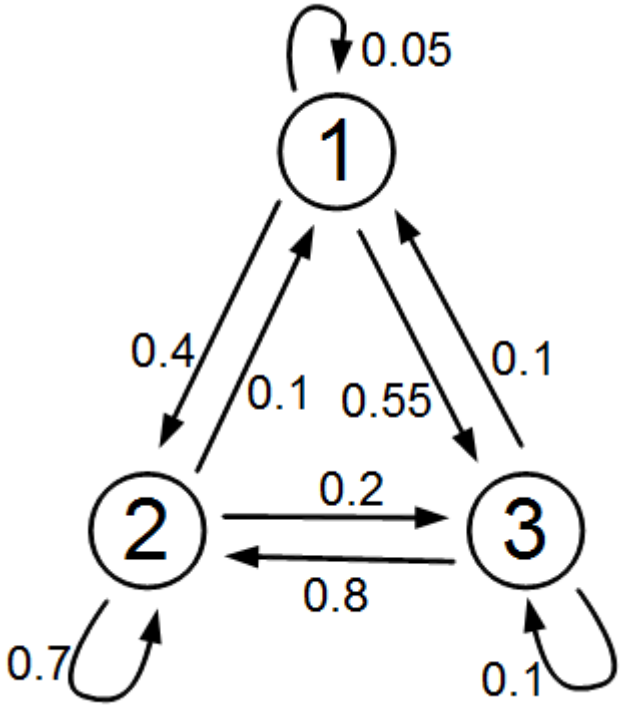

Figure: Example of a Markov chain shown as a state transition **diagram** with three states. Arrows indicate possible transitions between states, labeled by their transition probabilities. Such a diagram is an equivalent representation to a transition matrix, illustrating the Markov property: from each state (1, 2, or 3), the chain "hops" to the next state based only on the current node's outgoing probabilities, forgetting where it was earlier.

Source: Wikimedia Commons - Stochmatgraph

In all these cases, the process can be thought of as a sequence of states $s_0 \to s_1 \to s_2 \to \cdots$ where each transition $s_t \to s_{t+1}$ follows a fixed probability rule $P(s_{t+1}\mid s_t)$ given by the matrix T. Because only the present matters, Markov chains dramatically simplify modeling sequential phenomena. Instead of tracking the entire history, we just keep track of the **current state**. This makes analysis and computation feasible for AI systems that must predict or plan over sequences, as we will see next.

PROF

2. Designing Markov Chains

When designing a Markov chain model for an AI problem, the first step is to decide on a clear **state representation**. Each state should capture all the relevant information about the past that is needed to determine the future. If the environment is **fully observable** (the agent has access to the true state) and **stochastic** (there's randomness in transitions), we can often model its dynamics as a Markov chain. The transition probabilities can be estimated from data or defined from domain knowledge. Essentially, you ask: *"Given state X at time t, what are the probabilities of moving to each possible state at time t+1?"* Those probabilities form the entries of the transition matrix T. In a game of tic-tac-toe, for instance, a "state" might be the configuration of the board; if we were to randomize players' moves, we could form a Markov chain of game states. In a traffic system, a state could be the current traffic level (e.g. light, medium, heavy) and transitions model how traffic tends to build up or dissipate in the next hour.

Designing a Markov chain also involves understanding the **environment's structure**. In a fully observable environment, the agent knows exactly which state it's in, so it can directly use the Markov chain. In a partially observable setting, the agent might still **assume** an underlying Markov chain (this leads to Hidden Markov Models, touched on later). The environment's behavior can be **deterministic or stochastic**. If it's deterministic (the next state is fixed given the current state), the transition probabilities are 0 or 1; if it's stochastic, we have a distribution over next states. AI modelers often design Markov chains to approximate complex processes. For example, a simplified turn-based game like *Dungeons & Dragons (D&D)* might have states representing the turn order or phase of the game (exploration, battle, etc.), and probabilities capturing the likelihood of moving from one phase to another due to player actions or dice rolls.

In game theory and multi-agent settings, Markov chains can model evolving **game states or player strategies**. Consider a competitive video game with two players: one could model the state of the match (who is winning, special conditions active, etc.) as a Markov chain, where transitions occur based on the combined strategies and randomness. Even though players choose actions, if we fix a policy for each player or use a mixed (probabilistic) strategy, the *resulting state evolution* can be seen as a Markov chain. For instance, in a simplified rock-paper-scissors tournament, the state might be the score difference, and given the current difference, the match might swing to a win, loss, or tie with certain probabilities (assuming certain strategy patterns). Markov chains are useful in such scenarios to analyze long-run behavior or the likelihood of states (like eventually winning the game from a given situation).

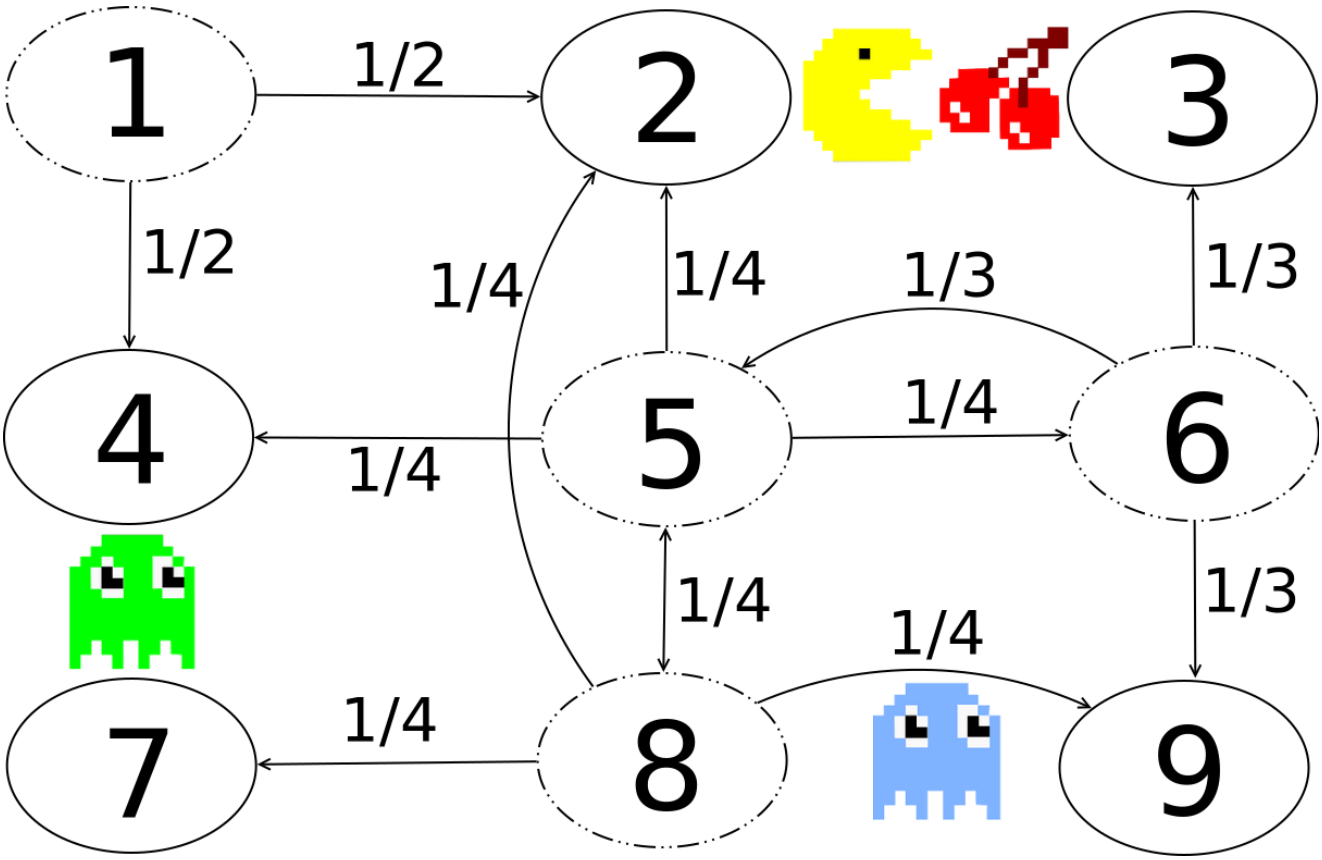

Figure: State transition diagram for a video game (Pac-Man as an example). Each oval is a game state (numbered regions in a Pac-Man grid) and each arrow shows a possible state transition with a probability. For example, from state 5 the game might move to state 6 with 1/3 probability, or back to state 8 with 1/4, etc., based on Pac-Man's movement and random ghost behavior. Such diagrams help in **designing Markov chains** by visualizing how an agent (like Pac-Man) can move through the state space. In designing your Markov model, you'd ensure that from any given state (node), the outgoing arrows

PROF

and their probabilities accurately reflect the environment's rules. By carefully choosing states and transition probabilities, we create a Markov chain model that captures the essential dynamics of the system or game.

Source: Wikimedia Commons - Transition graph pac-man

3. Computing Markov Chains

Once we have a Markov chain defined (state space and transition matrix T), we can perform various computations to understand its behavior. A central concept is the **steady-state distribution** (also called the *stationary distribution* π). This is a probability distribution over states that remains *unchanged* by the transition process – in other words, π satisfies $\pi T = \pi$. If you start the chain in the steady-state distribution, it will stay in that distribution at all times. Intuitively, as the chain runs for a long time, it "forgets" its initial state and the fraction of time it spends in each state converges to the steady-state probabilities. For example, if a particular state has a steady-state probability of 0.25, then in the long run the chain will be in that state about 25% of the time (on average). Finding π involves solving a system of linear equations (one for each state), or by computing T^n for large n and looking at where the rows (or columns) stabilize.

To compute **n-step transition** probabilities (the probability of going from state i to state j in exactly n steps), we can take the matrix power T^n. The (i,j) entry of T^n gives $P(s_n = j \mid s_0 = i)$. This is useful for questions like "If the system is in state A now, what's the probability it will be in state B after 5 transitions?" – you'd look at $(T^5)_{AB}$. Computationally, one can do this by repeated matrix multiplication or using algorithms like **power iteration** which repeatedly multiplies an initial distribution by T (simulating the chain) until convergence. Power iteration is a simple way to approximate the steady-state: start with any guess $\pi^{(0)}$ and compute $\pi^{(k+1)} = \pi^{(k)} T$ at each step; as k grows, $\pi^{(k)}$ will approach the steady-state π for an *ergodic* chain.

Ergodicity is a property that ensures a Markov chain will converge to a unique steady-state distribution. An ergodic chain is one that is both **irreducible** (it's possible to eventually reach any state from any other state, at least in theory) and **aperiodic** (it doesn't get caught in cycles with a fixed period). If these conditions hold, T^n as $n \to \infty$ approaches a matrix where each row is the steady-state π. Many real-world processes have this tendency of "forgetting" initial conditions if they run long enough and if no state is absorbing. For example, consider web page surfing as a Markov chain: given some reasonable randomness (the basis of Google's PageRank algorithm), eventually the probability of being on any given page stabilizes (that's a steady-state).

PROF

Other computations of interest include **expected hitting times** (e.g., "how many steps on average to eventually reach state X starting from state Y?") and **absorption probabilities** in chains that have absorbing states (states that once entered cannot be left). These can often be derived from T^n or by solving systems of linear equations. In summary, by leveraging linear algebra (matrix powers, eigenvalues, eigenvectors) we can analyze Markov chains quantitatively. For instance, steady-state probabilities correspond to an eigenvector of T (specifically, the left eigenvector with eigenvalue 1). Tools like eigenvalue decomposition give insight into the chain's long-term behavior and how fast it converges to steady-state (the second-largest eigenvalue magnitude relates to the convergence rate).

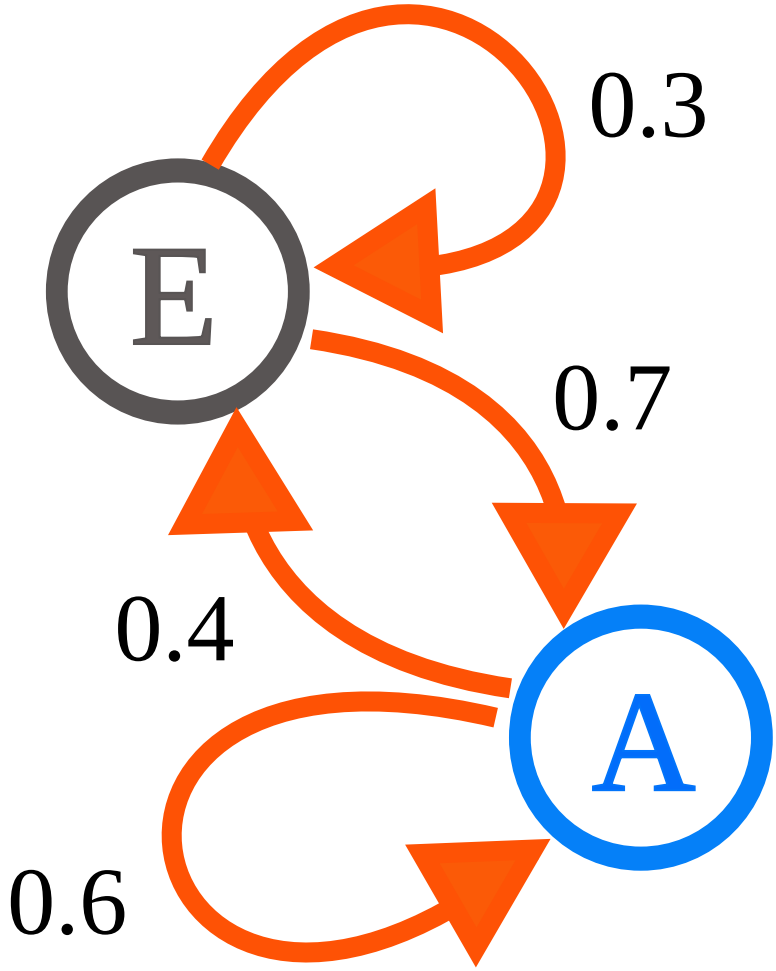

Figure: A simple two-state Markov chain (State A and State E). Arrows indicate the probability of transitioning from one state to the other (e.g., the arrow from A to E might have probability p, and from A to A is $1-p$). In such a chain, it's easy to compute the steady-state by hand – it will be the ratio of the opposing transition probabilities. For example, if $P(A \to E) = P(E \to A) = 0.5$, the steady-state distribution is 50% A and 50% E. In more complex chains, we use algorithms to find this equilibrium.

Source: Wikipedia - Markov chain

PROF

4. Inference and AI Agents

Markov chains provide a foundation for how AI agents *infer* the dynamics of their environment and make decisions. If an AI agent assumes the world follows the Markov property, it can simplify its decision-making by focusing on the current state. Different types of agents use Markov chain reasoning in various ways:

- **Simple Reflex Agents:** These agents choose actions based only on the current state, ignoring history. This aligns perfectly with the Markov property. If the agent's rule says "if in state S, do action A," it's effectively leveraging the transition probability $t(S, \cdot)$ (even if implicitly) to react. For example, a thermostat is a simple reflex agent – it doesn't care whether it was hot an

hour ago, only whether it's hot *now* (current state) to decide turning the AC on. We can model the room temperature as a Markov chain where the next temperature depends only on the current temperature and the AC/heater actions.

- **Model-Based Reflex Agents:** These have an internal **model** of the world's transition dynamics (a model of $t(s,s')$). In other words, the agent builds or is given a Markov chain of the environment. It can predict next states: "If I'm in state S now, my model says there's a 70% chance I go to S' and 30% chance to S'' after one step." This is essentially using the transition matrix T for inference. For instance, a game AI might know the Markovian rules of the game (like how opponents usually behave from the current game state) and can update its beliefs about what will happen one turn ahead.
- **Goal-Based Agents:** These agents consider future sequences of states to achieve a goal. They can use Markov chain properties to plan by looking ahead multiple steps (often this is done with search or planning algorithms). If the agent knows the transition matrix T, it can compute T^n or simulate many transitions to see where it might end up. Goal-based reasoning often involves finding a sequence of transitions (a path through the Markov chain) that leads to a **goal state**. In Markov terms, the agent might be interested in the probability of eventually reaching a goal state (a hitting probability) or the expected number of steps to reach it from the current state. By examining the chain (sometimes using algorithms like dynamic programming if rewards are involved), the agent can choose actions that increase the likelihood of reaching the goal. In a path-planning AI for navigation, for example, the map can be viewed as a Markov chain of locations, and the goal-based agent searches for a high-probability path to the destination.
- **Utility-Based Agents:** These agents not only have goals but also a utility (value) function to evaluate how desirable each state is. When an environment is modeled as a Markov chain, a utility-based agent will combine the transition probabilities with the utilities of future states to decide the best action. Essentially, it performs a kind of *expectimax* or expectated utility calculation: it considers "if I am in state S, and I take action A, the next state could be S' or S'' with certain probabilities (from the Markov model); how much utility will I likely get?" The agent will prefer the action that leads to the highest expected utility. In practice, this often leads into the realm of **Markov Decision Processes (MDPs)** – which are Markov chains extended with actions and rewards. The utility-based agent, via algorithms like value iteration or policy iteration, calculates the optimal policy on an underlying Markov chain of states.

PROF

- **Learning Agents:** These agents improve over time by learning from experience. In the context of Markov chains, a learning agent might start with unknown transition probabilities and then **learn** them by observing the frequencies of state transitions (essentially doing empirical estimation of $t(s,s')$). This is exactly what happens in **reinforcement learning**, where an agent explores an environment and estimates the state transition model and/or the value of states. By assuming the environment is Markovian, the learning algorithms (like Q-learning or Monte Carlo simulation) update estimates based on the current state and the observed next state, gradually converging to an accurate model of the Markov chain. Learning agents connect game theory and Markov chains when, for example, they adjust their strategy in a game (the transition probabilities might change as the opponent learns too – a more complex scenario!). Nonetheless, many AI learning scenarios use the Markov assumption as a backbone; even if the agent doesn't explicitly learn "the matrix," it often learns a policy that is optimal for the Markov chain dynamics of the environment.

It's worth noting how Markov chains are the stepping stone to more advanced models in AI:

- A **Hidden Markov Model (HMM)** is essentially a Markov chain where the states are not directly observed (they're "hidden") and we only see some observations emitted from states. The underlying chain might model, say, the emotional state of a user (happy, neutral, sad) which is not directly observable, while the observations are their messages or actions. The AI must infer the hidden state distribution using algorithms like forward-backward – all built on the Markov assumption for state transitions.
- A **Markov Decision Process (MDP)**, as mentioned, is a Markov chain augmented with actions and rewards. In an MDP, when an agent takes a given action in state s, the state transitions according to $P(s_{t+1}=s' \mid s_t=s, a_t=a)$. This still satisfies the Markov property (next state depends only on current state and action), and forms the core of how we model decision-making problems for AI. Solving an MDP (via dynamic programming, reinforcement learning, etc.) essentially means finding a good policy assuming the environment's transitions are a Markov chain influenced by actions.
- A **Partially Observable MDP (POMDP)** generalizes HMMs and MDPs – here the agent doesn't directly observe the true state, so it keeps a **belief** (a probability distribution over states). The belief itself can be seen as a state in an *even larger* Markov process! Markov chains thus even underlie the belief-state dynamics: given the current belief (distribution over where we might be), and an action and observation, we update to a new belief (this belief update can be seen as a transition in a belief-state Markov chain).

In summary, Markov chains provide the reasoning framework for AI agents to predict and plan. They allow agents to **simulate the environment** in their mind, reason about "if I'm in state X now, what's likely next?" and make informed decisions. From reflexive actions to strategic planning, assuming a Markovian world simplifies the complexity. AI researchers and engineers often start with this assumption because it's mathematically tractable and often a reasonable approximation of many domains.

PROF

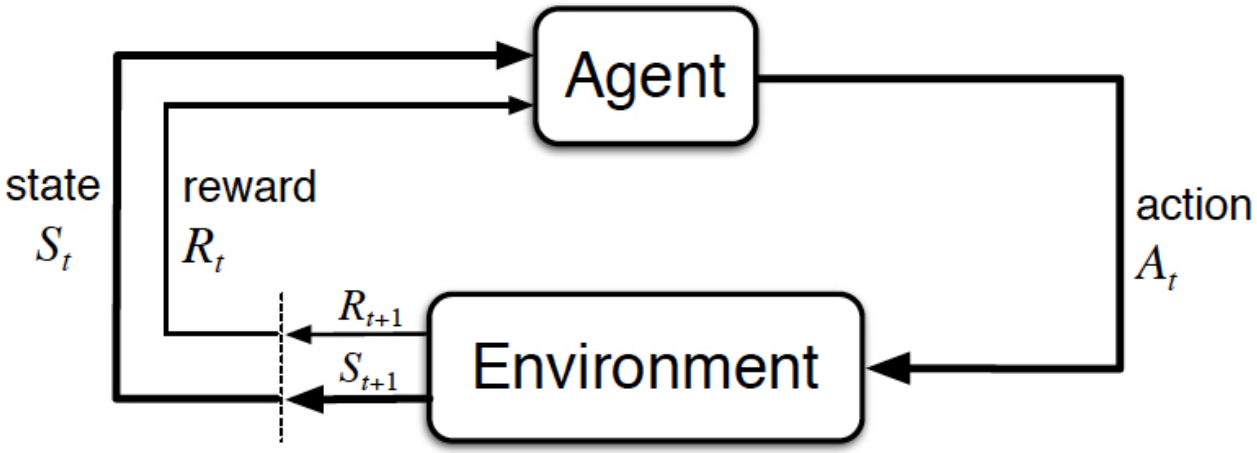

Figure: An illustration of an AI agent interacting with its environment, which can be modeled as a Markov process. The agent perceives the current **state** S_t and takes an **action** A_t; the environment then transitions to a new **state** S_{t+1} (following the Markov chain dynamics) and provides a **reward** R_{t+1} (in purely Markov chains without actions, you can ignore the action and reward). This loop (often depicted as in the figure) highlights the Markov property: the environment's transition to S_{t+1} depends only on the state S_t and the agent's action, not on any earlier history.

Source: Oscar Bastardo - Markov Decision Process

5. Wrap-Up & Next Steps

Markov chains are a fundamental concept in AI and many other fields because they provide a simple yet powerful way to model sequential processes. In this introductory chapter, we defined Markov chains, topics such as **Hidden Markov Models (HMMs)** for sequence data in AI (like speech or text), **Markov Decision Processes (MDPs)** for optimal decision-making in uncertain environments, and **Partially Observable MDPs (POMDPs)** for dealing with uncertainty in perception. All these advanced topics build on the ideas introduced here: states, transitions, and the memoryless property.

To reinforce these concepts, let's consider a creative real-world example of a Markov chain in action. Imagine the trending topics on a social media platform (like Twitter or TikTok). We can define states for a given topic such as: **Dormant** (hardly anyone is talking about it), **Trending** (it's gaining popularity rapidly), or **Viral** (it's extremely popular and widespread). On each day, the topic transitions between these states with certain probabilities. Perhaps if a topic is Trending today, there's a 50% chance it stays Trending tomorrow, a 40% chance it goes Viral, and a 10% chance it falls off to Dormant if interest dies down. This can be captured in a transition matrix. Over time, this forms a Markov chain of topic popularity. We could use it to answer questions like "What's the long-term fraction of time a typical topic spends in Viral state?" or "What's the probability that a currently Viral meme will be Dormant in two days?" – all using the computations we discussed (by looking at T^n or solving for steady-state). This social network dynamics example shows how Markov chains can model everyday phenomena in AI – in this case, the behavior of users collectively driving topics between popularity states.

As you move forward, keep in mind how the **Markov property** simplifies learning and inference. In more complex models (HMMs, MDPs, etc.), we often assume an underlying Markov chain because it makes the mathematics workable and often approximates reality well. In the coming chapters, we will delve into those topics: for instance, we'll see how an HMM uses a Markov chain to model sequences with hidden states, and how solving an MDP is essentially finding an optimal policy on a Markov chain of states when actions are involved. By mastering Markov chains, you'll be well-equipped to understand and build upon these advanced AI techniques.

PROF

Markov chains demonstrate that sometimes **memorylessness is more than enough** – by focusing on "now", we can predict the future and make intelligent decisions without drowning in past data. Now that you've been introduced to Markov chains, you're ready to explore the rich landscape of sequential models in AI that build on this concept. Happy learning as you step into HMMs, MDPs, POMDPs, and beyond, armed with the knowledge of Markov chains!