> "When you think you've figured out the probabilities, but the next observation sends your entire model spiraling into chaos:
> 'What... Huh?! Ain't no way!'"
>
> — Miyuki Shirogane, *Kaguya-sama: Love is War*

Hidden Markov Models (HMMs) – Extending Markov Chains with Hidden States

1. Introducing Hidden Markov Models (HMMs)

A **Hidden Markov Model (HMM)** is a statistical model where the system being modeled is assumed to be a Markov process with unobservable (hidden) states. While we cannot directly observe the hidden states, we can observe some outputs that depend on those states.

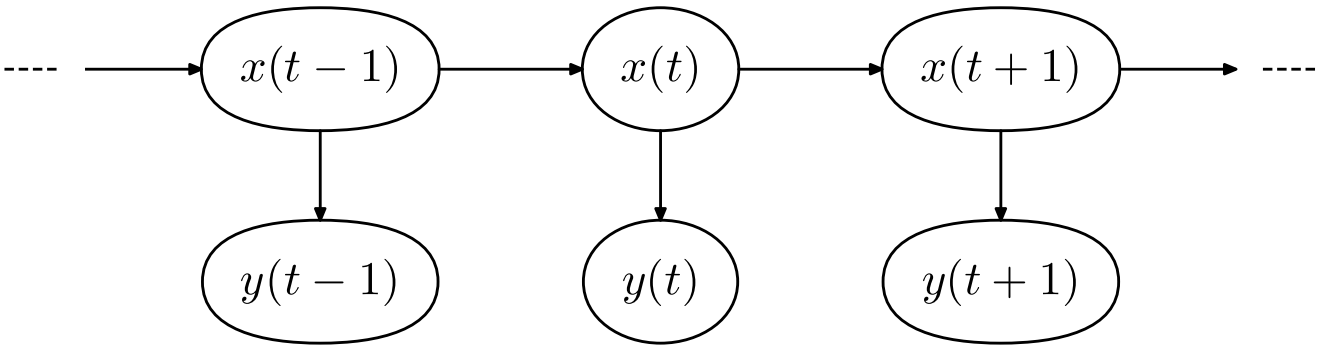


Figure: Hidden Markov Model represented as a Bayesian network with time slices. The hidden states x(t) follow the Markov property, while the observations y(t) depend only on the current state.

Source: Wikimedia Commons - Hidden Markov Model

Key Components

PROF

1. **Hidden States**: The sequence of states that the system can be in ($x_1, x_2, ..., x_n$)
2. **Observations**: The visible outputs we can measure ($y_1, y_2, ..., y_n$)
3. **Transition Probabilities**: The probability of moving from one hidden state to another
4. **Emission Probabilities**: The probability of observing a particular output given a hidden state

Application: Speech Recognition

One of the most successful applications of HMMs is in speech recognition. In this context:

- **Hidden States**: Represent the underlying phonemes or speech units
- **Observations**: The actual audio signal features (e.g., spectral properties)
- **Transition Probabilities**: The likelihood of one phoneme following another
- **Emission Probabilities**: The likelihood of observing certain audio features given a specific phoneme

For example, when someone says the word "hello", we observe the audio waveform, but the actual phonemes (/h/, /ə/, /l/, /oʊ/) are hidden states that we want to infer.

The Three Basic Problems of HMMs

1. **Evaluation Problem**: Given an HMM and a sequence of observations, what is the probability that the observations were generated by the model? (Solved using the Forward algorithm)

2. **Decoding Problem**: Given an HMM and a sequence of observations, what is the most likely sequence of hidden states that produced these observations? (Solved using the Viterbi algorithm)

3. **Learning Problem**: Given a sequence of observations, what should the HMM parameters be to best explain these observations? (Solved using the Baum-Welch algorithm)

Example: Weather Prediction

Imagine we want to predict weather patterns but can only observe if people are carrying umbrellas:

- **Hidden States**: The actual weather (Sunny, Rainy)
- **Observations**: Whether people carry umbrellas (Yes, No)
- **Transition Probabilities**: How likely is it for a rainy day to be followed by a sunny day?
- **Emission Probabilities**: How likely are people to carry umbrellas on rainy vs sunny days?

This simple example illustrates how HMMs can help us infer hidden states (weather) from observable data (umbrella usage).

To build intuition, consider a fun scenario from anime: *Kaguya-sama: Love is War*. In this story, each character's true feelings (like *"Does Kaguya secretly love Miyuki?"*) are hidden states – they keep those internal states secret. What we observe are their actions and words, which are often misleading "signals."

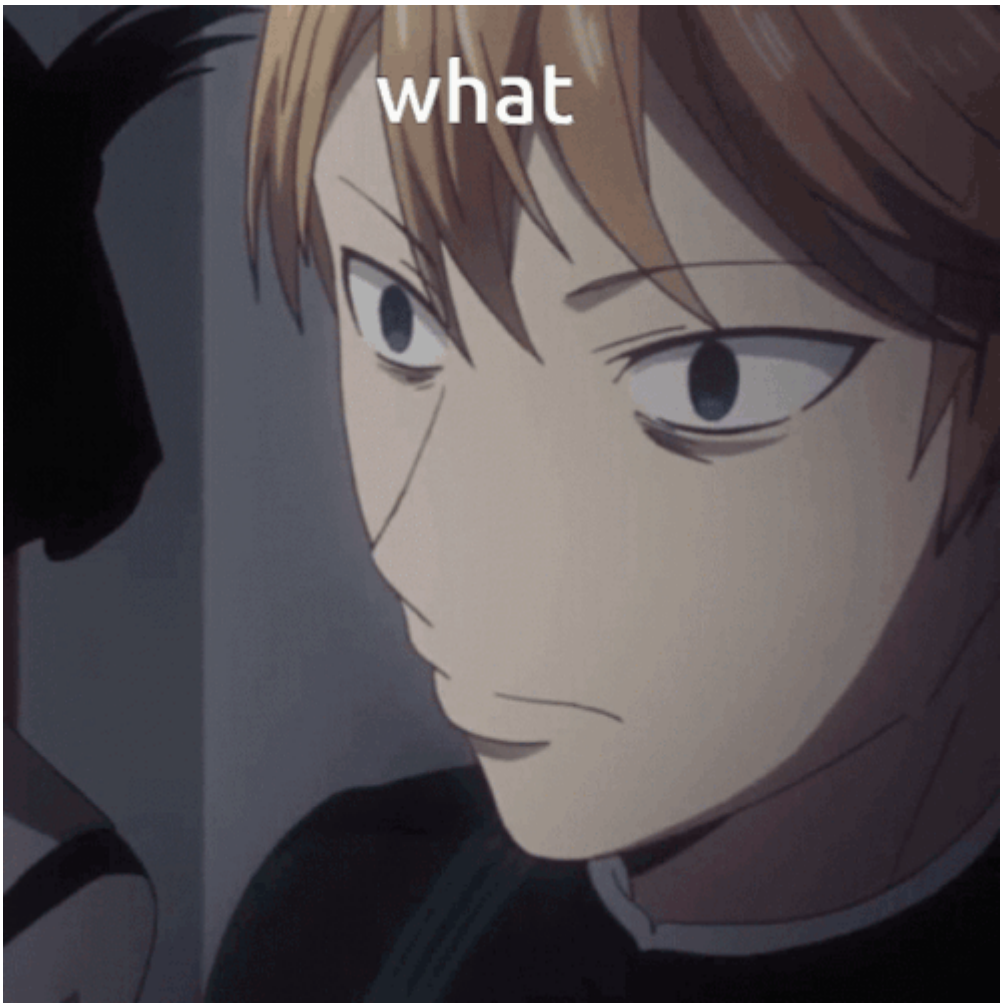


Figure: President Shirogane's reaction when his probabilistic model of Kaguya's feelings completely breaks down - a perfect representation of when our HMM's predictions are confronted with unexpected observations!

If Kaguya teases Miyuki or blushes, those are observations from which we try to infer her hidden emotion state (love or not love). **HMM intuition**: we have a sequence of observable actions (dialogue, body language) generated by a sequence of unobserved emotional states. As viewers (or AI observers), we maintain a **belief** (b(s)) about the state – essentially a probability for each possible state given the observations so far. This belief is updated as new observations come in. (In HMM terms, (b(s_t) = P(s_t = \text{"in love"} \mid \text{all observations up to time } t)), for example.)

PROF

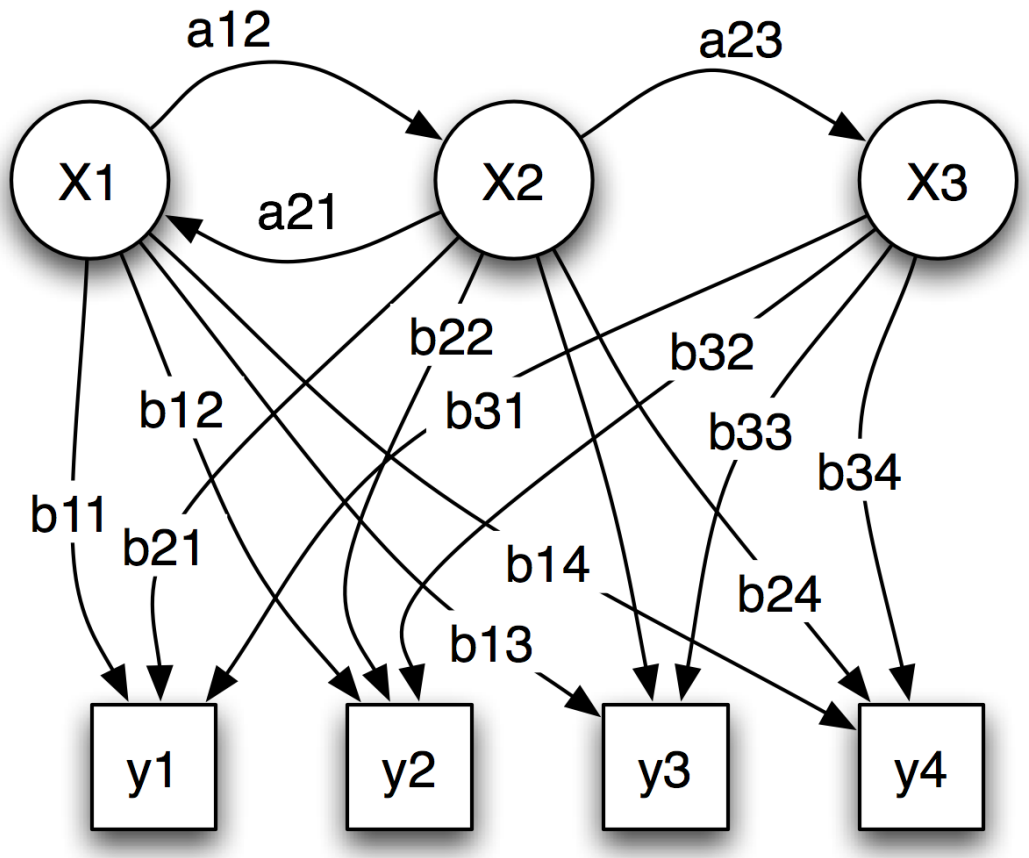
Another relatable example is interpreting a friend's mood from their WhatsApp messages. You can't directly read their mind (hidden state: happy, sad, etc.), but you see their texting style, emoji usage, and response time (observations). Over a chat conversation (a sequence of observations), you infer a likely sequence of moods. Perhaps short replies with "..." and no emojis suggest your friend is in a grumpy state; a sudden "LOL" indicates a transition to a happier state. Here you are **filtering** observations through an HMM in your head – using the *observable signals* to guess the *hidden context*.

Formally, an HMM is often described as a doubly stochastic process: a hidden **state process** that evolves by Markov transitions, and an **observable process** that produces emissions from each state (Hidden Markov Models and their Applications in Biological Sequence Analysis - PMC). We usually denote an HMM as (\lambda = (S, O, T, E)) where:

- (S) is the set of hidden states,

- (O) is the set of observations,
- (T = {t(s,s')}) is the state transition probability matrix (like a Markov chain's transition matrix),
- (E = {e(o|s)}) is the set of emission probabilities for observations.

Each time step, the HMM transitions from state (s) to a new state (s') according to (t(s,s')), then generates an observation (o) with probability (e(o|s')). The **Markov property** still holds for the hidden state sequence: the next state depends only on the current state, not the full history (Hidden Markov model - Wikipedia). However, unlike a visible Markov chain, we as observers only see the observations (o_t), not the states (s_t).



PROF

Figure 1: Illustration of an HMM's state-observation structure. Hidden states (X_1, X_2, X_3) (circles) have transitions between them (curved arrows with labels (a_{ij})), similar to a Markov chain. Each state can emit observable symbols (y_1, y_2, y_3, y_4) (squares) with certain probabilities (b_{ij}). We cannot see the (X) states directly – we only see the sequence of (y) outputs.

Source: Hidden Markov Model - Wikipedia

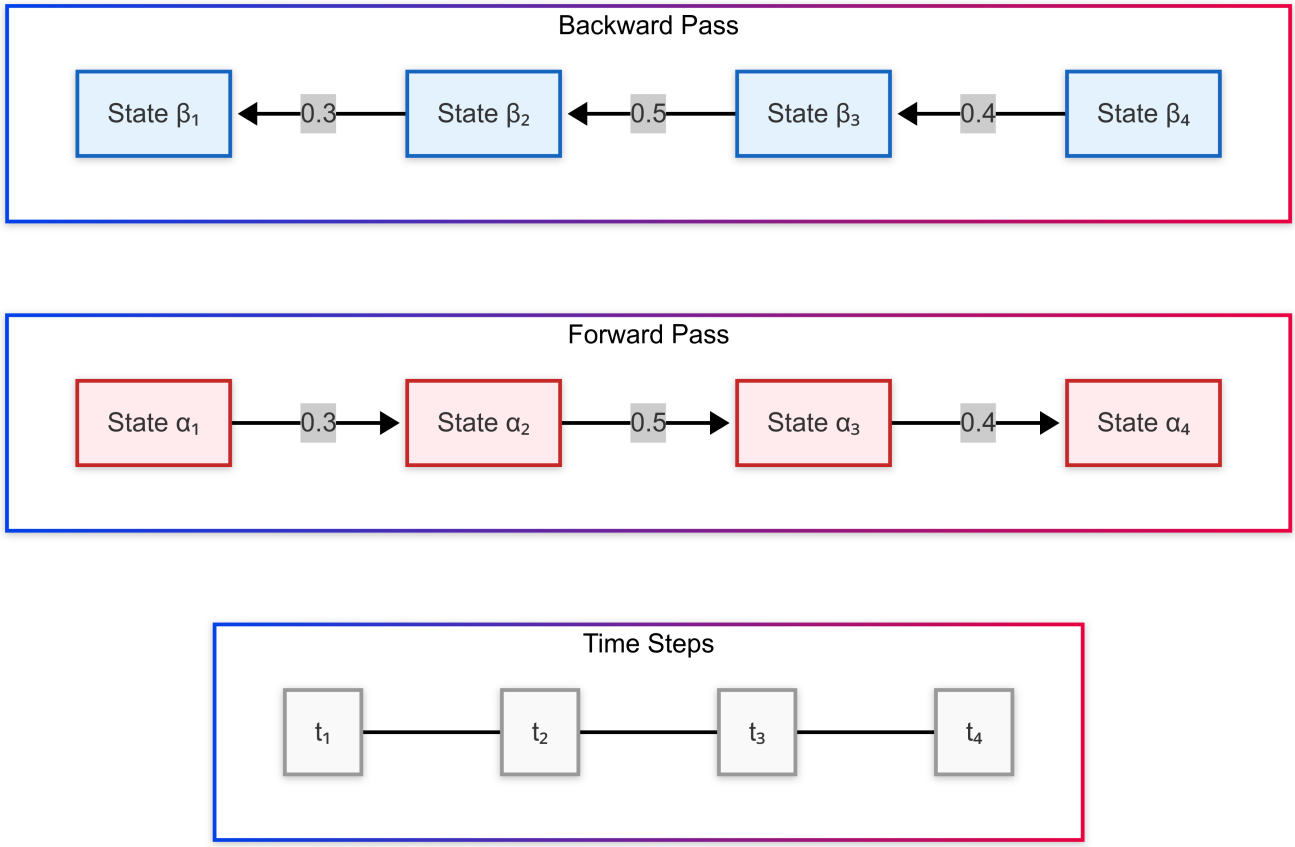


Figure 2: The Forward-Backward algorithm in action. The forward pass (α) computes probabilities from start to current time, while the backward pass (β) computes from end to current time. This allows us to combine information from both past and future observations when inferring the state at any time t.

Source: Forward-Backward Algorithm - Wikipedia

In summary, an HMM is a **Markov model with hidden states** (7.3: Markov Chains and HMMS - From Example to Formalizing - Biology LibreTexts). It models situations where a system's true state evolves with Markovian dynamics, but an agent only receives *emissions* (sensor readings, symptoms, spoken words, etc.) that depend on those states. We'll next see how to **design** an HMM for a problem, and then how to **compute** useful quantities from an HMM.

PROF

2. Designing HMMs

Figure: Kaguya displaying observable behavior (drinking tea nervously) while her true emotional state remains hidden - a perfect illustration of HMM's hidden states and observable outputs.

PROF

Designing an HMM for a real-world problem means deciding what the hidden states and observations should be, and estimating the transition and emission probabilities from data. We start by identifying the hidden process we care about and the signals we can observe.

- **Hidden States ((S)):** These should meaningfully represent the underlying condition that changes over time. For example, in a speech recognition HMM, the hidden states might be phonemes or words being spoken at each moment (Hidden Markov Models and their Applications in Biological Sequence Analysis - PMC). In our Kaguya-sama analogy, the hidden state could be a character's true emotion or intention in a given scene. In a weather model, the hidden state is the actual weather (which we can't see directly if we're indoors). Choosing the right state representation is part of the art: states should be distinct enough that they produce different observations.

- **Observations ((O)):** These are the data we actually get to see. In speech, the observations could be short audio signal features at each time (the sound frequencies). In Kaguya-sama, the observations are the characters' outward actions or dialogue lines (which *hint* at their feelings). For a chatbot analyzing WhatsApp messages, the observations might be the text sentiment or number of emojis in each message. The observation space can be discrete symbols or continuous values. We often discretize or categorize observations to fit an HMM (e.g., "message tone is positive/negative/neutral" as discrete observation categories).

- **Transition Probabilities ((T)):** We need to define how likely the hidden state is to persist or change. This is a *state transition model*. For example, if we model daily mood as states, (t(s,s')) might encode that "if someone is happy today, there's an 80% chance they stay happy tomorrow, 20% chance they become neutral, and very low chance to jump directly to sad." These probabilities can be estimated from labeled data if available (how often do we see state *s* followed by state *s′* in training sequences?). In design, you might start with a guess or domain knowledge. If data is untagged (states not labeled), we will learn these parameters via algorithms like Baum-Welch (discussed later).

- **Emission Probabilities ((E)):** This is the *observation model*: for each state, how likely is each observation. For instance, if hidden state is "Kaguya is secretly in love," what's the probability she will make a teasing remark (observation1) versus give a compliment (observation2)? Or, if the state is "rainy", how likely are we to observe "person has umbrella" versus "no umbrella"? These probabilities (e(o|s)) can be estimated from data by seeing how often each observation happens when the system is in state *s*. If we have paired state-observation data, we simply count frequencies. If not, we treat them as unknowns to learn with training algorithms.

Building an HMM means **modeling a partially observable, stochastic environment**. "Partially observable" because the agent cannot directly see the state (); "stochastic" because state transitions and observations are probabilistic, not deterministic. We thus embrace uncertainty in both how the state evolves and how observations are generated. This is powerful for AI because many real-world processes have hidden factors and randomness.

Let's connect this to some concrete scenarios:

PROF

- **Game Strategy Inference:** Consider a multiplayer video game or a board game. Your opponent's true strategy (aggressive, defensive, bluffing, etc.) is hidden. You see only their moves (observations). An AI agent could use an HMM where states represent the opponent's strategy or "type" (), and observations are their actions each turn. By modeling transition probabilities, the AI can infer if the opponent is likely to switch strategies mid-game. For example, in **poker**, an opponent's cards and whether they are bluffing are hidden states; their betting behaviors are observations. Over rounds, an HMM can help detect patterns (maybe a certain betting pattern suggests a bluff state) and update the AI's belief about the opponent's hand strength. This is essentially a simplified *game theory* application of HMMs: it's like a repeated game with incomplete information, which can be modeled by an HMM where the hidden state encodes the opponent's private information ().

- **Social Media Trend Analysis:** Imagine an AI monitoring Twitter (X) to detect trending topics. The actual *trend state* (what topic is currently "hot") is hidden in the noise of millions of posts. Observations might be the frequencies of certain keywords or hashtags. An HMM could have

states like "Topic A trend", "Topic B trend", etc., or more abstract states like "peak of trend vs. declining". Transition probabilities model how trends rise or fall (e.g., a trend often stays same for a while, or transitions from rising to falling). Emission probabilities model how likely certain hashtags or keywords are seen given a particular trend state. By learning from data, the AI can detect when a shift in state occurs (say, when tweets about #GameStop suddenly surge, transitioning into a "GameStop-hype" state). Essentially, the AI *learns hidden transitions*: it might detect that after a period of increasing mentions there is an X% chance a topic goes viral versus fizzles out. This example shows how HMMs can detect *shifts in hidden contexts* like public interest, even though we only directly see the noisy observations (tweets).

- **Emotional State Tracking:** (Returning to our WhatsApp example) Suppose we design an HMM to interpret a friend's emotional trajectory over a week of chats. We define hidden states like {Happy, Neutral, Stressed, Sad}, and observations derived from messages: maybe {positive emoji, neutral text, negative words, slow reply, etc.}. By analyzing past chat data (if we had it labeled with the friend's actual mood, or by self-report), we could estimate transition probabilities (maybe "Stressed" tends to persist 2-3 days then transition to "Sad" with 0.5 probability or back to "Neutral" with 0.5) and emission probabilities (when "Happy", 70% chance of positive emojis; when "Sad", 50% chance of slow replies, etc.). The result is an HMM that an AI chatbot could use to **infer your friend's current mood** as you continue chatting, even though it never observes the mood directly. This kind of model-based reflex is similar to how AI mental health assistants might track patient state from conversation cues (hidden clinical state inferred from observed speech/text).

When designing HMMs, **domain knowledge** is very useful for picking states and shaping the model. Sometimes the hidden states correspond to physical realities (like "raining" vs "not raining" in the umbrella example), but other times they are abstract (like "regime 1" vs "regime 2" in a financial market model (Hidden Markov Models - An Introduction | QuantStart)). We might not know exactly what each hidden state *means*, but if the HMM with 2 or 3 states fits the data well, it can still be useful for prediction.

It's also worth noting that HMMs assume the process is *memoryless* (Markov) and *stationary* (transition probabilities don't change over time). In reality, these assumptions might be approximate. For instance, a person's mood transitions might not be strictly Markov (yesterday's and the day before's mood could influence tomorrow, violating first-order Markov), or the probabilities might change seasonally. Nonetheless, HMMs often serve as a reasonable and simpler model that captures the essential dynamics. They have been **extremely popular in AI** fields like speech recognition, bioinformatics, and language processing because they balance mathematical rigor with the ability to handle hidden information (
Hidden Markov Models and their Applications in Biological Sequence Analysis - PMC
).

PROF

3. Computing HMMs

Figure: Hayasaka expertly analyzing situations - representing how we decode hidden states from a sequence of observations.

PROF

Once we have an HMM, there are three fundamental computational problems we usually want to solve:

1. **Likelihood:** Computing the probability of an observation sequence, (P(o_1, o_2, \dots, o_T)), given the model. In other words, how *likely* is it that our HMM would produce a certain sequence of observations? This is often called the **evaluation problem**. For example, given an HMM that models English sentences, what's the probability it generates the observed word sequence "I lost my phone"? This computation sums over all possible hidden state sequences that could produce the observations. Doing this by brute force is exponential (there are (N^T) state sequences if there are N states and T time steps), but there is a dynamic programming solution known as the **Forward Algorithm** (Forward algorithm - Wikipedia). The forward algorithm efficiently computes the likelihood (P(o_{1:T})) by iteratively aggregating probabilities of being in each state at time *t* and emitting (o_t). It essentially propagates a **belief state** forward in time (Forward algorithm - Wikipedia). By the end, we get (P(o_{1:T})). This is useful, for instance, if we want to compare

models or detect anomalies (an observation sequence with extremely low probability might indicate something unexpected, like in gesture recognition an impossible movement sequence).

2. **State Inference:** Determining the most likely hidden state(s) given the observations. This can mean computing the **belief distribution** (b(s_t) = P(s_t \mid o_1,\dots,o_t)) at time *t* (known as filtering), or the probability of being in each state at time *t* given *all* observations (smoothing), or finding the single most likely sequence of states (\arg\max_{s_{1:T}} P(s_{1:T} \mid o_{1:T})) (known as **decoding**). For example, an AI speech recognizer might continuously update its belief about which word is being spoken as sound comes in (filtering). Later, after hearing the whole sentence, it might revise its guess for earlier words (smoothing). The **Forward-Backward Algorithm** provides an efficient way to do smoothing by combining forward probabilities (from start up to t) and backward probabilities (from end down to t) for each time (Forward algorithm - Wikipedia). On the other hand, to get the single most likely hidden state path, we use the **Viterbi Algorithm** (Viterbi algorithm - Wikipedia). Viterbi is a dynamic programming algorithm that traces the best path of states that could produce the observations, essentially by keeping track of the highest probability path to each state at each time and then backtracking (Viterbi algorithm - Wikipedia). If the forward algorithm is like *summing over all rhythms that could produce a muffled song*, Viterbi is like *finding the single catchiest melody that fits the muffled notes*. In practical terms, if our HMM is tracking user intent from mouse movements, Viterbi could give the most likely sequence of intents (e.g., "select -> drag -> drop") behind an observed cursor path.

3. **Learning (Parameter Estimation):** Figuring out the best model parameters (the (t(s,s')) and (e(o|s)) probabilities, and possibly the state set itself) from data. This is the **training problem** for HMMs. If we have a set of observation sequences (and maybe some partial knowledge of states), how do we adjust the HMM to fit them? The classic solution is the **Baum–Welch algorithm**, which is a special case of the Expectation-Maximization (EM) algorithm for HMMs (Baum–Welch algorithm - Wikipedia). Baum–Welch starts with an initial guess of the parameters and then iteratively improves them: in the E-step it uses the current model to compute expected counts of various transitions and emissions (often using the forward-backward procedure to handle the hidden states probabilistically), and in the M-step it updates the probabilities to maximize the likelihood of the observations given those expected counts (Baum–Welch algorithm - Wikipedia). Over iterations, the model hopefully converges to a set of parameters that explain the training data well. For instance, if we are training an HMM for gesture recognition, Baum–Welch will adjust the transition probabilities between "gestures" and the emission probabilities of various touch events for each gesture, until the model best fits all the example gesture traces we have. One can think of Baum–Welch like *an AI trying to learn people's hidden emotions over time by observing their messages*: it starts with some random assumptions, then refines its beliefs about how states lead to words (and how states follow each other) each time it sees a new conversation, eventually converging to a pretty good guess of the patterns.

PROF

Let's make these concepts more concrete with a quick analogy per algorithm:

- **Forward Algorithm (Filtering):** Imagine you're listening to a song played in another room. You catch bits of the melody (observations) but some notes are muffled. You maintain a belief of what chord or note the song might be at now given what you've heard so far. As each new sound comes, you update this belief. This is like filtering: progressively updating (b(s)) as evidence accumulates (Forward algorithm - Wikipedia).

- **Viterbi Algorithm (Decoding):** Now think of trying to identify the exact song. You mentally consider all possible songs and find the one whose sequence of notes best matches the muffled sounds you heard. You output the most likely song (state sequence). That's akin to Viterbi: finding the single most likely hidden sequence explanation for the observations (Viterbi algorithm - Wikipedia). In an AI context, Viterbi might be used to decode the most likely sequence of words from speech audio – effectively what many speech-to-text systems did when they used HMMs (Viterbi algorithm - Wikipedia) (each word or phoneme is a hidden state, the audio features are observations).

- **Baum–Welch Algorithm (Learning):** Suppose you're trying to learn a friend's mood patterns without them telling you directly. Initially, you guess randomly how their mood transitions day to day and what words they use in each mood. Each day, you observe their messages and update your guess about the chances ("Hmm, they said 'I'm tired' – that's something I expected more in the Sad state, maybe the model should increase probability of Sad today"). Over weeks, you refine these probabilities to better predict their moods. Baum–Welch does this systematically for an HMM, using all observation data to adjust the model parameters toward higher likelihood (Baum–Welch algorithm - Wikipedia).

Mathematically, the core computations involve a lot of summing and multiplying probabilities, but the dynamic programming algorithms ensure it's efficient (typically O(N^2 * T) time, where N is number of states and T is sequence length, for algorithms like forward or Viterbi). HMM software implementations take care of under-the-hood details like scaling (to avoid underflow of tiny probabilities) (Baum–Welch algorithm - Wikipedia). The result is that even if you have, say, 5 hidden states and a sequence of 1000 observations, you can compute things in milliseconds rather than checking millions of state paths.

Pros and Cons: HMMs are relatively efficient and well-understood. They can model sequences in an interpretable way (you can often interpret the hidden states meaningfully). The dynamic programming solutions (Forward, Viterbi, etc.) guarantee optimal results for their respective problems (likelihood, best path) given the model. HMMs handle noise and uncertainty gracefully by using probabilities. However, they do have limitations: the Markov assumption (state only depends on last state) might be too simplistic for some tasks with long-range dependencies. Also, standard HMMs assume the probabilities don't change over time (stationarity), which might not hold if the process behavior shifts (for example, an HMM trained on Monday-Friday might not directly apply to weekend behavior if patterns differ). Another consideration is that if the observation space is very large or continuous, one might need to use more complex emission models (like Gaussian mixtures for continuous observations) which can make training trickier. Also, training via Baum–Welch (EM) can get stuck in local optima, so it might not find the absolute best model, just a good one for the data. Despite these, HMMs remain a foundational tool for sequential data in AI, and understanding them sets the stage for more complex models.

PROF

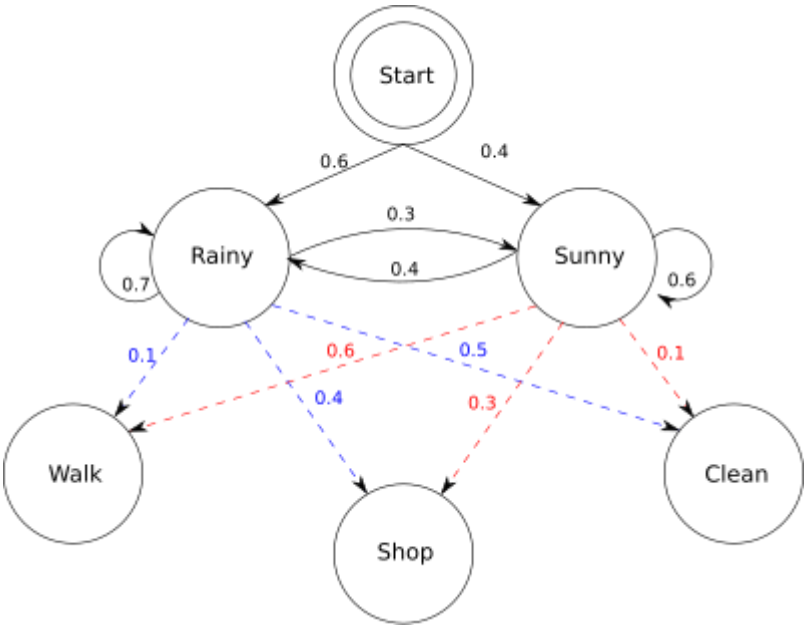


Figure 3: The Viterbi algorithm finds the most likely state sequence through a trellis of possibilities. Each node represents a state at a particular time, and edges show transitions. The algorithm efficiently computes the best path (highlighted) by keeping track of the maximum probability path to each state.

Source: Viterbi Algorithm - Wikimedia Commons

4. Inference and AI Agents

PROF

Figure: Chika in detective mode - representing how we evaluate and investigate patterns in observations to determine if they match our model's expectations.

Now that we understand HMMs and how to compute with them, let's see how **AI agents** can leverage HMM inference in decision making. In AI (as described in many textbooks), there are different types of agents: Simple Reflex, Model-Based, Goal-Based, Utility-Based, and Learning agents. HMMs can play a role in the more sophisticated ones that need to handle hidden state information. We'll discuss each in turn:

- **Simple Reflex Agents:** These agents choose actions based only on the current *observation*, with no internal state. They don't maintain an HMM or any model of the world – they simply do condition-action responses ("if see X, do Y"). Since HMMs are all about estimating hidden state, a

pure simple reflex agent doesn't use an HMM explicitly. For example, a spam filter that looks at an email (observation) and immediately decides "spam" or "not spam" is reflexive. However, if we stretch the definition, even a reflex agent can be improved by an HMM if we feed it an observation that is the *output of HMM inference*. For instance, a WhatsApp notification classifier might use the **filtered belief** from an HMM as part of the percept. Imagine a notification system that alerts you only if your friend's mood (hidden) is estimated as "urgent" by an HMM analyzing their message patterns; the agent's rule could be "IF (believe friend is upset) THEN alert user". In that case, the HMM is running in the background to provide a better percept to a basically reflex rule. But generally, simple reflex = no memory of past, so no need for HMM (which thrives on sequence info).

- **Model-Based Reflex Agents:** These maintain some internal state to handle partial observability. This is exactly where HMMs shine. A model-based agent has a model of how the world works (transitions) and how observations relate to state (emissions) (). That is essentially an HMM! The agent uses the model to **update its belief state** each time it receives an observation. For example, consider a **speech recognition AI** in a smart home device. The device doesn't directly know what word a person is saying (hidden state), it only hears audio signals (observations). It maintains a belief over possible words as sound comes in, using a model of speech (which could be an HMM or a more advanced variant). This is a model-based reflex agent: its decision (e.g., to execute a voice command) is based on the inferred state (which command was spoken) rather than the raw sound. The HMM here is continuously updating "what phoneme/word do I think they're saying?" (
  Hidden Markov Models and their Applications in Biological Sequence Analysis - PMC
). Another example: a robot navigating a building with blurry vision. It has an internal HMM for its location: states = possible locations, observations = what it sees through the blurry camera. The robot updates its belief of where it is as it moves and sees new observations (forward algorithm in action). Its actions (like "turn left") are chosen based on that belief (e.g., if it strongly believes it's in the hallway outside room 101, it will act accordingly). In summary, model-based agents **use HMM inference to handle hidden state**, making them much more effective in the real world than simple reflex agents.

PROF

- **Goal-Based Agents:** These agents not only track state but choose actions to achieve goals. In a partially observable setting, they will rely on the belief (from an HMM) to do **goal-directed planning**. For instance, consider an agent whose goal is to find a treasure in a maze, but it can't see the whole maze (only local observations). The agent's HMM might track a probability distribution over where the treasure could be (hidden state) based on clues it finds (observations). With a goal in mind ("find treasure"), the agent will plan actions that maximize the chance of reaching the treasure's location according to its belief. Another scenario: an autonomous drone has a goal to deliver a package to a target person in a crowd. The person's current location is hidden; the drone gets sensor observations (maybe partial facial recognition or color of clothing glimpses). The drone's internal HMM maintains a belief over where the person is in the crowd. A **goal-based strategy** uses this belief to plan search movements ("the person is likely to be on the east side of the market, I should fly there first"). In essence, the agent considers *belief states* as part of the search for a plan that achieves the goal. The HMM itself doesn't decide the goal, but it feeds into the decision-making by providing the best estimate of the current world state given the observations () ().

- **Utility-Based Agents:** These extend goal-based agents by not just having a binary goal, but a utility function that rates how good different outcomes are. In a partial observable context, the agent will use the belief (from the HMM) to calculate expected utilities of actions. For example, in a poker game, an AI agent might have a utility function corresponding to its monetary outcome. It doesn't know the opponents' cards (hidden state), but it has a belief distribution over their possible hands based on the betting observations so far. When deciding whether to bet or fold, the agent will consider the expected utility of betting – which depends on the probabilities (belief) that the opponent is bluffing or has a strong hand. If the belief is 70% the opponent is bluffing (weak hand) and 30% strong, and the utility of winning vs losing is weighed, the agent uses these to choose the action with highest expected payoff. This is where HMM meets decision theory: the HMM provides (b(s)) (like "70% weak, 30% strong"), and the utility-based agent combines that with the potential outcomes of actions to pick the best one. Essentially, **it weighs the HMM's belief** with the costs/benefits of actions. Another example could be a medical diagnosis agent that has a utility for correct treatment vs risks. The patient's true condition is hidden; the agent has beliefs from an HMM that monitors symptoms over time. A utility-based approach would consider treatments (actions) and use the belief to expect how effective or harmful each might be, then choose the treatment with the best expected utility (e.g., if it's 90% sure the patient has condition A, treat for A, but if unsure, maybe do a test action).

- **Learning Agents:** These agents can dynamically improve their performance by learning from experience. In terms of HMMs, a learning agent could **adjust the HMM's parameters (or structure) on the fly** as it gathers more data. Imagine an AI personal assistant that learns a user's routine (hidden states could be "at work", "at home", "commuting"; observations are phone sensor data like GPS or app usage). Initially, it might have a generic HMM for daily routine. As it observes the specific user's behavior, it learns that, say, this user often goes to a gym at 6pm (adjust transition probabilities to add a state or increase probability of "gym" at that time). This could be done by an online version of Baum-Welch or simple counting if states are known. Another learning aspect is if the agent encounters a new hidden state or observation it didn't know before – a learning agent might expand its HMM to accommodate this. For example, an AI speech assistant might adapt to a user's accent over time: it updates the emission probabilities in its HMM for certain phonemes as it learns the user pronounces "cat" with a distinct accent (adjust (e(o|s)) for those audio features). Learning could even involve refining the state definitions – e.g., splitting a state into two if observations suggest it's actually two distinct contexts. In summary, a learning agent uses feedback (maybe from the environment or a reward signal) to refine the HMM's (T) and (E) matrices, thereby improving its inference accuracy over time.

PROF

To connect this back to game theory, consider **bluff detection in poker** again. A utility-based learning agent could use an HMM to model an opponent's play style (hidden state could be tendency to bluff vs play straight). Initially, the agent might not know the opponent (so it starts with some generic model). As it plays multiple rounds (observing the opponent's bets and eventually the revealed cards at showdowns), it *learns* the opponent's hidden strategy distribution and transition – perhaps the opponent bluffs more often after losing a big hand (state transition triggered by that event). The agent updates its model (learning) and uses it to make better decisions next time (inference for action). Over many games, this essentially trains an HMM specific to that opponent, allowing the agent to anticipate bluffs better than a non-learning agent.

Beyond HMM to MDP/POMDP: So far, we assumed the agent's actions don't affect the state transitions of the HMM (we've considered either passive observation or one-way decisions). In an **MDP (Markov Decision Process)**, the agent's actions influence state transitions (it's like a controlled Markov chain). If the agent can fully observe the state, we have an MDP. If not, we get a **POMDP (Partially Observable Markov Decision Process)**, which is basically an MDP + an HMM for perception () (). In a POMDP, the agent maintains a belief (like (b(s)) from the HMM filtering) and chooses actions based on that belief. Our discussion of goal-based and utility-based agents under uncertainty is essentially a POMDP framework: the agent's *belief state* is updated via an HMM (model-based updating), and decisions are made to maximize expected utility or achieve goals. POMDPs are more complex to solve than HMM inference alone (because now we also have to consider future observations and long-term planning), but conceptually they show how HMMs fit into the larger picture of AI decision-making. Many practical systems approximate POMDP solutions by doing HMM state estimation + heuristic decisions, because exact POMDP solving is computationally intense.

To sum up this section, HMMs serve as the **"brain" of the perception component** in AI agents dealing with uncertainty. Whether it's a reflexive system augmenting its input or a full planning agent deciding actions, the ability to infer hidden state from observations is critical. HMMs provide a mathematically grounded, efficient way to do this inference. They transform raw sequences of sensor data into a meaningful state estimate that the decision logic of the agent can then use.

(For a visual depiction, one could imagine a diagram where an agent receives observations from the environment, feeds them into an HMM belief updater (a Bayesian inference module), and out comes an updated belief state which goes into the agent's decision module. The agent then takes an action which affects the environment, and the loop continues – essentially a POMDP agent architecture.)

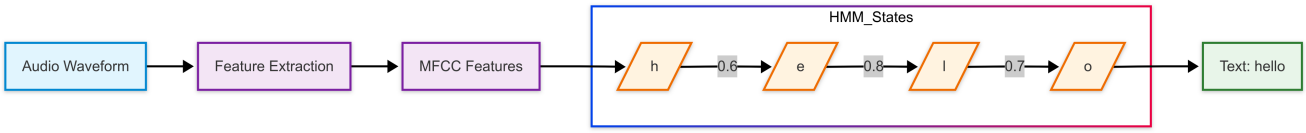


Figure 4: Application of HMMs in speech recognition. The diagram shows how audio waveforms are processed into feature vectors (observations), which are then used by the HMM to infer the most likely sequence of phonemes or words (hidden states).

PROF Source: Speech Recognition - Wikipedia

5. Wrap-Up & Next Steps

We have introduced Hidden Markov Models as a powerful extension of Markov chains for modeling hidden dynamics in sequential data. An HMM lets us **maintain a belief over hidden states** and update it as observations arrive, which is invaluable for AI systems operating with partial information. We saw how HMMs are defined by transition probabilities and emission probabilities, and how they generate observations from hidden state sequences. By leveraging algorithms like Forward, Viterbi, and Baum-Welch, an AI can perform probabilistic inference – computing how likely a given observation sequence is, inferring the most likely current state or state sequence, and even learning the model itself from experience.

In practical terms, HMMs bridge the gap between *pure Markov chains* (with fully visible states) and the uncertain, sensor-driven scenarios that AI agents face. They enable **probabilistic inference** in a

principled way, which is a core theme in AI. Many applications we discussed – speech recognition, user intent prediction, opponent modeling, trend detection – illustrate how HMMs bring Markov chains into the real world where you can't directly observe the system's state. By converting observation data into probabilistic state estimates, HMMs allow AI agents to make informed decisions rather than guessing blindly.

As we move forward, understanding HMMs prepares us for more advanced topics. One such next step is the **Markov Decision Process (MDP)**, where we introduce *actions* and *rewards* into the Markov model framework. In an MDP, an agent chooses actions that cause state transitions (with certain probabilities), and we aim to find optimal policies (actions to take in each state) to maximize reward. If we combine the idea of MDPs with partial observability – essentially an agent planning in a hidden-state environment – we arrive at **Partially Observable MDPs (POMDPs)** (). In a POMDP, the agent doesn't know the state for sure, so it carries along a belief (often an HMM-like filter) and makes decisions based on that belief. Solving POMDPs optimally is complex, but many robotics and AI problems can be formulated this way. In essence, HMMs (which handle state uncertainty) become the perception component of a POMDP, and MDP algorithms handle the decision component.

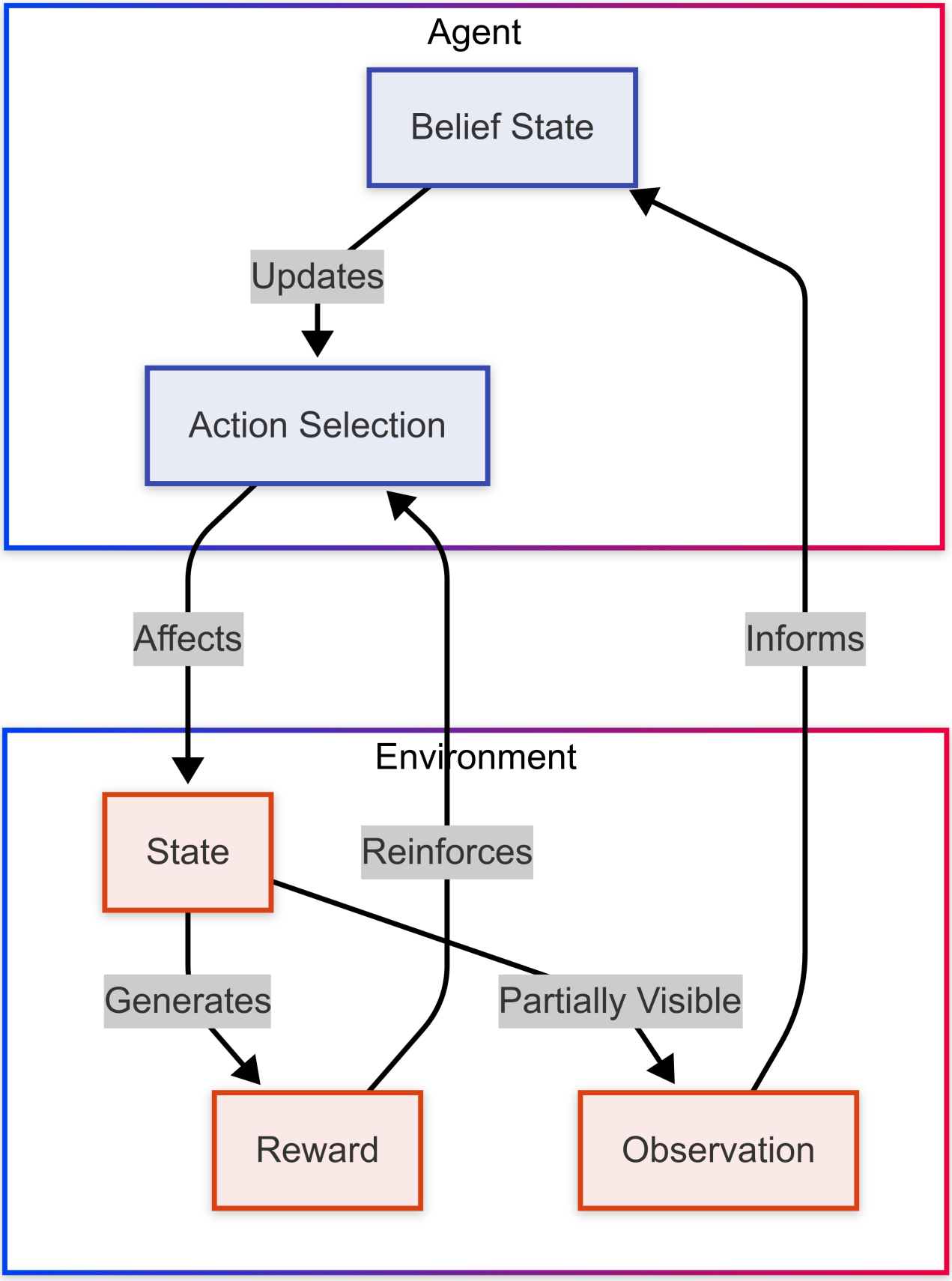
Before diving into MDPs and POMDPs, it's worth appreciating where HMMs stand: they are **a fundamental tool for temporal probabilistic reasoning**. Even in the age of deep learning, the concepts from HMMs (state, observation, transition, emission, belief updates) are present in more complex models like Hidden Markov Random Fields, state-space models, or the inference algorithms of dynamic Bayesian networks.

To wrap up, let's consider a creative real-world example that ties everything together: **gesture recognition** for a smart device. Suppose we want an AI to understand user gestures (like swipe, pinch, zoom) on a touchscreen. We can design an HMM where the hidden states correspond to the phases of a gesture (perhaps "start", "middle", "end" of a swipe, or different intended commands), and the observations are the touch events (finger positions, movements). As the user moves their fingers, the HMM continuously updates the belief of which gesture is happening. By the end of the motion, the device can decode the most likely gesture (using Viterbi). If we deploy this in a real app, over time the AI could even learn refinements (maybe the user has a habit of long-pressing before swiping – the HMM can learn an extra state for that). This demonstrates how an HMM can be embedded in an interactive AI system, interpreting noisy input to produce a useful understanding that drives the system's response.

PROF

Your Turn – Brainstorm! Think of your own example of an HMM in a real-world AI scenario. It could be anything where there's a sequence of observations and some hidden context. For instance, how might an HMM be used in **health monitoring** (hidden state = patient health, observations = wearable sensor readings)? Or in **finance** (hidden state = market regime, observations = price movements (Hidden Markov Models - An Introduction | QuantStart))? By coming up with examples, you practice identifying the hidden/observed decomposition that is at the heart of HMM modeling.

In conclusion, Hidden Markov Models give us a principled way to **"see the invisible"** – to make the hidden states of the world a little less hidden by embracing uncertainty and running with it. As we advance to decision-making and planning under uncertainty, keep the HMM intuition in mind: an agent doesn't need to know everything with certainty to act; it can keep a probability distribution in its head and still make smart choices. HMMs are the first step in that journey of *probabilistic AI*, leading us toward MDPs, POMDPs, and beyond.



PROF

Figure 5: A Markov Decision Process (MDP) diagram showing the interaction between an AI agent and its environment. The agent takes actions based on the current state, and the environment responds with a new state and reward. In a POMDP, the agent doesn't see the true state directly, but only gets observations - making it a combination of an MDP and HMM.

Source: Partially Observable Markov Decision Process - Wikipedia

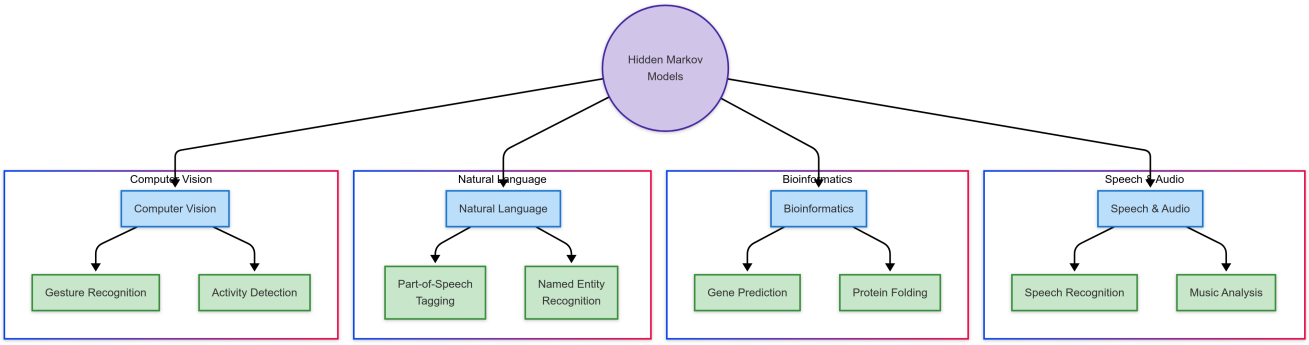


Figure 6: Overview of HMM applications in various fields. From speech recognition to bioinformatics, HMMs help uncover hidden patterns in sequential data by modeling the relationship between hidden states and observable outputs.

Source: Hidden Markov Model - Wikipedia

Figure: The student council's strategic planning sessions - illustrating how agents must make decisions based on partial observations and inferred hidden states.

Source: Kaguya-sama: Love is War - Tenor