MSML610: Advanced Machine Learning

Reasoning Over Time

Instructor: GP Saggese, PhD - gsaggese@umd.edu

References: - AIMA 14: Probabilistic reasoning over time

Reasoning over time

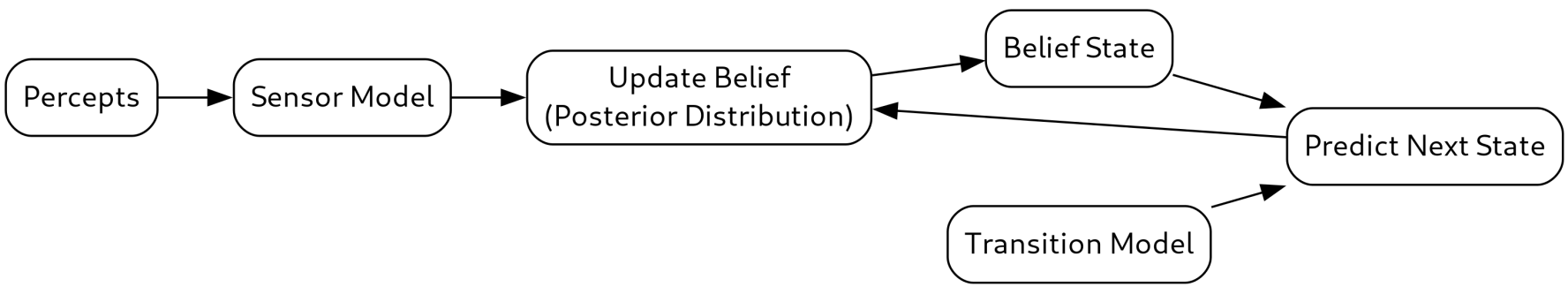
- **Reasoning over time**
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- Dynamic Bayesian networks
- State space model
- Variational Inference

Reference

- AIMA: 14

Agents

- Agents in partially observable environments track the current state using sensor information
  1. **Belief state**
     - Store possible world states (by enumeration or logical formulas)
     - Use probability theory to quantify belief
     - Belief state is the posterior distribution of the current state given all evidence so far
  2. **Belief state + Transition model**
     - Predict how the world might evolve in the next step
  3. **Sensor model + Percepts**
     - Update belief state
- Time is handled by making each quantity a function of time



Static vs dynamic probabilistic reasoning: example

- **Static probabilistic reasoning**
  - Each random variable has a single fixed value over time
  - E.g., when repairing a car:
    - Whatever is broken stays broken during the diagnosis
    - Observed evidence remains fixed
- **Dynamic probabilistic reasoning**
  - Random variables change over time
  - E.g.,
    - Tracking the location of a plane
    - Tracking the economic activity of a nation
  - E.g., treating a diabetic patient
    - Goal: assess the state of the patient and decide on insulin dose
    - Evidence: previous insulin doses, food intake, blood sugar (which change over time)
    - Dependency on time (e.g., metabolic activity and time of day)

Model components

1. State of the world: $\underline{\boldsymbol{X}}_t$
2. Prior probability of the state at time 0: $\underline{\boldsymbol{X}}_0$
3. Evidence variables: $\underline{\boldsymbol{E}}_t$
4. Transition model: $\Pr(\underline{\boldsymbol{X}}_t | \underline{\boldsymbol{X}}_{0:t-1})$
   - How the world evolves
   - Specifies the probability distribution of the state $\underline{\boldsymbol{X}}_t$, given all previous values
5. Sensor model: $\Pr(\underline{\boldsymbol{E}}_t | \underline{\boldsymbol{X}}_{0:t}, \underline{\boldsymbol{E}}_{0:t-1})$
   - How the evidence variables $\underline{\boldsymbol{E}}_t$ are generated

Discrete vs continuous time models

- **Discrete time models**
  - View the world as a series of time slices ("snapshots")
    - Assume time intervals are equal, so samples are equispaced
    - Label times $t = 0, 1, 2, ...$
  - Each time slice contains random variables:
    - Some RVs are not observable $\underline{\boldsymbol{X}}_t$ (hidden)
    - Other RVs are observable $\underline{\boldsymbol{E}}_t$ (evidence)
    - $\underline{\boldsymbol{X}}_{a:b}$ represents variables in $[a, b]$
- **Continuous time models**
  - Uncertainty over continuous time can be modeled by stochastic differential equations (SDEs)
  - Discrete time models can be discrete approximations to SDEs

Markov property

- In general, the current state $\underline{\boldsymbol{X}}_t$ depends on a growing number of past states:

$$\Pr(\underline{\boldsymbol{X}}_t | \textit{history}) = \Pr(\underline{\boldsymbol{X}}_t | \underline{\boldsymbol{X}}_0, \underline{\boldsymbol{X}}_1, ..., \underline{\boldsymbol{X}}_{t-1})$$

- **Markov property**: the current state (conditionally) depends only on a finite fixed number of k previous states:

$$\Pr(\underline{\boldsymbol{X}}_t | \underline{\boldsymbol{X}}_0, \underline{\boldsymbol{X}}_1, ..., \underline{\boldsymbol{X}}_{t-k-1}, \underline{\boldsymbol{X}}_{t-k}, ..., \underline{\boldsymbol{X}}_{t-1}) = \Pr(\underline{\boldsymbol{X}}_t | \underline{\boldsymbol{X}}_{t-1:t-k})$$

Markov process

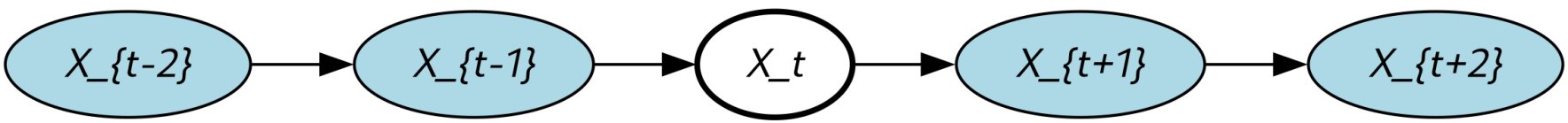
- **Markov processes** (aka **Markov chains**) have the Markov property

$$\Pr(\underline{\boldsymbol{X}}_t | history) = \Pr(\underline{\boldsymbol{X}}_t | \underline{\boldsymbol{X}}_{t-1:t-k})$$

- **First-order Markov process**: the current state depends only on the previous state (and no other earlier state):

$$\Pr(\underline{\boldsymbol{X}}_t | history) = \Pr(\underline{\boldsymbol{X}}_t | \underline{\boldsymbol{X}}_{t-1})$$

  - E.g., the probability of rain today depends only on what happened yesterday $\Pr(R_t | R_{t-1}) \; \forall t$
  - The Bayesian network for a first-order Markov process looks like:



- **Second-order Markov process**: the current state $\underline{\boldsymbol{X}}_t$ is conditionally dependent only on $\underline{\boldsymbol{X}}_{t-1}$ and $\underline{\boldsymbol{X}}_{t-2}$ and no other earlier state

Time-homogeneous process

- Even with the Markov assumption, there are infinite probability distributions $\Pr(\underline{\boldsymbol{X}}_t|\underline{\boldsymbol{X}}_{t-1})$, one for every value of t
- **Stationarity**: the transition probability doesn't change over time

$$\Pr(\underline{\boldsymbol{X}}_t|\underline{\boldsymbol{X}}_{0:t-1}) = \Pr(\underline{\boldsymbol{X}}_{t-k}|\underline{\boldsymbol{X}}_{0:t-k-1}) \; \forall k, t$$

  - The process evolves over time, but the governing laws don't change
- **First-order time-homogeneous**: only one conditional probability table is needed

$$\Pr(\underline{\boldsymbol{X}}_t|\underline{\boldsymbol{X}}_{t-1}) = \Pr(\underline{\boldsymbol{X}}_{t-k}|\underline{\boldsymbol{X}}_{t-k-1}) \; \forall k, t$$

  - E.g., the probability of rain depends on what happened yesterday and is the same every day: $\Pr(R_t|R_{t-1}) = f(R_{t-1}) \; \forall t$

Sensor model

- **Aka "observation model"**
- In general, the evidence variables $\underline{\boldsymbol{E}}_t$ could depend on previous $\underline{\boldsymbol{X}}$ (state of the world) and $\underline{\boldsymbol{E}}$ (sensor value) variables:

$$\Pr(\underline{\boldsymbol{E}}_t | \underline{\boldsymbol{X}}_{0:t}, \underline{\boldsymbol{E}}_{0:t-1})$$

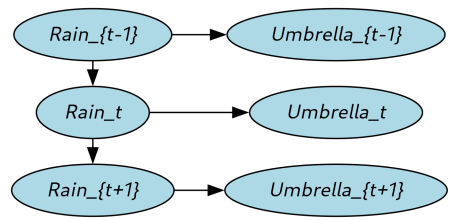
- **Sensor Markov property**
  - We assume that the sensor value $\underline{\boldsymbol{E}}_t$ depends only on the current state of the world $\underline{\boldsymbol{X}}_t$, not on previous sensor values

$$\Pr(\underline{\boldsymbol{E}}_t | \underline{\boldsymbol{X}}_{0:t}, \underline{\boldsymbol{E}}_{0:t-1}) = \Pr(\underline{\boldsymbol{E}}_t | \underline{\boldsymbol{X}}_t)$$

  - In a Bayesian network, even if vX_t and $\underline{\boldsymbol{E}}_t$ are contemporaneous in the time step, the arrow goes from the state of the world $\underline{\boldsymbol{X}}_t$ to the sensor value $\underline{\boldsymbol{E}}_t$, i.e., $\underline{\boldsymbol{X}}_t \rightarrow \underline{\boldsymbol{E}}_t$ since the world causes the sensor to take on particular values

Sensor model: rain example

- In a Bayesian network, $\underline{\boldsymbol{X}}_t \rightarrow \underline{\boldsymbol{E}}_t$ since the world causes the sensor to take on particular values
  - E.g., the rain "causes" the umbrella to appear
  - The inference goes in the other direction: we see the umbrella and need to guess if it's raining
- E.g.,
  - The transition model is $\Pr(Rain_t|Rain_{t-1})$
    - $\Pr(R_t|R_{t-1} = T) = 0.7$
    - $\Pr(R_t|R_{t-1} = F) = 0.3$
    - The sum doesn't have to be 1 since it's a conditional probability
  - The sensor model is $\Pr(Umbrella_t|Rain_t)$
    - $\Pr(U_t|R_t = T) = 0.9$
    - $\Pr(U_t|R_t = F) = 0.2$



Prior probability

- To complete the system specification, we need the prior probability of the state variables at time 0, $\Pr(\underline{\boldsymbol{X}}_0)$
  - Represents the initial belief about the state of the system before any observations are made
  - It is crucial for initializing the state estimation process
- E.g.,
  - $\underline{\boldsymbol{X}}_0$ represents the position and velocity of a moving object
  - $\Pr(\underline{\boldsymbol{X}}_0)$ could be a Gaussian distribution centered around an initial guess of the object's position and velocity with some uncertainty

First-order Markov process: Joint Distribution

- Model a sequence of states $\underline{\boldsymbol{X}}_0, \underline{\boldsymbol{X}}_1, ..., \underline{\boldsymbol{X}}_t$ and observations $\underline{\boldsymbol{E}}_1, ..., \underline{\boldsymbol{E}}_t$ over time, with the simplifying assumptions:
  - First-order Markov assumption: $\Pr(\underline{\boldsymbol{X}}_i | \underline{\boldsymbol{X}}_{0:i-1}) = \Pr(\underline{\boldsymbol{X}}_i | \underline{\boldsymbol{X}}_{i-1})$
  - Sensor model: $\Pr(\underline{\boldsymbol{E}}_i | \underline{\boldsymbol{X}}_{0:i}, \underline{\boldsymbol{E}}_{1:i-1}) = \Pr(\underline{\boldsymbol{E}}_i | \underline{\boldsymbol{X}}_i)$
- The joint distribution of n random variables:

$$\Pr(X_1, ..., X_n) = \prod_{i=1}^{n} \Pr(X_i | \text{parents}(X_i))$$

- The joint distribution probability can be written for any t:

$$\begin{aligned} \Pr(\underline{\boldsymbol{X}}_{0:t}, \underline{\boldsymbol{E}}_{1:t}) &= \Pr(\underline{\boldsymbol{X}}_0) \prod_{i=1}^{t} \Pr(\underline{\boldsymbol{X}}_i | \underline{\boldsymbol{X}}_{i-1}) \Pr(\underline{\boldsymbol{E}}_i | \underline{\boldsymbol{X}}_i) \\ &= \text{prior} \times \prod_{i} \text{transition model} \times \text{sensor model} \end{aligned}$$

First-order Markov process: intuition

- The joint distribution probability for a time-homogeneous first-order Markov process can be written, for any t:

$$\Pr(\underline{\boldsymbol{X}}_{0:t}, \underline{\boldsymbol{E}}_{1:t}) = \Pr(\underline{\boldsymbol{X}}_0) \prod_{i=1}^{t} \Pr(\underline{\boldsymbol{X}}_i | \underline{\boldsymbol{X}}_{i-1}) \Pr(\underline{\boldsymbol{E}}_i | \underline{\boldsymbol{X}}_i)$$
$$= \text{prior} \times \prod_{i} \text{transition model} \times \text{sensor model}$$

- Intuition:
  - Each observation depends only on the current state (sensor model)
  - The state evolves probabilistically from the previous state (transition model)
  - This structure reduces complexity and enables tractable inference
- How to represent this process?
  - A Bayesian network can represent a temporal model by modeling time with indices t, i.e., "unrolling the model"
  - Problem: There are infinite t, even assuming the Markov property

Improving approximation of real-world systems

- A first-order Markov process can be reasonable or not, e.g.,
  - A particle following a random walk is well represented by Markov process (by definition)
  - In the umbrella example the rain depends only on what happened the previous day
- How to improve the approximation
  1. Increase the order of the Markov process model
     - E.g., to model "rarely rains more than two days in a row", we need a second-order Markov model $\Pr(Rain_t|Rain_{t-1}, Rain_{t-2})$
  2. Increase the set of state variables
     - E.g., add $Season_t$ to incorporate the historical records
     - This makes the transition model more complicated
  3. Increase the number of sensor variables
     - E.g., $Location_t$, $Temperature_t$, $Humidity_t$, $Pressure_t$
     - This can simplify modeling of the state

Inference tasks in temporal models

- There are several possible applications that we will consider in details

Task	**Description**	**Estimate**
Filtering	Estimate *current* state given past / current obs	$\Pr(\underline{\boldsymbol{X}}_t \mid \underline{\boldsymbol{E}}_{1:t})$
Prediction	Estimate *future* state given past / current obs	$\Pr(\underline{\boldsymbol{X}}\_t + k \mid \underline{\boldsymbol{E}}_{1:t})$ for $k > 0$
Smoothing	Estimate *past* state given past, current, and *future* obs	$\Pr(\underline{\boldsymbol{X}}_k \mid \underline{\boldsymbol{E}}_{1:T})$ for $k < T$
Most Likely Explanation	Find most probable sequence of states given the evidence	$\text{argmax}_{\underline{\boldsymbol{x}}_{1:T}} \Pr(\underline{\boldsymbol{X}}_{1:t} \mid \underline{\boldsymbol{E}}_{1:t})$
Learning	Learn model parameters or structure from data	θ of a model

Task 1: Filtering

- **Aka "state estimation"**
- **Filtering** computes the posterior distribution of the *current* state (belief state) given all evidence to date:

$$\Pr(\underline{\boldsymbol{X}}_t | \underline{\boldsymbol{E}}_{1:t} = \underline{\boldsymbol{e}}_{1:t})$$

  - E.g., estimate the probability of rain today, given all umbrella observations so far $\Pr(Rain_t | Umbrella_{1:t})$
- Filtering is needed by a rational agent to track the current state of the world
  - The agent has a belief about the current state $\Pr(\underline{\boldsymbol{X}}_{t-1})$ at time $t-1$
  - New evidence $\underline{\boldsymbol{e}}_t$ arrives for time t
  - The agent updates its belief about the current state $\Pr(\underline{\boldsymbol{X}}_t)$ at time t
- The term "filtering" refers to the problem in signal processing of filtering out noise in a signal by estimating system parameters

Task 2: Prediction

- **Prediction** involves predicting the posterior distribution over a *future* state, given all evidence to date:

$$\Pr(\underline{\boldsymbol{X}}_{t+k} | \underline{\boldsymbol{e}}_{1:t}) \text{ with } k > 0$$

  - E.g., compute the probability of rain three days from now:

$$\Pr(Rain_{t+3} | Umbrella_{0:t})$$

- Prediction helps a rational agent evaluate possible actions based on expected outcomes

Task 3: Smoothing

- **Smoothing** is the task of computing the posterior distribution over a *past* state given *all* the past, present, and future evidence:

$$\Pr(\underline{\boldsymbol{X}}_k | \underline{\boldsymbol{e}}_{1:t}) \text{ with } 0 \leq k < t$$

  - Note: you have information about the "future" of the evidence, but you don't know the state
  - Smoothing provides a better estimate of the state since it incorporates more evidence about the future
  - E.g., compute the probability that it rained last Wednesday, given all the observations made up to today
- The name "smoothing" refers to the fact that the state estimate is smoother than filtering

Task 4: Most-likely explanation

- **Most-likely explanation** finds the sequence of states $\underline{\boldsymbol{x}}_{1:t}$ most likely to have generated observations $\underline{\boldsymbol{e}}_{1:t}$:

$$\text{argmax}_{\underline{\boldsymbol{x}}_{1:t}} \Pr(\underline{\boldsymbol{x}}_{1:t} | \underline{\boldsymbol{e}}_{1:t})$$

  - E.g.,
    - Umbrella appeared on 3 days, not on the fourth
    - Most likely explanation: rained for 3 days, then stopped
- Applications
  - Speech recognition: most likely sequence of words given sounds
  - Digital processing: reconstruct bit strings over a noisy channel

Task 5: Learning

- Learning involves estimating the transition model $\Pr(\underline{\boldsymbol{X}}_t|\underline{\boldsymbol{X}}_{0:t-1})$ and the sensor model $\Pr(\underline{\boldsymbol{E}}_i|\underline{\boldsymbol{X}}_i)$ from observations
- **Learning:**
  - Can be a byproduct of inference
  - Requires smoothing rather than filtering for better state estimates
    - Smoothing uses all available data to estimate states, leading to more accurate models
    - E.g., in a weather prediction system, smoothing might use past, present, and future data to better estimate the current weather state

Task 1: Recursive filtering: goal

- Aka "recursive state estimation"
- A practical filtering algorithm updates the current state estimate $\underline{\boldsymbol{X}}_{t+1}$ using the previous state $\underline{\boldsymbol{X}}_t$ and the new evidence $\underline{\boldsymbol{e}}_{t+1}$, rather than recomputing it by going over the entire history of the percepts

$$\Pr(\underline{\boldsymbol{X}}_{t+1}|\underline{\boldsymbol{e}}_{1:t+1}) = f(\Pr(\underline{\boldsymbol{X}}_t|\underline{\boldsymbol{e}}_{1:t}), \underline{\boldsymbol{e}}_{t+1})$$

$$\mathit{NextState} = f(\mathit{PreviousState}, e_{t+1})$$

- **Why?**
  - Time and space requirements for updating must be constant if a (finite) agent needs to keep track of current state indefinitively
- **Is it possible?**
  - What is the formula $f(...)$?

Task 1: Recursive filtering: update formula

- Compute the state at time $t+1$ with all the evidence up to that time
- Assume that state and evidence are scalar and not vector: $\Pr(X_{t+1}|e_{1:t+1})$

$$
\begin{aligned}
&\text{Divide up the evidence} && = \Pr(X_{t+1}|e_{1:t}, e_{t+1}) \\
&\text{Bayes rule given } e_{1:t} && = \alpha \Pr(e_{t+1}|X_{t+1}, e_{1:t}) \Pr(X_{t+1}|e_{1:t}) \\
&\text{Markov sensor assumption} && = \alpha \Pr(e_{t+1}|X_{t+1}) \Pr(X_{t+1}|e_{1:t}) \\
&\text{Condition on current state} && = \alpha \Pr(e_{t+1}|X_{t+1}) \sum_{x_t} \Pr(X_{t+1}|x_t, e_{1:t}) \Pr(x_t|e_{1:t}) \\
&\text{Markov assumption} && = \alpha \Pr(e_{t+1}|X_{t+1}) \sum_{x_t} \Pr(X_{t+1}|x_t) \Pr(x_t|e_{1:t})
\end{aligned}
$$

$$\Pr(X_{t+1}|e_{1:t+1}) = f(\Pr(X_t|e_{1:t}), e_{t+1})$$

- The next state is "Sensor model x Transition model x Recursive state"
  - Sensor model: $\Pr(e_{t+1}|X_{t+1})$
  - Transition model: $\Pr(X_{t+1}|x_t)$
  - Recursive term: $\Pr(x_t|e_{1:t})$

Task 1: Recursive filtering: intuition

- Recursive state estimation updates in **two steps** the belief about a system's state over time as new evidence arrives

$$\Pr(X_{t+1}|e_{1:t+1}) = \alpha \Pr(e_{t+1}|X_{t+1}) \sum_{x_t} \Pr(X_{t+1}|x_t) \Pr(x_t|e_{1:t})$$

1. **Prediction step**: Use the transition model to predict the next state based on the current belief

$$\Pr(X_{t+1}|e_{1:t}) = \sum_{x_t} \Pr(X_{t+1}|x_t) \Pr(x_t|e_{1:t})$$

   - Intuition: Project the current belief forward using the model of system evolution

2. **Update step**: Incorporate the new observation to refine the prediction

$$\Pr(X_{t+1}|e_{1:t+1}) = \alpha \Pr(e_{t+1}|X_{t+1}) \Pr(X_{t+1}|e_{1:t})$$

   - Intuition: Correct the prediction using the likelihood of the new evidence

- Maintain $\Pr(X_t|e_{1:t})$, the probability of the current state given all past evidence
  - E.g., in a weather model, if it was likely to rain today and rain usually continues, the prediction leans toward rain tomorrow
  - Seeing an umbrella supports this and updates the belief accordingly

Task 1: Forward update

- We achieved:

$$\Pr(\underline{\boldsymbol{X}}_{t+1}|\underline{\boldsymbol{e}}_{1:t+1}) = \alpha \Pr(\underline{\boldsymbol{e}}_{t+1}|\underline{\boldsymbol{X}}_{t+1}) \sum_{x_t} \Pr(\underline{\boldsymbol{X}}_{t+1}|\underline{\boldsymbol{x}}_t) \Pr(\underline{\boldsymbol{x}}_t|\underline{\boldsymbol{e}}_{1:t})$$
$$= f(\Pr(\underline{\boldsymbol{X}}_t|\underline{\boldsymbol{e}}_{1:t}), \underline{\boldsymbol{e}}_{t+1})$$

- The filtered estimate $\underline{\boldsymbol{f}}_{1:t} = \Pr(\underline{\boldsymbol{X}}_t|\underline{\boldsymbol{e}}_{1:t})$ is propagated forward and updated by each transition and new observation

$$\underline{\boldsymbol{f}}_{1:t+1} = \mathit{Forward}(\underline{\boldsymbol{f}}_{1:t}, \underline{\boldsymbol{e}}_{t+1})$$

  starting with the initial condition $\underline{\boldsymbol{f}}_{1:0} = \Pr(\underline{\boldsymbol{X}}_0)$

  - This is called "forward update"

- This process allows efficient online inference without storing the full history

  - Time and space requirements for updating is constant
  - A (finite) agent can keep track of current state indefinitively

Task 2: Prediction: update formula

- Prediction is equivalent to filtering without updating the state with new evidence (since we lack evidence)
  - Only the transition model is needed, not the sensor model
- The rule predicting state $\underline{\boldsymbol{X}}_{t+k+1}$ given state $\underline{\boldsymbol{X}}_{t+k}$ and evidence $\underline{\boldsymbol{E}}_{1:t}$ is:

$$\Pr(\underline{\boldsymbol{X}}_{t+k+1}|\underline{\boldsymbol{e}}_{1:t}) = \sum_{\underline{\boldsymbol{x}}_{t+k}} \Pr(\underline{\boldsymbol{X}}_{t+k+1}|\underline{\boldsymbol{x}}_{t+k}) \Pr(\underline{\boldsymbol{x}}_{t+k}|\underline{\boldsymbol{e}}_{1:t})$$

- This equation can be used recursively to advance over time
  - Predicting even a few steps ahead generally incurs large uncertainty

Task 3: Smoothing: intuition

- We want to calculate the probability distribution over the hidden state at time k, given all evidence up to time t (in the future!)

$$\Pr(X_k|e_{1:t}) \text{ where } 0 \leq k < t$$

- Filtering gives $\Pr(X_k|e_{1:k})$ using past and present evidence
- Smoothing refines the estimate of past states using later evidence
- **E.g.,**
  - You're tracking whether it was raining yesterday
  - You had some evidence up to yesterday (e.g., a cloudy sky)
  - Today you see puddles on the ground
  - That new observation supports the idea that it was raining

Task 3: Smoothing: update formula

- Using the same math as for filtering and the two key assumptions of Markov process and Markov sensor
- **Forward Pass (aka filtering):**
  - Move forward through time, using the filtering algorithm to compute:

$$f_{1:k} = \Pr(X_k | e_{1:k})$$

  - This gives you a "best guess" of the state at time k, based only on evidence up to k
- **Backward Pass (aka smoothing):**
  - Move backward through time from time t, computing:

$$b_{k+1:t} = P(e_{k+1:t} | X_k)$$

  - This captures how likely the future evidence is, given a particular value of X_k
- **Combine them:**
  - Multiply forward and backward messages to get:

$$P(X_k | e_{1:t}) \propto f_{1:k} \times b_{k+1:t}$$

Task 4: Most likely explanation: Intuition 1/2

- You are tracking the weather (sunny or rainy) based on whether someone carries an umbrella
- You can't see *Weather* directly (hidden state), but you observe umbrellas (which is a noisy observation)
  - You have 5 observations $Umbrella = [T, T, F, T, T]$
- Question: what is the most likely sequence of *Weather* states that explains the *Umbrella* observations?
  - You know something about the transition model (i.e., "it tends to rain several days in a row") and the sensor model (i.e., "people often forget the umbrella")
- Mathematically

$$\text{argmax}_{x_{1:t}} \Pr(x_{1:t}|e_{1:t}) = \textit{argmax}_{Weather_{1:t}} \Pr(Weather_{1:t}|Umbrella_{1:t})$$

Task 4: Most likely explanation: Intuition 2/2

- **Naive approach**: Use smoothing to choose the most likely state at each time step
  - Cons
    - Might lead to an implausible overall path
    - Suboptimal since the question addresses joint probability and we are not using all the information (only one step at the time!)
- **Viterbi algorithm**:
  - Constructs a path through a state-time graph with states as nodes and transitions as edges
  - Finds the most likely entire path through the hidden states
- **Key difference:**
  - E.g., in speech recognition, find the most likely word sequence behind a noisy audio signal
  - Smoothing: Best guess per time step (may not find words that are not English and / or suboptimal sequence)
  - Viterbi: Best overall path (maximizes joint probability of the entire sequence)

Viterbi algorithm: Intuition

- Objective: Find the most likely **sequence** of hidden states given observations

1. Initialization
   - At $t = 1$, estimate the probability of starting in each state using the initial state distribution and observation likelihood
2. Recursion via dynamic programming
   - At each time $t > 1$, for each state x_t:
   - Compute the maximum probability path to x_t from any previous state
   - Use:
     - $\Pr(x_t|x_{t-1})$: transition model
     - $\Pr(e_t|x_t)$: sensor model
     - Best path probability to x_{t-1} from prior step
   - Store the probability and the corresponding back-pointer to x_{t-1}
3. Termination and backtrace
   - At final time $t = T$, identify the state with the highest final probability
   - Trace back through the stored pointers to reconstruct the optimal path

Viterbi algorithm: Example 1/2

- You observe a friend carrying an umbrella over 3 days:
  $Umbrella = [Yes, Yes, No]$
- You want to infer the most likely sequence of hidden *Weather* states
  - States: $S = \{\text{Sunny}, \text{Rainy}\}$ (weather)
  - Observations: $O = \{\text{Yes}, \text{No}\}$ (umbrella)
- Initial Probabilities:

$$\Pr(\text{Sunny}) = 0.6, \quad \Pr(\text{Rainy}) = 0.4$$

- Transition Probabilities:

$$\Pr(\text{Sunny} \to \text{Sunny}) = 0.7, \quad \Pr(\text{Sunny} \to \text{Rainy}) = 0.3$$
$$\Pr(\text{Rainy} \to \text{Sunny}) = 0.4, \quad \Pr(\text{Rainy} \to \text{Rainy}) = 0.6$$

- Observation (Emission) Probabilities:

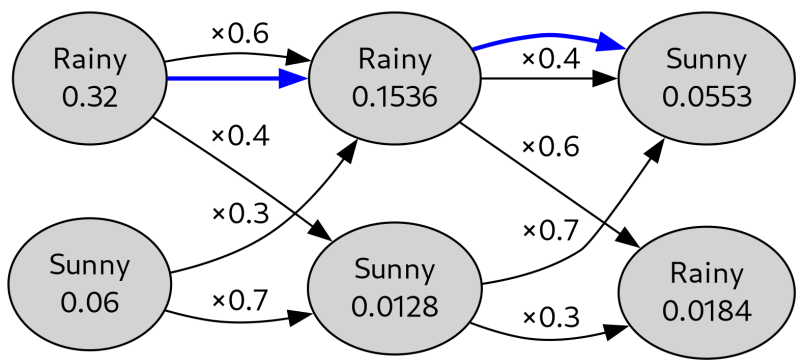
$$\Pr(\text{Yes}|\text{Sunny}) = 0.1, \quad \Pr(\text{No}|\text{Sunny}) = 0.9$$
$$\Pr(\text{Yes}|\text{Rainy}) = 0.8, \quad \Pr(\text{No}|\text{Rainy}) = 0.2$$

Viterbi algorithm: example 2/2

- Viterbi table

Day	State	Probability	Backpointer
1	Sunny	$0.6 \times 0.1 = \mathbf{0.06}$	—
	Rainy	$0.4 \times 0.8 = \mathbf{0.32}$	—
2	Sunny	$\max(0.06 \times 0.7,\ 0.32 \times 0.4) \times 0.1 = \mathbf{0.0128}$	Rainy
	Rainy	$\max(0.06 \times 0.3,\ 0.32 \times 0.6) \times 0.8 = \mathbf{0.1536}$	Rainy
3	Sunny	$\max(0.0128 \times 0.7,\ 0.1536 \times 0.4) \times 0.9 = \mathbf{0.0553}$	Rainy
	Rainy	$\max(0.0128 \times 0.3,\ 0.1536 \times 0.6) \times 0.2 = \mathbf{0.0184}$	Rainy

- Final most probable state: **Sunny** (Day 3)
- Find the most likely sequence: *Rainy* → Rainy → Sunny



HMMs

- Reasoning over time
- **HMMs**
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- Dynamic Bayesian networks
- State space model
- Variational Inference

Algorithms for Specific Models

- General temporal probabilistic reasoning makes minimal assumptions:
  - Markov property for transitions
  - Sensor model depends only on current state
  - No constraints on:
    - Mathematical form of transition/sensor models
    - Nature of state and evidence variables (discrete or continuous)
- Efficiency and accuracy can improve by exploiting specific model structures:
  - **Hidden Markov Models (HMMs)**:
    - State is a single discrete variable
    - Transition and observation models are discrete probability tables
    - Enables fast algorithms like the Viterbi algorithm, forward-backward, etc
  - **Kalman Filters** (for continuous domains):
    - State variables are continuous and normally distributed
    - Linear-Gaussian models for transitions and observations
    - Allows exact, efficient updates using matrix operations
- Tailored algorithms can be orders of magnitude faster and more accurate than general methods

Hidden Markov Model: State and Transition Model

- **Hidden Markov Model (HMM)**: A temporal model with simplified structure for efficiency
  - **State model**:
    - The system state at time t is a discrete random variable $X_t \in \{1, \ldots, S\}$
    - E.g., in the umbrella domain, $X_t = Rain_t$ with states {rain, no rain}
    - Generality: Multiple variables can be combined into one "mega-state" variable
  - **Transition model** $\Pr(X_t|X_{t-1})$:
    - Described by a transition matrix $\underline{\boldsymbol{T}}$ of size $S \times S$
    - Entry $T_{ij} = \Pr(X_t = j|X_{t-1} = i)$: probability of transitioning from state i to j
  - **Sensor model**:
    - Defined as $\Pr(E_t|X_t = i)$ for each state i
    - Representable as a vector (discrete observations) or a diagonal matrix $\underline{\boldsymbol{O}}$ (for convenience)
    - No assumptions about the number or type (discrete / continuous) of observation variables
- **Benefit**
  - This structure enables efficient algorithms like forward, backward, and Viterbi

Hidden Markov Model: umbrella example

- E.g., if $Rain = T$ is state 1 and $Rain = F$ is state 2, then the transition matrix for the umbrella world

R_{t-1}	$\Pr(R_t \mid R_{t-1})$
T	0.7
F	0.3

becomes the transition model

$$\underline{\underline{\boldsymbol{T}}} = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$$

- On day 1 we observe $U_1 = T$ and on day 3, $U_3 = F$, we have the observation matrices

$$\underline{\underline{\boldsymbol{O}}}_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix} \quad \underline{\underline{\boldsymbol{O}}}_3 = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.8 \end{pmatrix}$$

Hidden Markov Model: algorithms

- Using the matrix representation all the forward / backward computations become matrix operations:

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^\top \mathbf{f}_{1:t} \qquad \mathbf{b}_{k+1:t} = \mathbf{T} \mathbf{O}_{k+1} \mathbf{b}_{k+2:t}$$

- So all the inference tasks (e.g., filtering, smoothing) can be expressed as matrix multiplication which are typically efficient
- There are several improvements that can be done to reduce time and space complexity
  - Baum-Welch
    - Special case of Expectation-Maximization (EM) algorithm
    - Pros: Converge to a local maximum of the likelihood
    - Cons: Only point-estimation of params, no uncertainty estimation
  - Viterbi
    - Find the most likely sequence of hidden states
    - Pros: Fast approximation of BW
    - Cons: returns a local optimum
  - Gradient-based methods
    - For more complex models with differentiable form
    - Use gradient descent to optimize parameter

Hidden Markov Model: applications

- HMMs are very versatile to model systems that have hidden states which produce observable outputs
- Audio / speech
  - Speech recognition: map audio to phonemes, words
  - Speaker identification: model vocal traits to recognize a speaker
  - Music generation and transcription
- Biology / genomics
  - Gene prediction: find regions of DNA
  - Protein structure prediction
- Finance / economics
  - Market regime detection: e.g., bull/bear markets, volatility regimes
  - Credit scoring: observe purchases and estimate financial health (hidden variable)
- Security / anomaly detection
  - User behavior modeling: detect anomalous login patterns or usage activities
  - Intrusion detection: model normal traffic patterns to spot attacks

HMMs: limitations

- Short memory
  - Markov assumption, i.e., current state depends only on the previous state
  - Inefficient for capturing long-range dependencies or context
- Predefined and fixed number of states
  - Underestimating / overestimating the number of states can lead to underfitting or overfitting
- Stationarity assumption
  - Transition and sensor probabilities are assumed to be constant over time
- Use an atomic representation
  - The states have no internal structure and are simply labels
- Training is computationally expensive for large datasets
- Struggles with sparse data
- Hard to interpret when there is a lot of states or states don't have a clean meaning
- Alternatives

Markov random fields

- Reasoning over time
- HMMs
- **Markov random fields**
- Markov logic network
- State space models and Kalman filter
- Dynamic Bayesian networks
- State space model
- Variational Inference

Markov logic network

- Reasoning over time
- HMMs
- Markov random fields
- **Markov logic network**
- State space models and Kalman filter
- Dynamic Bayesian networks
- State space model
- Variational Inference

State space models and Kalman filter

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- **State space models and Kalman filter**
  - g-h filter
  - Discrete Bayes filter
- Dynamic Bayesian networks
- State space model
- Variational Inference

Reference

- https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python

Tracking objects

- Many problems can be formulated as "tracking objects", e.g.,
  - Navigation of aircraft, drones, autonomous cars
  - Robotics: arm kinematics to predict the position of joints
  - Sensor fusion: merge multiple sensor readings
  - Finance: predict economic variables (e.g., stock prices)
  - Computer vision: track moving objects across video
  - Aerospace: radar tracking, missile, satellite
- Kalman filter is widely used for state estimation in dynamic systems, when measurements are noisy and uncertain
  - Track something over time using a combination of predictions (i.e., a model) and observations

Some guiding principles

- Sensors are noisy
  - E.g., kitchen scale gives different readings when you weigh the same object twice
- World is noisy
  - A car might have swerved around a pothole or have braked for a pedestrian
  - Wind or ice might change the path of the car
- Knowledge is uncertain
  - We alter our beliefs based on the strength of the evidence
- Use past information and the knowledge of a system to estimate future information
  - E.g., if a car is moving at a certain speed at time t, the speed at time $t + 1$ is probably close to the previous speed
- Never throw information away, no matter how poor it is
  - Two sensors, even if one is less accurate than the other, is better than one
  - Include all the possible sources of information to form the best estimate possible
- Data is better than a guess, if it's noisy

g-h filter

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
  - **g-h filter**
  - Discrete Bayes filter
- Dynamic Bayesian networks
- State space model
- Variational Inference

Blending predictions and measurements

- We want to track the position of a car on a drag race track (1 dimensional)
- We measure velocity with sensors
  - Based on physics equation we should know exactly the position of the car
- Imagine we measure also the position
  - The prediction of the position doesn't match the measurement of the position
  - This is expected since:
    - velocity sensor is noisy
    - the wheels lose grip
    - there is air attrition
- If we form estimates from the measurements, then the predictions are useless
- If we only form estimates from the prediction, then the measurements will be ignored
- We need to blend prediction and measurement

Example of weight: correct gain_rate

- We have a noisy scale
- We estimate the change of weight from one day to another based on the calories consumed
  - Assume that we gain 1lb / day
- At time $t-1$ our last estimate $\hat{x}_{t-1} = 158$
- At time t:
  - we measure 164 with a scale
  - we estimate $\hat{x}_{t|t-1} = 159$ based on the calories intake
- Assume we blend the estimates like:

$$\text{estimate} = 0.6 \text{ prediction} + (1 - 0.6) \text{ measurement}$$

  - This means that we believe that the prediction is more likely to be correct than the measurement
- ***Algorithm***
  0. Start with an initial guess (assume it's correct for now)
  1. Predict the next weight based on the model
  2. Measure the weight
  3. Estimate the next weight by merging different values:
     - The prediction is always between the prediction and the measurement
  4. Go back to 1)

Code

- The code in Python is here /// TODO
- The initial guess is 160 lbs
- The red line is the **prediction** from previous day's weight
- The **measurements** are the circles
- The blue line is the **estimate** from the filter
    - They always fall between measurement and prediction
- The black line is the actual weight, i.e., **ground truth**
- It's not very impressive since the prediction model describes exactly the ground truth, so we don't really need the measurements

Example of weight: incorrect gain_rate

- Consider the case where the model predicts a gain of -1lb / day, which is incorrect
- The estimates diverge from the measurements
- The filter needs a correct guess of the rate of weight change
  - The problem is that the rate of change can change over time
  - The filter will adjust but just not fast enough

Example of weight: learning gain_rate

- A solution is to estimate the rate of change from the measurements
  - Data is better than a guess, even if it's noisy
  - The idea is to refine the estimate of the gain rate:

$$\text{new gain} = \text{old gain} + 0.3\ (\text{measurement - prediction}) / 1\ \text{day}$$

- Note that now the state is given by `weight` and `gain_rate`, so we need to predict and update both
- The code in Python is here /// TODO

g-h filter

- The previous algorithm is called **g-h filter**
  - g: scaling used to blend predicted state and measurement
  - h: scaling used to update the parameter of the system model based on the measurements
- g-h filters have different values of g and h to achieve different properties
  - E.g., pick g to minimize the transient error when the derivative of the signal has a step (i.e., a discontinuity of the slope)
- Many filters (including Kalman filter) are just generalization of a g-h filter

Control theory nomenclature

- system = object we want to estimate / track
- state of the system `x` = current values we are interest in
  - E.g., weight
  - Part of the state might be hidden (i.e., not observable)
  - We cannot observe the entire state directly, we can only measure it indirectly
- measurement `z` = the measured value of the system
  - It can be inaccurate
    - E.g., 99.3kg instead of 100kg
  - It is observable
- state estimate `x_est` = our filter estimate of the state
- system model = mathematical model of the system
  - E.g., "weight today = weight yesterday + weight gain"
  - The system model is typically imperfect
- system error = error in the specification of the model
- system propagation = predict step using the system model to form a new state estimate `x_pred`
  - Because of system error, the estimate is imperfect
- measurement update = update step

g-h filter algorithm: pseudo-code

0. Initialization

- initialize the state of the filter
- initialize our belief in the state

1. Predict

- use system model to predict state at next time step
- adjust belief to account for uncertainty in prediction

2. Update

- get measurement and associated belief about its accuracy
- use as estimate of the next state a point between estimated state and measurement

Interpretation of g

- If $g = 0$, the filter follows the system model, ignoring the measurements
- If g increases, the filter follows the measurements more and more, ignoring the prediction
  - This is useful when the measurements are accurate and the system model inaccurate
- If $g = 1$, the filter follows only the measurements, ignoring the system model

Interpretation of h

- The system model can be parameterized and we might need to estimate some model parameters from data, e.g.,
  - in g-h models the rate of change of the measurements
  - the change of weight
  - the speed of the car on different terrains
- If $h = 0$, the filter follows the previous values of the rate of change of the underlying model
  - I.e., it adapts slowly to the change of the signals
- If $h = 1$, the filter reacts to the transient rapidly if the signal varies a lot with respect to the time step
- Note that incorrect initial state (e.g., initial value / rate of change) is similar to a changing state

Discrete Bayes filter

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
  - g-h filter
  - **Discrete Bayes filter**
- Dynamic Bayesian networks
- State space model
- Variational Inference

Tracking a dog: example

- There is a dog wandering in a circular hallway connecting offices with doors
  - There are 10 positions in the hallway numbered 0 to 9
  - Because of the circular hallway after position 9 there is 0
- The dog has a sensor that can:
  - detect if the dog is in front of an open door or not (e.g., we get signals like `door`, `hall`, `hall`)
  - report in which direction the dog has moved
- Updates are reported every second
- When we start listening to the sensor we don't know where the dog is
  - The prior is uniform over the 10 positions
  - We assign a probability equal to $1/10$ to each door
- Prior is the probability before incorporating measurements or other information
- Posterior distribution is the probability distribution after seeing the data

Tracking a dog

- Initially let's assume that the sensor is always correct
- Before the signal we believed that the dog could have been anywhere
  - Uniform belief over the 10 locations
- We receive the signal `door`
  - There are 3 doors
  - Now we know that the dog is in front on one of the doors
- If we receive a sequence of signals like `door`, `move_right`, `door` we might be able to conclude where exactly the dog is
  - E.g., if there are only two doors close to each other

Noisy sensors

- No sensor is rare
- To account for that we need to assign a probability for the dog to be in other positions to account for the noisy sensors

Convolution and PDF

- You have a certain PMF of where the dog can be in [0, 9] position X
- The sensor reporting how many positions the dog moved is noisy
  - The PMF of the sensor has a certain shape centered around the correct value (unbiased) d
- Computing the "update" of the position of the dog, after a reading of the movement of the dog from the sensor is:
  - $X + d$ (normalized to 1)
  - equivalent to a convolution between the two PMFs
- Note that after the convolution the PMF becomes more spread
  - This represents the fact that uncertainty increases

Dynamic Bayesian networks

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- **Dynamic Bayesian networks**
- State space model
- Variational Inference

Dynamic Bayesian Networks (DBNs)

- DBNs extend Bayesian networks to model temporal processes
- Main idea
  - "Unroll" the model over time
  - Capture intra-slice (within time) and inter-slice (across time) dependencies
- Each time slice includes:
  - State variables X_t
  - Evidence variables E_t
- Assumptions
  - First-order Markov process: current state depends only on the previous state
  - First-order sensor Markov process: evidence depends only on current state
  - Stationarity: each time slice is the same, both structure and parameters do not change over time
    - Structure and CPTs (Conditional Probability Tables) are the same across slices (time-homogeneous model)
  - No Gaussian distribution

DBNs vs HMMs

- DBNs generalize Hidden Markov Models (HMMs)
- HMMs are a special case with a single hidden and evidence variable per time step
- DBNs model more complex systems than HMMs by:
  - Using multiple state variables
    - Enables modeling large systems like robot localization with many state components
  - Exploiting sparse connections among variables yielding compact model
    - HMM: transition matrix of size $O(d^{2n})$
    - DBN: size $O(nd^k)$ with k bounded parents per variable

DBNs vs Kalman filters

- DBNs generalize Kalman filters
- Every Kalman filter can be represented in a DBN with:
  - Continuous variables
  - Linear / Gaussian conditional distributions
- Not every DBN can be represented by a Kalman filter, since:
  - DBN variables can mix discrete/continuous and non-Gaussian
  - Allow arbitrary conditional dependencies among variables
- **Pros of DBNs**
  - DBNs are applicable to broader domains including:
    - Fault diagnosis in networks
    - Complex system monitoring
- **Pros of Kalman filters:**
  - Optimal for linear systems with Gaussian noise
  - Support exact inference, DBNs often require approximate methods

Constructing a DBN

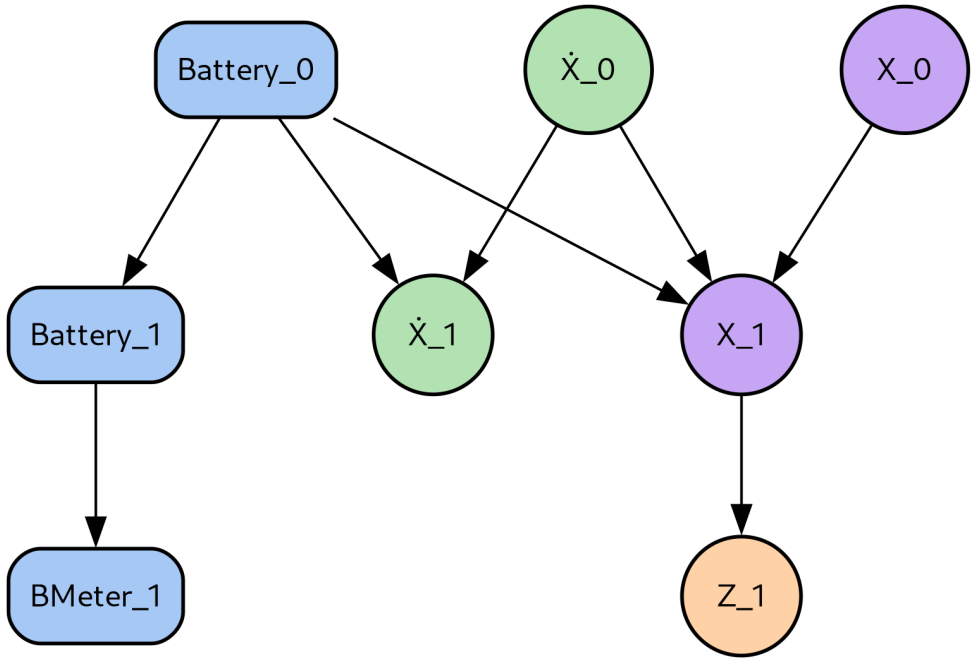
- Key components of a DBN
  - Prior distribution of state $\Pr(X_0)$
  - Transition model $\Pr(X_{t+1}|X_t)$
  - Sensor model $\Pr(E_t|X_t)$
  - Transition and sensor models are time-homogeneous
- Network topology includes:
  - Intra-slice topology
  - Inter-slice links

DBN example: Tracking a robot (1/3)

- **Problem:**
  - Tracking a robot moving randomly on a line X over time
- **Initial model:**
  - Position X_t and velocity $\dot{X}_t$ as state variables
  - Update via Newton's laws
  - Easy to generalize for 2d or 3d by using a $\underline{\boldsymbol{X}}_t$
- **Issue:**
  - Velocity changes over time
  - Battery exhaustion affects velocity systematically
  - Effect depends on cumulative energy use
  - Violates the Markov property (future depends on full history)
- **Solution:**
  - Include battery level $Battery_t$ in the state X_t
  - Restores the Markov assumption
  - Allows motion prediction considering energy constraints
  - Enables coherent reasoning about motion and power consumption over time
- **New requirement for state:**
  - $S_t = (X_t, \dot{X}_t, \text{BatteryLevel}_t)$
  - $E_t = (\text{GPS}_t, \text{BMeter}_t)$

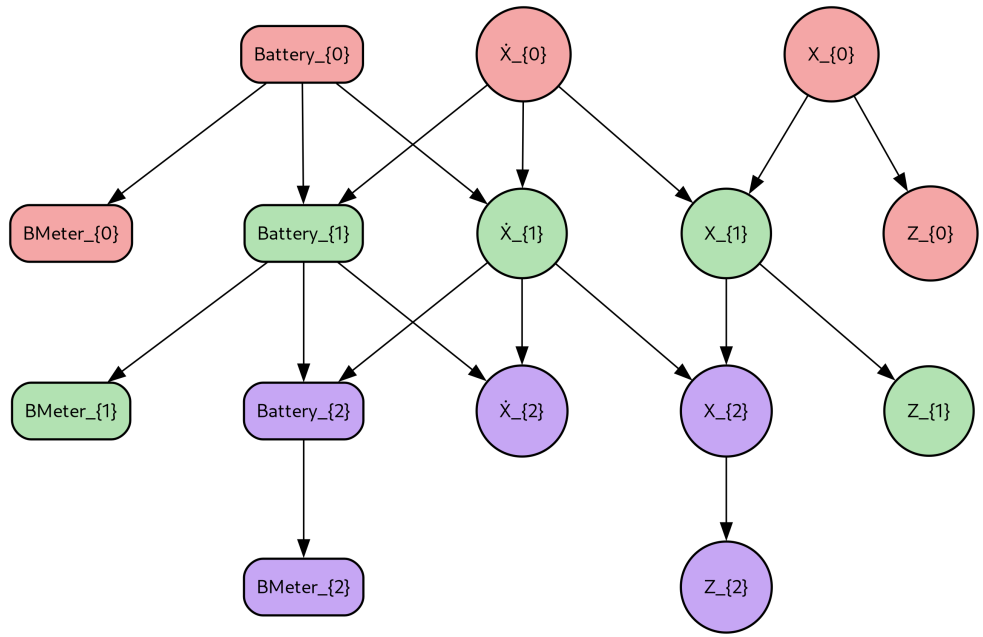
DBN example: Tracking a robot (2/3)

- The DBN structure models both intra-slice (within time) and inter-slice (across time) dependencies
- Intra-slice dependencies:
  - Position X_t influences velocity $\dot{X}_t$
  - BatteryLevel_t influences velocity $\dot{X}_t$
  - Battery_{t+1} depends on Battery_t and $\dot{X}_t$
  - BMeter_t depends on Battery_t
  - GPS_t depends on X_t
- Inter-slice dependencies:
  - Position X_{t+1} depends on Position X_t and velocity $\dot{X}_t$
  - Velocity $\dot{X}_{t+1}$ depends on $\dot{X}_t$ and Battery_t



DBN example: Tracking a robot (3/3)

- **Replicate for Multiple Time Slices**:
  - Create slices for $t = 0, 1, 2, \ldots$ with the above variables and dependencies
  - Group each time slice vertically or horizontally for clarity
- **Unrolling**:
  - Visualize the full DBN by unrolling these slices over the desired number of time steps (e.g., three slices for $t = 0, 1, 2$)



Inference in DBNs

- DBNs are Bayesian networks and we can use the same inference algorithms
  - "Unroll" the DBN over time (i.e., replicate slices for each time step) and apply standard BN inference
  - We can't unroll "forever", but we limit to a certain number of slices to approximate a fixed amount of time dependency
- Use recursive methods to get a constant time and space update complexity
  - Variable elimination with temporal ordering
  - At time step $t+1$ add slice $t+2$ and remove slice t so one has always two slices to do inference
  - Maintains constant memory by keeping only two slices at a time
- Complexity:
  - Exponential in number of state variables ($O(nd^{n+k})$)
  - More efficient than full HMM representation ($O(d^{2n})$)
- Even though we can use DBNs to represent very complex temporal processes with many sparsely connected variables, we cannot reason efficiently and exactly about those processes
  - The prior joint distribution over all the variables is factorizable into its constituents CPTs
  - The posterior joint distribution conditioned on observation sequence is not

Approximate Inference in DBNs

- Particle Filtering:
  - Represent belief state with weighted samples (particles)
  - Steps: propagate, weight, resample
- Benefits:
  - Focuses computation on high-probability regions
  - Maintains manageable memory and time per step
- Challenges:
  - Approximation error
  - Sensitive to transition and observation model assumptions
- Used when exact inference is computationally impractical
- Real-world application: robot localization, speech recognition

DBN to represent changing model

- We can model the fact that the system can change over time
  - Transient failure: a sensor reads wrong measures
  - Persistent failure model: we can model it with additional variables (e.g., *SensorBroken*)

DBN: inference

- We can unroll the DBN and get a BayesNet and then perform exact or approximate inference with the known methods (e.g., MCMC)

DBN: optimization for inference

- Many optimizations are possible, e.g.,
  - Instead of running each sample through the entire DBN one can run all the samples evaluating one slice at a time to compute the posterior distribution

State space model

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- Dynamic Bayesian networks
- **State space model**
- Variational Inference

Variational Inference

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- Dynamic Bayesian networks
- State space model
- **Variational Inference**
  - Expectation-Maximization (EM) Algorithm

Expectation-Maximization (EM) Algorithm

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- Dynamic Bayesian networks
- State space model
- Variational Inference
  - **Expectation-Maximization (EM) Algorithm**

EM Algorithm: Intuition and Applications

- Expectation-Maximization (EM) is a method for learning with hidden or missing data
  - Useful when some variables influencing the data are not directly observed
  - Works by iteratively improving parameter estimates
  - Alternates between estimating missing data and optimizing parameters
- Two main steps:
  - **E-step (Expectation)**: Estimate distribution over hidden variables using current parameters
  - **M-step (Maximization)**: Update parameters to maximize expected log-likelihood from the E-step
- Used in diverse settings:
  - Unsupervised clustering (e.g., Gaussian Mixture Models)
  - Learning with incomplete data in Bayesian networks
  - Hidden Markov Models (HMMs)
- Key property: EM increases data likelihood at each iteration
- Converges to a local maximum of the likelihood function
- No need for a step size parameter unlike gradient descent

EM Algorithm: Mechanics and Example in Gaussian Mixture Models

- Goal: Recover parameters of Gaussian components from unlabeled data
- **E-step**:
  - Compute $p_{ij} = P(C = i \mid x_j)$ using Bayes' rule
  - $p_{ij} \propto P(x_j \mid C = i)P(C = i)$
  - Calculate effective count: $n_i = \sum_j p_{ij}$
- **M-step**:
  - Update means: $\mu_i \leftarrow \sum_j p_{ij} x_j / n_i$
  - Update covariances: $\Sigma_i \leftarrow \sum_j p_{ij}(x_j - \mu_i)(x_j - \mu_i)^T / n_i$
  - Update weights: $w_i \leftarrow n_i / N$
- Intuition: Softly assign points to components, then re-estimate the components
- Example scenario:
  - 500 data points from a mix of 3 Gaussians
  - EM reconstructs original distribution closely after iterations
- Limitations:
  - Sensitive to initialization
  - May converge to poor local optima
  - Component collapse or merging can occur

Introduction to the Expectation–Maximization (EM) Algorithm

- **Purpose of EM Algorithm**
  - Iterative method for finding maximum likelihood or maximum a posteriori (MAP) estimates in statistical models with latent variables
  - Particularly useful when data is incomplete or has missing values
- **Key Concepts**
  - **Observed Data (X)**: The data we can directly observe
  - **Latent Variables (Z)**: Hidden or unobserved variables that influence the observed data
  - **Parameters (θ)**: Unknown parameters to be estimated
- **Challenge Addressed**
  - Direct maximization of the likelihood function $p(\mathbf{X}|\theta)$ is often intractable due to the presence of latent variables
- **EM Algorithm Overview**
  - Alternates between estimating the expected value of the log-likelihood (E-step) and maximizing this expectation (M-step)
- **Applications**
  - Widely used in clustering (e.g., Gaussian Mixture Models), natural language processing, and image reconstruction

The EM Algorithm: Step-by-Step

- **Initialization**
  - Start with initial guesses for the parameters $\boldsymbol{\theta}^{(0)}$
- **E-Step (Expectation Step)**
  - Compute the expected value of the log-likelihood function, with respect to the conditional distribution of the latent variables given the observed data and current parameter estimates:
    - $Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) = \mathbb{E}_{\mathbf{Z}|\mathbf{X},\boldsymbol{\theta}^{(t)}}[\log p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta})]$
- **M-Step (Maximization Step)**
  - Maximize the expected log-likelihood found in the E-step to update the parameters:
    - $\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$
- **Iteration**
  - Repeat E and M steps until convergence, i.e., until the parameters stabilize or the increase in likelihood is below a threshold

Mathematical Foundation of EM

- **Likelihood with Latent Variables**
  - The marginal likelihood of the observed data is:
    - $p(\mathbf{X}|\theta) = \int p(\mathbf{X}, \mathbf{Z}|\theta) d\mathbf{Z}$
- **Intractability**
  - The integral is often difficult to compute due to the complexity introduced by the latent variables
- **EM Solution**
  - EM circumvents this by iteratively applying the E and M steps to find parameter estimates that locally maximize the likelihood
- **Convergence**
  - Each iteration of EM is guaranteed to increase the likelihood function, ensuring convergence to a local maximum

Example: Gaussian Mixture Models (GMM)

- **Problem Setup**
  - Data is assumed to be generated from a mixture of Gaussian distributions, each with its own mean and covariance
- **Latent Variables**
  - Each data point is associated with a latent variable indicating the Gaussian component from which it was generated
- **E-Step in GMM**
  - Compute the posterior probabilities (responsibilities) that each data point belongs to each Gaussian component
- **M-Step in GMM**
  - Update the parameters (means, covariances, and mixing coefficients) of each Gaussian component using the responsibilities computed in the E-step
- **Iteration**
  - Repeat E and M steps until the parameters converge

Properties and Limitations of EM

- **Advantages**
  - Can handle missing or incomplete data effectively
  - Provides a framework for parameter estimation in complex models
- **Limitations**
  - Converges to a local maximum, which may not be the global maximum
  - Sensitive to initial parameter estimates; poor initialization can lead to suboptimal solutions
- **Extensions and Variants**
  - **Variational Bayes**: Provides a fully Bayesian approach by estimating distributions over parameters
  - **Generalized EM (GEM)**: Relaxes the requirement of fully maximizing the expected log-likelihood in the M-step
  - **Expectation Conditional Maximization (ECM)**: Breaks the M-step into several conditional maximization steps
- **Practical Considerations**
  - Multiple runs with different initializations can help in finding better solutions
  - Monitoring the increase in likelihood can help in determining convergence