MSML610: Advanced Machine Learning

Reasoning Over Time

Instructor: Dr. GP Saggese - gsaggese@umd.edu

References:

- AIMA 14: Probabilistic reasoning over time
- https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python

- ***Reasoning Over Time***
    - Definitions
    - Defining Temporal Inference Tasks
    - Solving Temporal Inference Tasks
- HMMs
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
- Multivariate Kalman Filters
- Dynamic Bayesian networks
- State Space Model
- Variational Inference

- Reasoning Over Time
    - ***Definitions***
    - Defining Temporal Inference Tasks
    - Solving Temporal Inference Tasks
- HMMs
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
- Multivariate Kalman Filters
- Dynamic Bayesian networks
- State Space Model
- Variational Inference

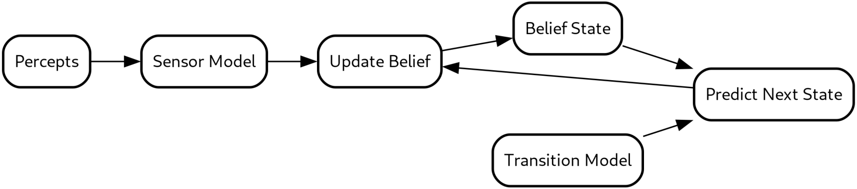
Reference

- AIMA: 14

Static vs Dynamic Probabilistic Reasoning

- **Static probabilistic reasoning**
    - Random variables have a 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)

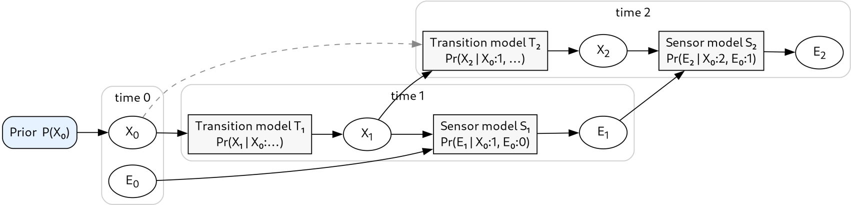
Agents in Partially Observable Environments



- Agents in partially observable environments track the current state using transition model and sensor information
  1. **Belief state**
     - Store possible world states
     - 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

Agent: Model Components

1. **State of the world**: $\underline{\boldsymbol{X}}_t$
   - Typically not observable directly
2. **Prior probability of the state** at time 0: $\underline{\boldsymbol{X}}_0$
3. **Evidence variables**: $\underline{\boldsymbol{E}}_t$
   - Observable
4. **Transition model**: $\Pr(\underline{\boldsymbol{X}}_t | \underline{\boldsymbol{X}}_{0:t-1})$
   - Models how the world evolves
   - Specifies the probability distribution of the state $\underline{\boldsymbol{X}}_t$, given all previous values
5. **Sensor model**: $\Pr(\underline{\boldsymbol{E}}_t | \underline{\boldsymbol{X}}_{0:t}, \underline{\boldsymbol{E}}_{0:t-1})$
   - Models how the evidence variables $\underline{\boldsymbol{E}}_t$ are generated



Discrete vs Continuous Time Models

- **Discrete time models**
  - View world as time slices ("snapshots")
    - Assume equal time intervals, equispaced samples
    - Label times $t = 0, 1, 2, \ldots$
  - Each slice contains random variables:
    - Hidden RVs (e.g., $\underline{\boldsymbol{X}}_t$)
    - Observable RVs (e.g., $\underline{\boldsymbol{E}}_t$)
    - $\underline{\boldsymbol{X}}_{a:b}$ represents variables in $[a, b]$
- **Continuous time models**
  - Model uncertainty over continuous time with stochastic differential equations (SDEs)
  - Discrete time models approximate SDEs

Markov Property

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

$$\Pr(\underline{\boldsymbol{X}}_t|history) \triangleq \Pr(\underline{\boldsymbol{X}}_t|\underline{\boldsymbol{X}}_{0:t-1}) = \Pr(\underline{\boldsymbol{X}}_t|\underline{\boldsymbol{X}}_0, \underline{\boldsymbol{X}}_1, \ldots, \underline{\boldsymbol{X}}_{t-1})$$

  - Of course, there can't be dependency from the future $\underline{\boldsymbol{X}}_{t+k}$ $k > 1$
- **Markov property**: current state depends (conditionally) only on a finite fixed number of k previous states:

$$\Pr(\underline{\boldsymbol{X}}_t|\underline{\boldsymbol{X}}_{0:t-1}) = \Pr(\underline{\boldsymbol{X}}_t|{\color{red}\underline{\boldsymbol{X}}_0, \underline{\boldsymbol{X}}_1, \ldots, \underline{\boldsymbol{X}}_{t-k-1}}, {\color{blue}\underline{\boldsymbol{X}}_{t-k}, \ldots, \underline{\boldsymbol{X}}_{t-1}}) = \Pr(\underline{\boldsymbol{X}}_t|{\color{blue}\underline{\boldsymbol{X}}_{t-k:t-}}$$

Markov Process

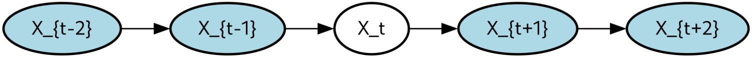
- **Markov processes** (aka Markov chains) have the Markov property

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

- **First-order Markov process**: current state $\underline{\boldsymbol{X}}_t$ depends only on the previous state $\underline{\boldsymbol{X}}_{t-1}$:

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

  - The next state depends only on the previous state, not the full history
  - The system "forgets" everything except the immediate last state
  - Bayesian network for a first-order Markov process:



  - E.g., probability of rain today depends only on yesterday, $\Pr(R_t | R_{t-1}) \; \forall t$

- **Second-order Markov process**: current state $\underline{\boldsymbol{X}}_t$ depends only on $\underline{\boldsymbol{X}}_{t-1}$ and $\underline{\boldsymbol{X}}_{t-2}$

Time-Homogeneous Process

- Even with the Markov assumption, there in an infinite number of probability distributions $\Pr(\underline{\boldsymbol{X}}_t|\underline{\boldsymbol{X}}_{t-1:t-k})$, one for each t
- **Time-homogeneous** (aka stationarity): probability remains constant by translation over t

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

  - Even if process evolves, governing laws remain unchanged
  - E.g., in the real-world, most physical laws are constant

First-Order Time-Homogeneous Process

- First-order time-homogeneous:
  - **First-order Markov property**:

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

  - **Time-homogeneous**:

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

- Putting both properties together, one conditional probability table suffices:

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

  - E.g., rain probability for today depends only on yesterday and is constant: $\Pr(R_t | R_{t-1}) = f(R_{t-1}) \; \forall t$

Sensor Model

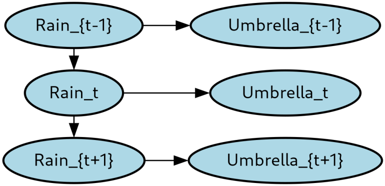
- Aka "observation model"
- In general, evidence variables $\underline{\boldsymbol{E}}_t$ depend on:
  - Previous state of the world $\underline{\boldsymbol{X}}_{0:t}$
  - Previous sensor values $\underline{\boldsymbol{E}}_{0:t-1}$

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

- **Sensor Markov property**
  - Assume sensor value $\underline{\boldsymbol{E}}_t$ depends only on current state $\underline{\boldsymbol{X}}_t$, 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 $\underline{\boldsymbol{X}}_t$ and $\underline{\boldsymbol{E}}_t$ are contemporaneous, the arrow goes from $\underline{\boldsymbol{X}}_t \rightarrow \underline{\boldsymbol{E}}_t$ since the world causes the sensor to take on particular values

Sensor Model: Rain Example

- In a Bayesian network, $\underline{\boldsymbol{X}}_t \rightarrow \underline{\boldsymbol{E}}_t$ as the world causes the sensor to take specific values
  - E.g., $Rain_t \rightarrow Umbrella_t$, since rain "causes" the umbrella to appear
- Inference goes the other direction: *"see the umbrella, guess if it's raining"*
- E.g.,
  - The transition model is $\Pr(Rain_t | Rain_{t-1})$
    - $\Pr(R_t | R_{t-1} = T) = 0.7$
    - $\Pr(R_t | R_{t-1} = F) = 0.4$
    - The sum doesn't have to be 1 since it's a conditional probability
  - The sensor model is $\Pr(Umbrella_t | Rain_t)$
    - $\Pr(U_t | R_t = T) = 0.9$ (people forget the umbrella)
    - $\Pr(U_t | R_t = F) = 0.2$ (people are paranoid)



Prior Probability

- Complete system specification needs the prior probability of the state variables at initial time $\Pr(\underline{\boldsymbol{X}}_0)$
  - Represents initial belief about system state before observations
  - Crucial for initializing state estimation process
- E.g.,
  - $\underline{\boldsymbol{X}}_0$ represents position and velocity of a moving object
  - $\Pr(\underline{\boldsymbol{X}}_0)$ could be a Gaussian distribution centered around an initial guess of object's position and velocity with uncertainty

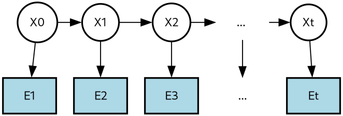
First-Order Markov Process: Joint Distribution

- 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, i.e., $\Pr(\underline{\boldsymbol{X}}_{0:t}, \underline{\boldsymbol{E}}_{1:t})$
  - Express the joint distribution of n random variables using the chain rule:

$$\Pr(Y_1, ..., Y_n) = \prod_{i=1}^{n} \Pr(Y_i | Y_{0:i-1})$$

  - Bayesian networks factorize joint distribution according to graph dependencies

$$\Pr(Y_1, ..., Y_n) = \prod_{i=1}^{n} \Pr(Y_i | \text{parents}(Y_i))$$



- First-order Markov assumption:

$$\Pr(\underline{\boldsymbol{X}}_i | \underline{\boldsymbol{X}}_{0:i-1}) = \Pr(\underline{\boldsymbol{X}}_i | \underline{\boldsymbol{X}}_{i-1})$$

- First-order Markov sensor model:

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

First-Order Markov Process: Intuition

- Putting everything together, the joint distribution probability for a time-homogeneous first-order Markov process:

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

- **Remarks**:
  - 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**: Infinite t, even assuming the Markov property

Improving Approximation of Real-World Systems

- Is first-order Markov process a **reasonable approximation of reality**?
  - 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 number 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

- Reasoning Over Time
    - Definitions
    - ***Defining Temporal Inference Tasks***
    - Solving Temporal Inference Tasks
- HMMs
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
- Multivariate Kalman Filters
- Dynamic Bayesian networks
- State Space Model
- Variational Inference

Inference Tasks in Temporal Models

- There are several possible applications

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

- Let's consider each of these applications in details

Task 1: Filtering

- **Filtering** (aka "state estimation") computes the posterior distribution of the *current 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 needed by a rational agent to track the current state of the world:
  - Agent believes current state $\Pr(\underline{\boldsymbol{X}}_{t-1})$ at time $t-1$
  - New evidence $\underline{\boldsymbol{e}}_t$ arrives for time t
  - Agent updates belief about current state $\Pr(\underline{\boldsymbol{X}}_t)$ at time t

- The term "filtering" refers to 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 rational agents evaluate actions based on expected outcomes

Task 3: Smoothing

- **Smoothing** compute posterior distribution over a *past state* given *all 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 not the state
  - Smoothing provides a better state estimate by incorporating more future evidence
  - E.g., compute the probability it rained last Wednesday, given all observations up to today
- The term "smoothing" refers to the state estimate being 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 benefits from smoothing rather than filtering for better state estimates
    - Smoothing uses all data to estimate states, leading to more accurate models
    - E.g., in weather prediction, smoothing uses past, present, and future data to better estimate current weather state

- Reasoning Over Time
    - Definitions
    - Defining Temporal Inference Tasks
    - ***Solving Temporal Inference Tasks***
- HMMs
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
- Multivariate Kalman Filters
- Dynamic Bayesian networks
- State Space Model
- Variational Inference

Solving Task 1: Filtering

- **Filtering** computes the posterior distribution of the *current state* given *all evidence to date*, i.e., $\Pr(\underline{\boldsymbol{X}}_t | \underline{\boldsymbol{E}}_{1:t} = \underline{\boldsymbol{e}}_{1:t})$
- 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 $\underline{\boldsymbol{e}}_{t+1}$
  - Instead of recomputing each state by going over the entire history of the percepts
  - Aka "recursive state estimation"

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

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

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

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

- It has the expected form:

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

Recursive Filtering: Update Formula

- The update formula for the state is:

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

- 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: $\Pr(x_t|e_{1:t})$

Recursive Filtering: Intuition

- Recursive state estimation updates the state belief as new evidence arrives

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

  in **two steps**

  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

Forward update

- We achieved:

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

- 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

Solving Task 2: Prediction

- Prediction is equivalent to filtering without updating the state with new evidence, since there is no 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

Solving Task 3: Smoothing

- You 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
- **Example**
  - 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 yesterday 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:

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

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

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

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")
    - the sensor model (i.e., "people often forget the umbrella")
- Mathematically:

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

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:**
  - In speech recognition, find the most likely word sequence behind a noisy audio signal
    - Smoothing: Best guess per time step (may miss non-English words or suboptimal sequence)
    - Viterbi: Best overall path (maximizes joint probability of the entire sequence)

Viterbi Algorithm: Intuition

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

1. Initialization
   - At $t = 1$, estimate probability of starting in each state using initial state distribution and observation likelihood
2. Recursion via dynamic programming
   - For each $t > 1$, for each state x_t:
   - Compute 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 probability and corresponding back-pointer to x_{t-1}
3. Termination and backtrace
   - At final time $t = T$, identify state with highest final probability
   - Trace back through stored pointers to reconstruct optimal path

Viterbi Algorithm: Example 1/2

- You observe a friend carrying an umbrella over 3 days
  - $\mathit{Umbrella} = [\mathit{Yes}, \mathit{Yes}, \mathit{No}]$
- 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:

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

  - Observation (Emission) Probabilities:

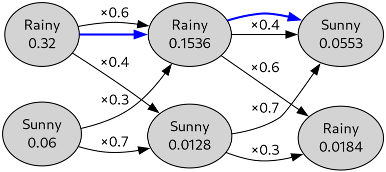
$$\begin{aligned} \Pr(\text{Yes}|\text{Sunny}) = 0.1, &\quad \Pr(\text{No}|\text{Sunny}) = 0.9 \\ \Pr(\text{Yes}|\text{Rainy}) = 0.8, &\quad \Pr(\text{No}|\text{Rainy}) = 0.2 \end{aligned}$$

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



- Reasoning Over Time
- ***HMMs***
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
- Multivariate Kalman Filters
- Dynamic Bayesian networks
- State Space Model
- Variational Inference

Algorithms for Specific Models

- **General temporal probabilistic reasoning** makes minimal assumptions:
  - Markov property for transitions
  - Markov property for sensor model
  - No constraints on:
    - Mathematical form of transition/sensor models
    - Nature of state and evidence variables (discrete or continuous)
- Improve efficiency and accuracy 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 Viterbi, forward-backward, etc
  - **Kalman Filters**:
    - 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: Formulation

- **Hidden Markov Model (HMM)**: Temporal model with simplified structure for efficiency
  - **State model**:
    - System state at time t is a discrete random variable $X_t \in \{1, \ldots, S\}$
    - E.g., in umbrella world, $X_t = Rain_t$ with states $\{Rain, Sunny\}$
    - Can combine multiple variables into one “mega-state” variable
  - **Transition model** $\Pr(X_t|X_{t-1})$:
    - 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 or diagonal matrix $\underline{\boldsymbol{O}}$
    - No assumptions about number or type (discrete/continuous) of observation variables
- **Benefit**
  - Enables efficient algorithms like forward, backward, and Viterbi

Hidden Markov Model: 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 matrix representation all the forward / backward computations become matrix operations:

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

  - Express inference tasks (e.g., filtering, smoothing) as efficient matrix multiplication
- Specialized algorithms to improve time and space complexity

Hidden Markov Model: Algorithms

- Baum-Welch
  - Special case of Expectation-Maximization (EM) algorithm
  - Pros: Converges to local maximum of likelihood
  - Cons: Only point-estimation, no uncertainty estimation
- Viterbi
  - Finds most likely sequence of hidden states
  - Pros: Fast approximation of BW
  - Cons: Returns local optimum
- Gradient-based methods
  - Use gradient descent to optimize parameters
  - Pros: Fast
  - Cons: Needs differentiable model
  - E.g., PyTorch / TensorFlow probability
- HMM with MCMC
  - Learn posterior distribution of parameters using Bayesian inference
  - Pros: Flexible, accounts for uncertainty
  - Cons: Computationally expensive
  - E.g., PyMC

Hidden Markov Model: Applications

- HMMs model systems with hidden states producing 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 DNA regions
  - Protein structure prediction
- **Finance / economics**
  - Market regime detection: bull/bear markets, volatility regimes
  - Credit scoring: observe purchases, estimate financial health (hidden variable)

Hidden Markov Model: Applications

- **Security / anomaly detection**
  - User behavior modeling: detect anomalous login patterns or usage
  - Intrusion detection: model normal traffic to spot attacks
  - Fraud detection: identify unusual transactions
- **NLP**
  - Part-of-speech tagging: map words to syntactic roles
  - Named entity recognition: identify entities, people, places
- **Operations and process monitoring**
  - Predictive maintenance: model machine health from sensor readings
  - Process monitoring: detect deviation from normal operations
  - Customer behavior modeling: understand customer intent
- **Environmental monitoring**
  - Weather prediction: infer atmospheric state from observed variables

Hidden Markov Model: Limitations

- **Short memory**
  - Markov assumption: current state depends only on previous state
  - Inefficient for long-range dependencies
- **Predefined, fixed number of states**
  - Mis-estimating states leads to underfitting or overfitting
- **Stationarity assumption**
  - Transition and sensor probabilities constant over time
- **Use atomic representation**
  - States are labels with no internal structure
  - Hard to interpret with many states or unclear state meanings
- **Training** is computationally expensive for large datasets
  - Struggles with sparse data
- **Alternatives**
  - Bayesian networks using factored representation
  - Deep learning handles complex temporal dependencies and long-term relationships

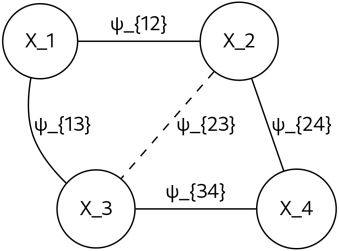
- Reasoning Over Time
- HMMs
- ***Markov Random Fields***
- Markov Logic Network
- State Space Models and Kalman Filter
- Multivariate Kalman Filters
- Dynamic Bayesian networks
- State Space Model
- Variational Inference

Markov Random Fields

- A **Markov Random Field** is a probabilistic graphical model
  - Represents a joint distribution using an undirected graph
    - Nodes = random variables
    - Edges = relationships (dependencies) between variables
  - Key idea: **Markov property**
    - Each variable is conditionally independent of non-neighbors given its neighbors
  - Model **spatial and contextual dependencies**
    - Capture local interactions that combine into a global structure
- **Example**: Image de-noising
  - Each pixel tends to have similar intensity to its neighbors
  - Noise introduces local inconsistencies
  - MRF models smoothness while respecting observed data
- **Example**: Social networks
  - Friends influence each other's behavior
  - Dependencies exist only among connected individuals

Markov Random Fields: Model Form

- $Pr(X) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(X_C)$
- X: set of all random variables in the model
- $\mathcal{C}$: set of cliques in the graph (fully connected subset of nodes)
- $\psi_C(X_C)$: **potential function** for clique C
  - Assigns a positive score to each possible configuration of variables in C
  - Clique potentials ψ_C encode preferences or constraints
  - Intuition: measures "compatibility" of values
  - High ψ_C = compatible configuration
  - Low ψ_C = unlikely configuration
- Z: **partition function**
  - Ensures probabilities sum to 1
  - Usually very hard to compute for large graphs



- Reasoning Over Time
- HMMs
- Markov Random Fields
- ***Markov Logic Network***
- State Space Models and Kalman Filter
- Multivariate Kalman Filters
- Dynamic Bayesian networks
- State Space Model
- Variational Inference

Markov Logic Networks: Intuition

- **Intuition**:
  - Logic rules are often *soft* (have exceptions) rather than *absolute* (true or false)
- Example: Social network friendships
  - Rule: “Friends of friends are likely friends”
  - Not always true, but often holds
- Example: Natural language processing
  - Rule: “Every sentence has a subject” weighted by importance
- Markov Logic Networks
  - Unify knowledge representation (logic) with uncertainty handling (probability)
  - Allow violations of rules but penalize them probabilistically
- Applications:
  - Information extraction
  - Entity resolution
  - Relational learning
- Main challenge: inference and learning are computationally expensive

Markov Logic Networks: Basics

- A **Markov Logic Network** (MLN) combines:
  - First-order logic (expressing knowledge with rules and quantifiers)
  - Markov Random Fields (modeling uncertainty with probabilities)
- Each element is a pair (F_i, w_i)
  - F_i: a first-order logic formula
    - E.g., "$Friends(x, y) \implies Similar(x, y)$"
  - w_i: a weight measuring the strength of belief in F_i
    - Higher w_i = formula more important in shaping the probability distribution
- **Semantics**:
  - An MLN defines a probability distribution over *possible worlds*
  - A world = a complete assignment of truth values to all ground atoms
  - If a world satisfies many high-weight formulas, it becomes *more probable*
- **Joint distribution**:
  - $\Pr(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right)$
  - $n_i(x)$ = number of **true groundings** of formula F_i in world x
    - Example: If F_i = "Friends(x,y) $\rightarrow$ Similar(x,y)" and in world x this holds for 7 pairs (x, y) out of 10, then $n_i(x) = 7$
- **Special cases**:
  - If all weights $w_i \rightarrow \infty$: only worlds where all formulas are satisfied have nonzero probability → recovers *classical logic*
  - If all weights are finite: allows some violations but assigns them lower probability

- Reasoning Over Time
- HMMs
- Markov Random Fields
- Markov Logic Network
- ***State Space Models and Kalman Filter***
  - g-h Filter
  - One Dimensional Kalman Filters
  - Multivariate Gaussians
- Multivariate Kalman Filters
- Dynamic Bayesian networks
- State Space Model
- Variational Inference

Tracking Objects

- Many problems can be formulated as **tracking objects**
- **Examples**
  - 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
- **Kalman filter**
  - Used for state estimation in dynamic systems with noisy, uncertain measurements
  - Track over time using predictions (model) and observations

Some Guiding Principles

- **The world is noisy**
  - E.g., a car might swerve around a pothole or brake for a pedestrian
  - E.g., wind or ice might change the car's path
- **Sensors are noisy**
  - A kitchen scale gives different readings for the same object
- **Knowledge is uncertain**
  - You alter beliefs based on evidence strength
- Use past information and system knowledge to estimate future information
  - E.g., if a car moves at a certain speed at time t, the speed at time $t+1$ is likely close to the previous speed
- Data is better than a guess, even if noisy
  - Never discard information, no matter how poor
  - E.g., two sensors, even if one is less accurate, are better than one

- Reasoning Over Time
- HMMs
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
  - ***g-h Filter***
  - One Dimensional Kalman Filters
  - Multivariate Gaussians
- Multivariate Kalman Filters
- Dynamic Bayesian networks
- State Space Model
- Variational Inference

Example of Weight: Blending Predictions and Measurements

- Imagine going to the gym to gain muscle mass
  - Estimate your weight over time
- You could:
  1. **Predict your weight**
     - Track calorie intake and energy expense
     - Compute expected weight gain
     - Cons: Difficult to track food intake and exercise accurately
  2. **Measure your weight**
     - Use a scale
     - Cons: Scale is noisy, water weight fluctuates, different clothes
- Prediction doesn't match measurements
  - At time $t-1$
    - Estimate: $\hat{x}_{t-1} = 158$
  - At time t:
    - Scale measures 164
    - Estimate $\hat{x}_{t|t-1} = 159$ based on calorie intake
- **What's your real weight?**
  - You need to blend prediction and measurement

Example of Weight: Correct Gain_Rate

- **Blend the estimates like:**

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

  - You believe the prediction is more likely correct than the measurement

- **Algorithm**

  1. Start with an initial guess
     - Assume it's correct for now
  2. Predict the next weight based on the model
  3. Measure the weight
  4. Estimate the next weight by merging values:
     - The prediction is always between the prediction and the measurement
  5. Go back to first step

Example of Weight:

- The black line is the actual weight, i.e., **ground truth**
- 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
  - Always falls between measurement and prediction
- It's not impressive since the prediction model describes the ground truth, so you don't need the measurements

Example of Weight: Learning Gain_Rate

- Consider when the model predicts a gain of -10lb/day, which is incorrect
  - Estimates diverge from measurements
- The filter needs a correct guess of the weight change rate
  - Also the rate of change can vary over time
- Solution: estimate the rate of change from measurements
  - "Data is better than a guess, even if it's noisy"
  - Refine the estimate of the gain rate:

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

- The "state" is given by `weight` and `gain_rate`, so you need to predict and update both

g-h Filter

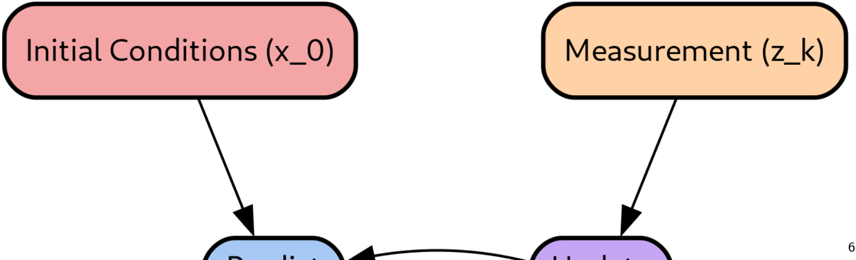
- The previous algorithm is called **g-h filter**
  - g: scaling used to blend predicted state and measurement
  - h: scaling used to update the parameter of the system model based on the measurements
- g-h filters have different values of g and h to achieve different properties
  - E.g., pick g to minimize 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 generalizations of a g-h filter

Control Theory Nomenclature

- State space models were developed in control theory, so there is a different nomenclature
- **System**: object you want to estimate/track
- **Filter**: algorithm to estimate the state of the system
- **State of the system** `x`: current values you are interested in
  - E.g., weight
  - Part of the state might be hidden (i.e., not observable)
  - You cannot observe the entire state directly, only measure it indirectly
- **Measurement** `z`: the measured value of the system
  - It is observable
  - It can be inaccurate
    - E.g., 99.3kg instead of 100kg
- **State estimate** `x_est`: 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

g-h Filter Algorithm: Pseudo-Code ::: Columns ::::

1. Initialization
   - Initialize the state of the filter
   - Initialize your belief in the state
2. Predict
   - Use system model to predict state at next time step
   - Adjust belief to account for uncertainty in prediction
3. Update
   - Get measurement and associated belief about its accuracy
   - Use as estimate of the next state a point between estimated state and measurement :::: :::: {.column width=40%}



Interpretation of g

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

Interpretation of h

- You might need to estimate some model parameters from data, e.g.,
  - The change of weight
  - The rate of change of the measurements
  - 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 significantly with respect to the time step
- **Note**: an incorrect initial state (e.g., initial value/rate of change) is similar to a changing state

- Reasoning Over Time
- HMMs
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
    - g-h Filter
    - ***One Dimensional Kalman Filters***
    - Multivariate Gaussians
- Multivariate Kalman Filters
- Dynamic Bayesian networks
- State Space Model
- Variational Inference

Updating Belief Using Gaussians

- The Bayes theorem tells that:

$$\text{posterior} = \text{normalized}(\text{prior} \times \text{likelihood})$$

- If the prior and the likelihood are Gaussian the result is also Gaussian (conjugate prior)
  - The belief and probability are represented as a Gaussian
  - We can encode the PDF in terms of mean and std dev
  - Updating belief is equivalent to sum and multiplication of Gaussians
- Algorithm:

```
# Create prior (using current estimate and system model)
prior = predict(x, process_model)

# Create likelihood (using measurement).
likelihood = gaussian(z, sensor_var)

# Update belief using prior and likelihood
posterior = update(prior, likelihood)
```

Sum of Gaussians

- The sum of two independent Gaussians

$$\mathit{Normal}(\mu_1, \sigma_1^2)$$
$$\mathit{Normal}(\mu_2, \sigma_2^2)$$

is a Gaussian $\mathit{Normal}(\mu, \sigma^2)$ with:

$$\mu = \mu_1 + \mu_2$$
$$\sigma^2 = \sigma_1^2 + \sigma_2^2$$

- The mean is the sum of the mean (by linearity)
- The variance always increases

Product of Gaussians

- The product of two independent Gaussians

$$Normal(\mu_1, \sigma_1^2)$$
$$Normal(\mu_2, \sigma_2^2)$$

is a Gaussian N(μ, σ^2) with:

$$\mu = \frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_1^2 + \sigma_2^2}$$
$$\sigma^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}$$

- **Interpretation:**
  - The variance may be reduced as more information is incorporated
  - If one Gaussian N_1 is much narrower than the other (i.e., one measure is more accurate), the result is pushed towards N_1
  - If two Gaussians are similar (i.e., two measures corroborate each other), the result becomes more certain

Kalman Gain

- Assume that:
  - x is the model prediction
  - z indicates the measurements
- The mean of the posterior is:

$$\mu = \frac{\sigma_x^2 \mu_z + \sigma_z^2 \mu_x}{\sigma_x^2 + \sigma_z^2} = \frac{\sigma_x^2}{\sigma_x^2 + \sigma_z^2} \mu_z + \frac{\sigma_z^2}{\sigma_x^2 + \sigma_z^2} \mu_x = K \mu_z + (1 - K) \mu_x$$

- The Kalman Gain K:
  - Is the scaling term that mixes the prediction and the measurement
  - Depends on the ratio of uncertainty of prior and measurement

Kalman Pseudo-Algorithm

- The typical formulation of the Kalman filter is in terms of the "orthogonal projection" approach to minimize mean squared error
    - Instead of a Bayesian formulation
- Typical symbols used in Kalman literature:
    - x: state
    - P: variance of state (uncertainty, belief)
    - $f()$: system model
    - Q: system model error
    - z: measurement
    - R: measurement noise
- **Initialization**
    - Initialize state of filter $x = x_0$
    - Initialize belief in the state $P = P_0$
- **Predict**
    - Use system model to predict state at the next time step $x = f(x)$
    - Adjust belief to account for uncertainty in prediction $P = P + Q$
- **Update**
    - Get measurement z and belief about its accuracy R
    - Compute residual between estimated state x and z: $y = z - x$
    - Compute scaling factor (Kalman K) based on accuracy of prediction P and measurement R

- Reasoning Over Time
- HMMs
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
  - g-h Filter
  - One Dimensional Kalman Filters
  - ***Multivariate Gaussians***
- Multivariate Kalman Filters
- Dynamic Bayesian networks
- State Space Model
- Variational Inference

Multivariate State

- Often the state variable is multivariate, e.g.,
  - Position and velocity of a dog (probably uncorrelated)
  - Height and weight of an adult (correlated)
- **Variance** is a measure of how a population varies, e.g.,
  - Variance $= 0$ means constant
  - Large variance means lots of variation
- **Covariance** are correlated variances
  - E.g., height and weight are generally positively correlated
- **Covariance matrix**
  - The diagonal contains the variance for each variable
  - The off-diagonal elements contain the covariance between i and j variables
  - The covariance matrix is symmetric
- Correlation allows prediction
  - E.g., "as winter comes you predict you will spend more on heating your house"

Multivariate Gaussian

- The marginal of a multivariate Gaussian is 1-d Gaussian
- Consider a contour plot (i.e., the intersection of a 2-d Gaussian $z = f(x, y)$ with a plane $z = c$)
    - The contour plot is always an ellipses

Multiplying Two Multivariate Gaussians

- Given two multivariate Gaussians $\sim \mathit{Normal}(\underline{\boldsymbol{\mu}}_i, \underline{\underline{\boldsymbol{\Sigma}}}_i)$
- The product of the Gaussians is still Gaussian $\sim \mathit{Normal}(\underline{\boldsymbol{\mu}}, \underline{\underline{\boldsymbol{\Sigma}}})$

$$\begin{aligned}
\underline{\boldsymbol{\mu}} &= \underline{\underline{\boldsymbol{\Sigma}}}_2(\underline{\underline{\boldsymbol{\Sigma}}}_1 + \underline{\underline{\boldsymbol{\Sigma}}}_2)^{-1}\underline{\boldsymbol{\mu}}_1 + \underline{\underline{\boldsymbol{\Sigma}}}_1(\underline{\underline{\boldsymbol{\Sigma}}}_1 + \underline{\underline{\boldsymbol{\Sigma}}}_2)^{-1}\underline{\boldsymbol{\mu}}_2 \\
\underline{\underline{\boldsymbol{\Sigma}}} &= \underline{\underline{\boldsymbol{\Sigma}}}_1(\underline{\underline{\boldsymbol{\Sigma}}}_1 + \underline{\underline{\boldsymbol{\Sigma}}}_2)^{-1}\underline{\underline{\boldsymbol{\Sigma}}}_2
\end{aligned}$$

- **Note**: this is a generalization of the 1-d case to multivariate

$$\begin{aligned}
\mu &= \frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \\
\sigma^2 &= \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}
\end{aligned}$$

replacing:

- σ^2 with covariance matrix $\underline{\underline{\boldsymbol{\Sigma}}}$
- Division with matrix inversion

Multivariate Filtering

- Covariance structure helps improve the estimate, e.g.,
  - You know an airplane direction can't change quickly
  - Knowing an approximate value for the velocity helps constrain possible next positions
- **E.g., airplane**
  - You are tracking a plane moving in a direction (1-d problem)
  - At time 1, you are fairly certain about the position $x = 0$, but you don't know the velocity
    - You plot position and velocity on an x-y plane
    - The covariance matrix between position and velocity is narrow and tall
    - It is narrow on the x-axis since you know that the position is around $x = 0$
    - It is tall on the y-axis because of your lack of knowledge about velocity
  - After 1 sec, you get a position update of $x = 5$
    - You can infer that the velocity is 5/s
    - The covariance matrix is then stretched diagonally

- Reasoning Over Time
- HMMs
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
- ***Multivariate Kalman Filters***
    - Tracking a Dog with a Kalman Filter
    - Non-Linear Filtering
- Dynamic Bayesian networks
- State Space Model
- Variational Inference

Notation

- A Bayesian notation $a|b$ means "a given the evidence of b"
  - The prior is $\underline{\hat{\boldsymbol{x}}}_{t|t-1}$, since you know only the information at time $t-1$, i.e., the previous state
  - The posterior is $\underline{\hat{\boldsymbol{x}}}_{t|t}$, since you know all the information at time t, i.e., the measurement
- **A simpler notation:**
  - Indicate the "prior" version of the variables (i.e., after the system update) with an overline (E.g., $\overline{x}$, $\overline{\boldsymbol{x}}$, $\overline{\boldsymbol{X}}$)
  - Omit the indices $t+1$ and t and use an assignment notation (representing "update in place" of a variable):

$$x = x + 1$$

    instead of the mathematical notation using a different variable for each time step:

$$x_{t+1} = x_t + 1$$

  - With this notation:
    - The prior is $\overline{x} = \hat{x}_{t|t-1}$
    - The posterior is $x = \hat{x}_{t|t}$

Multivariate Kalman Filter

- With the previous notation:
  - State update: $\overline{\boldsymbol{x}} = \boldsymbol{F}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}$
  - State uncertainty: $\overline{\boldsymbol{P}} = \boldsymbol{F}\boldsymbol{P}\boldsymbol{F}^T + \boldsymbol{Q}$
  - Residual: $\boldsymbol{y} = \boldsymbol{z} - \boldsymbol{H}\overline{\boldsymbol{x}}$
  - Kalman gain: $\boldsymbol{K} = \overline{\boldsymbol{P}}\boldsymbol{H}^T(\boldsymbol{H}\overline{\boldsymbol{P}}\boldsymbol{H}^T + \boldsymbol{R})^{-1}$
  - Updated state: $\boldsymbol{x} = \overline{\boldsymbol{x}} + \boldsymbol{K}\boldsymbol{y}$
  - Update state uncertainty: $\boldsymbol{P} = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})\overline{\boldsymbol{P}}$
- Where
  - $\boldsymbol{x}$ and $\boldsymbol{P}$ are the state mean and covariance
  - $\boldsymbol{F}$ is the state transition function
  - $\boldsymbol{Q}$ is the system error (i.e., the noise in the model assessment)
  - $\boldsymbol{B}$ and $\boldsymbol{u}$ model the control inputs to the system
  - $\boldsymbol{H}$ is the measurement function
  - $\boldsymbol{z}$ and $\boldsymbol{R}$ are the measurement mean and covariance
  - $\boldsymbol{y}$ is the residual
  - $\boldsymbol{K}$ is the Kalman gain
- Use the system model to predict the next state
  - When we multiply $\boldsymbol{F}$ to $\boldsymbol{x}$ we get the prior (i.e., the state before seeing any measurement)
- Form an estimate between the prior and the measurement

From Univariate to Multivariate Kalman Filter

- Let's compare

Definition	**Univariate (Bayesian)**	**Univariate (Kalman)**	**Multivariate (Kalman)**
State update	$\overline{\mu} = \mu + \mu_f$	$\overline{x} = x + dx$	$\overline{\boldsymbol{x}} = \boldsymbol{F}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}$
State uncertainty	$\overline{\sigma}^2 = \sigma^2 + \sigma_f^2$	$\overline{P} = P + Q$	$\overline{\boldsymbol{P}} = \boldsymbol{F}\boldsymbol{P}\boldsymbol{F}^T + \boldsymbol{Q}$
Residual		$y = z - \overline{x}$	$\boldsymbol{y} = \boldsymbol{z} - \boldsymbol{H}\overline{\boldsymbol{x}}$
Kalman gain		$K = \frac{\overline{P}}{\overline{P}+R}$	$\boldsymbol{K} = \overline{\boldsymbol{P}}\boldsymbol{H}^T(\boldsymbol{H}\overline{\boldsymbol{P}}\boldsymbol{H}^T + \boldsymbol{R})^{-1}$
Updated state	$\hat{\mu} = \frac{\overline{\sigma}^2\mu_z + \sigma_z^2\overline{\mu}}{\overline{\sigma}^2 + \sigma_z^2}$	$x = \overline{x} + Ky$	$\boldsymbol{x} = \overline{\boldsymbol{x}} + \boldsymbol{K}\boldsymbol{y}$
Upd. state uncertainty	$\sigma^2 = \frac{\overline{\sigma}^2\sigma_z^2}{\overline{\sigma}^2 + \sigma_z^2}$	$P = (1 - K)\overline{P}$	$\boldsymbol{P} = (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})\overline{\boldsymbol{P}}$

Designing a Kalman filter

- The designer of the model needs to design:
  - The form of the state $\underline{\boldsymbol{x}}$ and $\underline{\underline{\boldsymbol{P}}}$
  - The system model $\underline{\underline{\boldsymbol{F}}}$ and $\underline{\underline{\boldsymbol{Q}}}$
  - The measurement $\underline{\boldsymbol{z}}$ and $\underline{\underline{\boldsymbol{R}}}$
  - The measurement function $\underline{\underline{\boldsymbol{H}}}$
  - The control inputs $\underline{\underline{\boldsymbol{B}}}$ and $\underline{\boldsymbol{u}}$ if there are control inputs

- Reasoning Over Time
- HMMs
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
- Multivariate Kalman Filters
  - ***Tracking a Dog with a Kalman Filter***
  - Non-Linear Filtering
- Dynamic Bayesian networks
- State Space Model
- Variational Inference

Tracking 1D Dog: Problem formulation

- There is a dog moving on a 1-d track
- The dog moves approximately 1 meter per step
  - The velocity has variance due to noise/imperfect model specification
- There is a sensor that measures the position of the dog
  - The sensor has a certain error
- Time is discrete

(Nuvolo)

Tracking Dog: Predict Step

- At each step, the position is described with a Gaussian distribution $\mathit{Normal}(\mu, \sigma^2)$
- The position is part of the system's state, along with the velocity
    - The position is "observed" by a sensor
    - The velocity is a "hidden" variable
    - You could use more variables (E.g., acceleration, jerk, etc.)

Tracking Dog: Design State Covariance

- Initialize variances to reasonable values
  - E.g., $\sigma_{position} = 500m$ due to uncertainty about initial position
  - Top speed for a dog is 21m/s, so set $3\sigma_{velocity} = 21$
  - Assume covariances to be zero due to unknown initial correlation between position and velocity
  - $\underline{\underline{\boldsymbol{P}}}$ is diagonal

Tracking Dog: Design System Model

- Describe mathematically the behavior of the system

$$x_{t+1} = x_t + v\Delta t$$

- No model to predict how dog velocity changes over time
  - Assume it remains constant

$$\dot{x}_{t+1} = \dot{x}_t$$

  - This is not correct, but if velocity doesn't change much, the filter will perform well
- Put the model in matrix form $\underline{\boldsymbol{x}}_{t+1} = \underline{\underline{\boldsymbol{F}}}\underline{\boldsymbol{x}}_t$

Tracking Dog: Predicting the System

- If we predict the system without measurements:
  - The state follows the system model
  - The state uncertainty grows
    - This is true even without system error (noise)

Tracking Dog: Design System Noise

- Consider a car driving on a road with cruise control on
- It should travel at constant speed:

$$x_t = \dot{x}_{t-1}\Delta t + x_{t-1}$$

- In reality, it is affected by unknown factors:
  - The cruise control is not perfect
  - Wind, hills, potholes affect the car
  - Passengers roll down windows, changing the drag profile of the car
- Model this as:

$$\dot{x}_t = \dot{x}_{t-1} + w$$

- Model all of this with a covariance matrix $\underline{\underline{\boldsymbol{Q}}} = \mathbb{E}[\underline{\boldsymbol{w}} \cdot \underline{\boldsymbol{w}}^T]$:
  - Assume the noise is iid, has zero mean, and is independent from the system
  - For these reasons, you don't have to change the position, only the velocity

Tracking Dog: Design the Control Function

- Incorporate control inputs to predict state based on this information

$$\Delta \underline{\bar{\boldsymbol{x}}} = \underline{\underline{\boldsymbol{B}}}\,\underline{\boldsymbol{u}}$$

- E.g., in the case of the car
  - Steering
  - Acceleration
- E.g., in the case of the dog, control inputs can be
  - The voice of its master
  - Seeing a squirrel

Tracking Dog: Design the Measurement Function

- Kalman filter computes the update step in the measurement space
- If the measurement is in the same units as the state, the residual is simple to compute:

$$\text{residual} = \text{measured position - predicted position}$$

- E.g., assume we are tracking the position of the dog using a sensor that outputs a voltage
  - We cannot compute the residual as:

    measure voltage - predicted position

  - We need to convert the position into voltage
- The Kalman space allows to have a measurement matrix $\underline{\underline{\boldsymbol{H}}}$ to convert the state into a measurement

$$\underline{\boldsymbol{y}} = \underline{\boldsymbol{z}} - \underline{\underline{\boldsymbol{H}}}\overline{\underline{\boldsymbol{x}}}$$

Why Working in Measurement and Not in State Space?

- The problem is that it is possible to convert state into measurement, but not vice versa because of the hidden variables
  - E.g., transform position (discarding velocity) into voltage
  - If the sensor doesn't read velocity how do we estimate the measured velocity

Tracking Dog: Design the Measurement

- Typically $\underline{\boldsymbol{z}}$ is easy since it just contains the measurements from the sensor
- The measurement noise matrix $\underline{\underline{\boldsymbol{R}}}$ can be difficult to estimate
    - Noise can be not Gaussian
    - There can be a bias in the sensor
    - The error can be not symmetrical (e.g., temperature sensor is less precise as the temperature increases)

- Reasoning Over Time
- HMMs
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
- Multivariate Kalman Filters
    - Tracking a Dog with a Kalman Filter
    - ***Non-Linear Filtering***
- Dynamic Bayesian networks
- State Space Model
- Variational Inference

Optimality

- **Assumptions:**
  - Everything is linear
  - System and sensor noise is Gaussian
- Under these assumptions, the Kalman filter is optimal in a least square sense
- The Kalman filter is a mathematical model of the world
  - The output is only as accurate as the model of the world

The World Is Non-Linear

- The Kalman filter uses linear equations and can only handle linear problems
- **The world is non-linear:**
    - System model can be non-linear:
        - Many physical systems are described by non-linear differential equations
        - E.g., a ball flying through air is affected by drag, leading to non-linear behaviors
    - Measurements can be non-linear:
        - To measure the height on a plane, you can measure the distance of the plane from the radar. Given the Pythagorean theorem, you get:

$$x = \sqrt{\text{dist}^2 - \text{height}^2}$$

- Rarely does a physical system have equations that can be solved analytically

Extended Kalman Filter

- Aka EKF
- EKF is a nonlinear version of the Kalman filter
  - Linearize the differential equations to compute the Jacobian (i.e., matrix of partial derivatives) at the point of the current estimate
  - Used for estimating the state of a nonlinear dynamic system
- Pros
  - Use the linear Kalman machinery
- Cons:
  - Analytical solution:
    - Difficult or impossible
  - Numerical solution:
    - Expensive computationally
    - Errors can compound forcing the filter to diverge (unstable)

Unscented Kalman Filter

- Aka UKF
- It is superior to EKF in almost every way

Intuition of Sampling Techniques

- Assume you have a distribution X and a non-linearity ϕ
- For every measurement:
  - Generate many points from X
  - Pass them through the non-linear function ϕ
  - Approximate the result (E.g., compute mean and variance)
- **Problem**:
  - "How many points are needed to build an accurate output distribution"?
  - Even if n=500,000 points are enough for 1 dimension, for k dimensions you might need n^k points (curse of dimensionality)

Unscented Transform

- Unscented transform estimates the result of applying a non-linear transformation to a probability distribution characterized by a finite number of moments (e.g., mean and covariance)
  - E.g., compute the non-linear transform of a distribution, given mean and covariance estimate
  - Called “unscented” since “it doesn’t stink.”
- **Intuition**
  - Given a PDF C with mean $\underline{\boldsymbol{\mu}}$ and covariance $\underline{\underline{\boldsymbol{\Sigma}}}$
  - Encode mean and covariance in a set of points (sigma points) that represent a discrete PMF D with the same mean $\underline{\boldsymbol{\mu}}$ and covariance $\underline{\underline{\boldsymbol{\Sigma}}}$
  - Propagate the discrete PMF D by applying the non-linear function ϕ to each point of the PMF
  - The mean and covariance of $\phi(D)$ approximate the mean and covariance of $\phi(C)$

Unscented Transform: 1D Case

- The idea is that we need 3 sigma points for a 1-d Gaussian
  - One point for the mean
  - Two points around the mean
- Each point has a weight

Unscented Transform: Sigma Points

- Consider a distribution F and a non-linearity ϕ
- There are algorithms to generate points and weights (given the mean and covariance of F) to evaluate mean and covariance of F transformed through ϕ
- In n dimensions, we need $2n+1$ points $\underline{\boldsymbol{x}}_i$ and weights w_i^m, w_i^c

$$\sum_i w_i^m = 1$$

$$\sum_i w_i^c = 1$$

$$\mu(\phi) = \sum_i w_i^m \phi(\underline{\boldsymbol{x}}_i)$$

$$\Sigma(\phi) = \sum_i w_i^c (\phi(\underline{\boldsymbol{x}}_i) - \mu(\phi))(\phi(\underline{\boldsymbol{x}}_i) - \mu(\phi))^T$$

- Note that selecting the sigma points has not a single solution

Monte Carlo Sampling

- Use a finite number of randomly sampled points to represent the problem
- Run the points through the transformation (e.g., non-linear function / system you are modeling)
- Compute the results on the transformed points

Particle Filters

- Aka Sequential Monte Carlo (SMC) methods
- $=$ Monte Carlo algorithms to solve problems in Bayesian statistical inference (e.g., in filtering problems)
- The goal is to compute posterior distributions of the states, given some data

Generic particle filter algorithm

1. Randomly generate particle

- Particles have all state variable that needs to be estimated (e.g., position, velocity)
- Each particle has a weight representing the probability that it represents the actual state of the system

2. Predict next state of the particles
3. Update weighting

- Update the weighting of the particles based on the measurements
- Particles that match closely the measurements are weighted higher

4. Resample

- Discard highly improbable particle

5. Compute estimate

- Compute weighted mean and covariance of the particles to get an estimate of the state and uncertainty

- Reasoning Over Time
- HMMs
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
- Multivariate Kalman Filters
- ***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

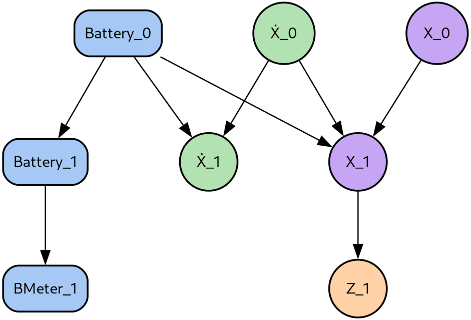
- Key components of a DBN
  - Prior distribution of state $\Pr(X_0)$
  - Transition model $\Pr(X_{t+1}|X_t)$
  - Sensor model $\Pr(E_t|X_t)$
  - Transition and sensor models are time-homogeneous
- Network topology includes:
  - Intra-slice topology
  - Inter-slice links

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:**
  - $S_t = (X_t, \dot{X}_t, \text{BatteryLevel}_t)$
  - $E_t = (\text{GPS}_t, \text{BMeter}_t)$

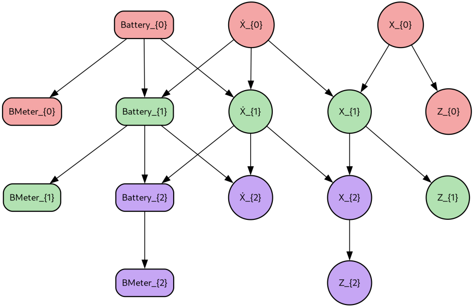
DBN Example: Tracking a Robot (2/3)

- The DBN structure models both intra-slice (within time) and inter-slice (across time) dependencies
- Intra-slice dependencies:
  - Position X_t influences velocity $\dot{X}_t$
  - $\mathrm{BatteryLevel}_t$ influences velocity $\dot{X}_t$
  - $\mathrm{Battery}_{t+1}$ depends on $\mathrm{Battery}_t$ and $\dot{X}_t$
  - BMeter_t depends on $\mathrm{Battery}_t$
  - 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 $\mathrm{Battery}_t$



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

- Reasoning Over Time
- HMMs
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
- Multivariate Kalman Filters
- Dynamic Bayesian networks
- ***State Space Model***
- Variational Inference

- Reasoning Over Time
- HMMs
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
- Multivariate Kalman Filters
- Dynamic Bayesian networks
- State Space Model
- ***Variational Inference***
  - Expectation-Maximization (EM) Algorithm

- Reasoning Over Time
- HMMs
- Markov Random Fields
- Markov Logic Network
- State Space Models and Kalman Filter
- Multivariate Kalman Filters
- 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 ($\boldsymbol{\theta}$)**: Unknown parameters to be estimated
- **Challenge Addressed**
  - Direct maximization of the likelihood function $p(\mathbf{X}|\boldsymbol{\theta})$ 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