

TEMPORAL PROBABILITY MODELS

CS194-10 FALL 2011 LECTURE 25

Outline

- ◇ Hidden variables
- ◇ Inference: filtering, prediction, smoothing
- ◇ Hidden Markov models
- ◇ Kalman filters (a brief mention)
- ◇ Dynamic Bayesian networks
- ◇ Particle filtering

Hidden variables

The underlying state of the process is usually **unobservable**;
observations at time t do not cause observations at time $t + 1$

E.g., diabetes management

\mathbf{X}_t = set of unobservable state variables at time t
e.g., *BloodSugar_t*, *StomachContents_t*, etc.

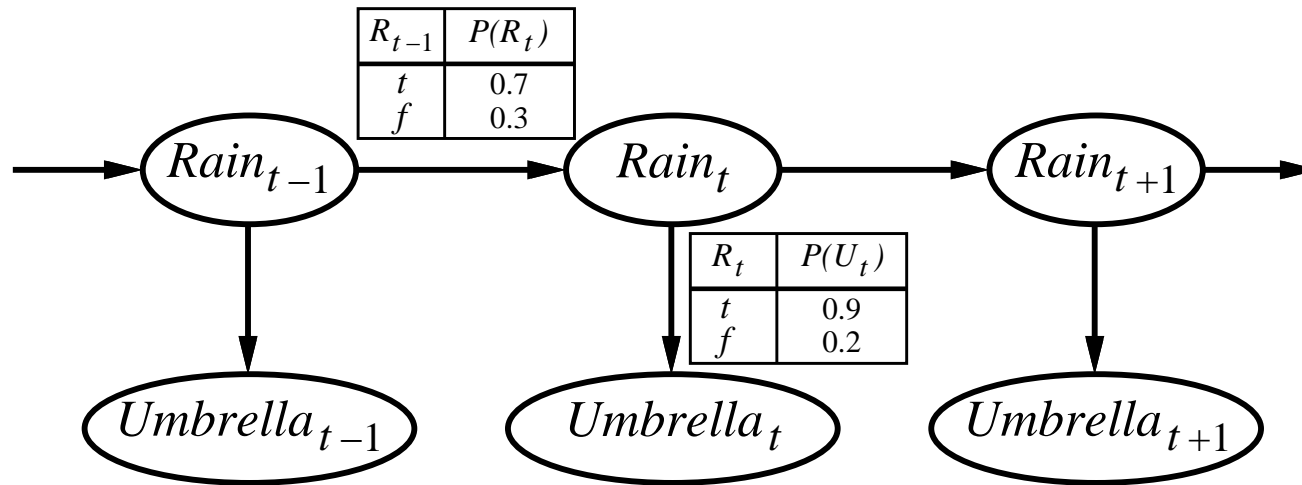
\mathbf{E}_t = set of observable evidence variables at time t
e.g., *MeasuredBloodSugar_t*, *PulseRate_t*, *FoodEaten_t*

Sensor Markov assumption: $\mathbf{P}(\mathbf{E}_t | \mathbf{X}_{0:t}, \mathbf{E}_{0:t-1}) = \mathbf{P}(\mathbf{E}_t | \mathbf{X}_t)$

This assumes **discrete time**

- step size depends on problem
- model structure depends on time step chosen

Example



First-order Markov assumption not exactly true in real world!

Possible fixes:

1. **Increase order** of Markov process
2. **Augment state**, e.g., add $Temp_t$, $Pressure_t$

Example: robot motion

- Augment position and velocity with $Battery_t$

Inference tasks

Filtering: $P(\mathbf{X}_t | \mathbf{e}_{1:t})$

belief state—input to the decision process of a rational agent

Prediction: $P(\mathbf{X}_{t+k} | \mathbf{e}_{1:t})$ for $k > 0$

evaluation of possible action sequences;
like filtering without the evidence

Smoothing: $P(\mathbf{X}_k | \mathbf{e}_{1:t})$ for $0 \leq k < t$

better estimate of past states, essential for learning

Most likely explanation: $\arg \max_{\mathbf{x}_{1:t}} P(\mathbf{x}_{1:t} | \mathbf{e}_{1:t})$

speech recognition, decoding with a noisy channel

Filtering

Aim: devise a **recursive** state estimation algorithm:

$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, \mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t}))$$

$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) &= \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}, \mathbf{e}_{t+1}) \\ &= \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}, \mathbf{e}_{1:t}) \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) \\ &= \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})\end{aligned}$$

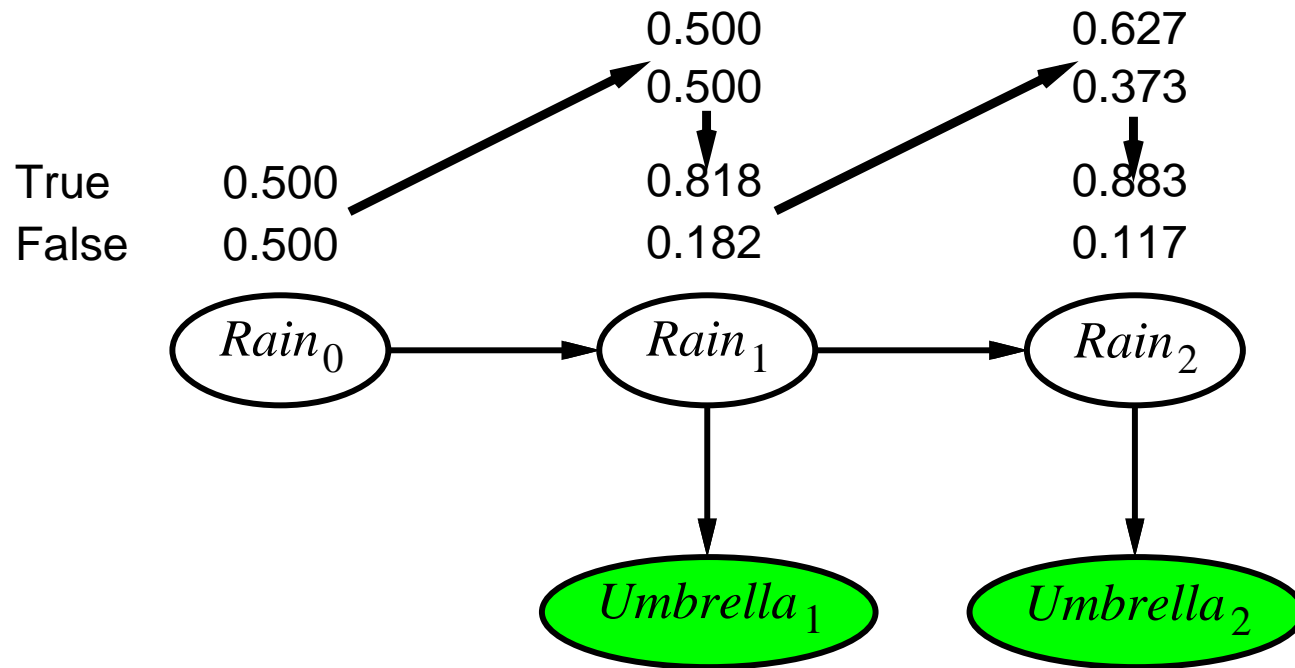
I.e., **prediction** + **estimation**. Prediction by summing out \mathbf{X}_t :

$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) &= \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, \mathbf{e}_{1:t}) P(\mathbf{x}_t|\mathbf{e}_{1:t}) \\ &= \alpha \mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t) P(\mathbf{x}_t|\mathbf{e}_{1:t})\end{aligned}$$

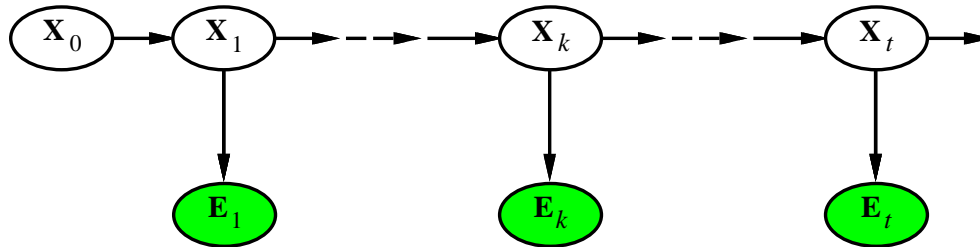
$\mathbf{f}_{1:t+1} = \text{FORWARD}(\mathbf{f}_{1:t}, \mathbf{e}_{t+1})$ where $\mathbf{f}_{1:t} = \mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$

Time and space **constant** (independent of t) for finite-state \mathbf{X}_t

Filtering example



Smoothing



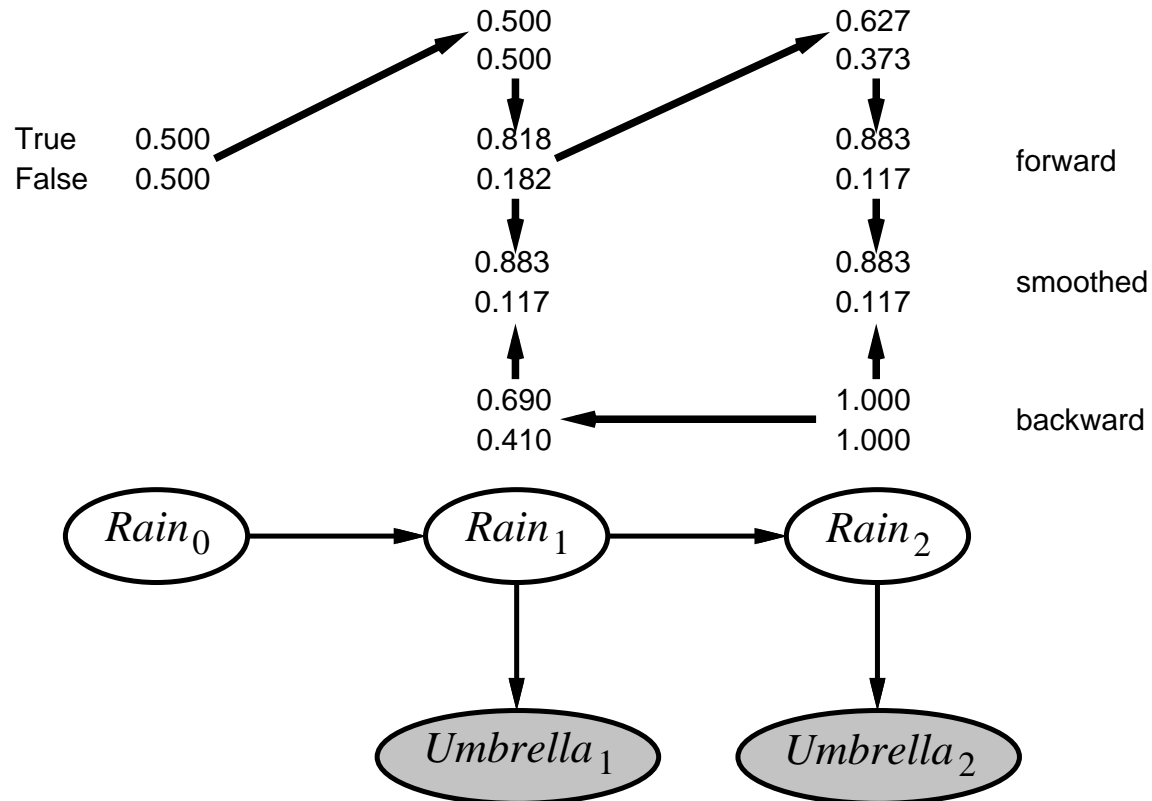
Divide evidence $\mathbf{e}_{1:t}$ into $\mathbf{e}_{1:k}$, $\mathbf{e}_{k+1:t}$:

$$\begin{aligned}
 \mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\
 &= \alpha \mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{X}_k, \mathbf{e}_{1:k}) \\
 &= \alpha \mathbf{P}(\mathbf{X}_k | \mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{X}_k) \\
 &= \alpha \mathbf{f}_{1:k} \mathbf{b}_{k+1:t}
 \end{aligned}$$

Backward message computed by a backwards recursion:

$$\begin{aligned}
 \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t} | \mathbf{X}_k, \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} | \mathbf{X}_k) \\
 &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} | \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} | \mathbf{X}_k) \\
 &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1} | \mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t} | \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} | \mathbf{X}_k)
 \end{aligned}$$

Smoothing example



Forward-backward algorithm: cache forward messages along the way
 Time linear in t (polytree inference), space $O(t|f|)$

Most likely explanation

Most likely sequence \neq sequence of most likely states!!!!

Most likely path to each \mathbf{x}_{t+1}

= most likely path to **some** \mathbf{x}_t plus one more step

$$\begin{aligned} & \max_{\mathbf{x}_1 \dots \mathbf{x}_t} \mathbf{P}(\mathbf{x}_1, \dots, \mathbf{x}_t, \mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) \\ & = \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \max_{\mathbf{x}_t} \left(\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t) \max_{\mathbf{x}_1 \dots \mathbf{x}_{t-1}} P(\mathbf{x}_1, \dots, \mathbf{x}_{t-1}, \mathbf{x}_t | \mathbf{e}_{1:t}) \right) \end{aligned}$$

Identical to filtering, except $\mathbf{f}_{1:t}$ replaced by

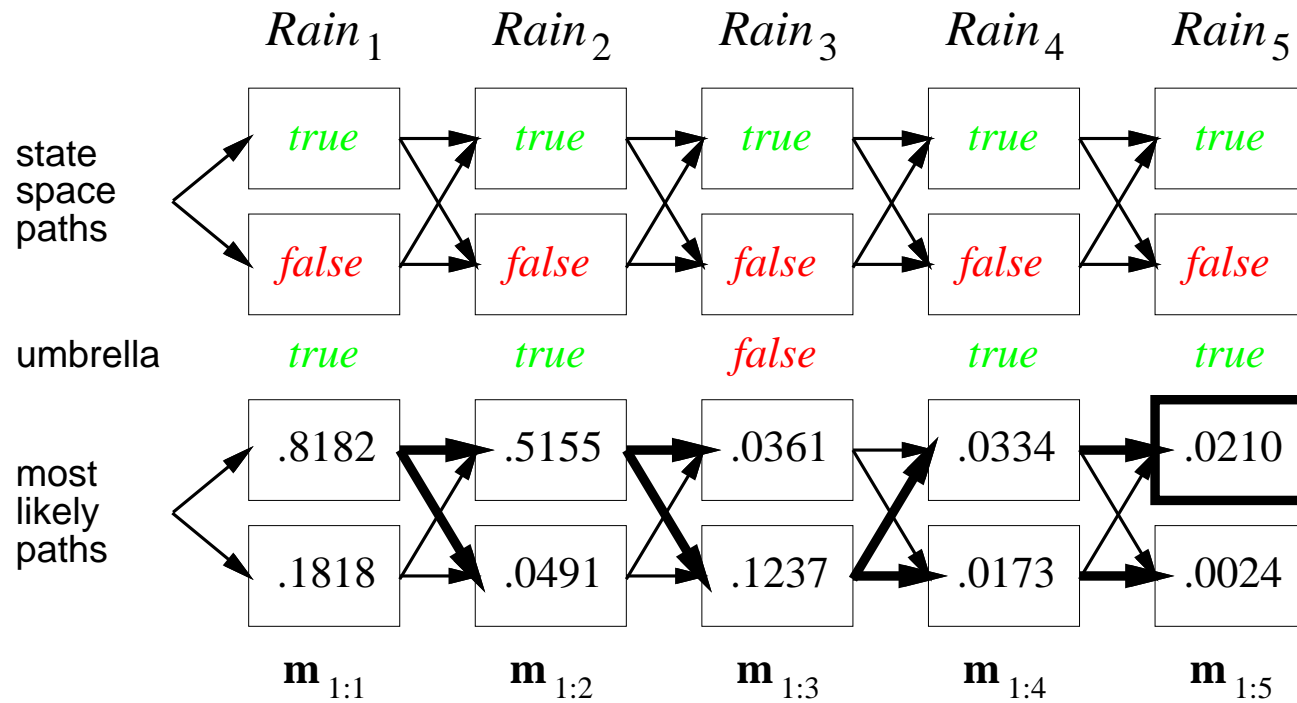
$$\mathbf{m}_{1:t} = \max_{\mathbf{x}_1 \dots \mathbf{x}_{t-1}} \mathbf{P}(\mathbf{x}_1, \dots, \mathbf{x}_{t-1}, \mathbf{X}_t | \mathbf{e}_{1:t}),$$

I.e., $\mathbf{m}_{1:t}(i)$ gives the probability of the most likely path to state i .

Update has sum replaced by max, giving the **Viterbi algorithm**:

$$\mathbf{m}_{1:t+1} = \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \max_{\mathbf{x}_t} \left(\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t) \mathbf{m}_{1:t} \right)$$

Viterbi example



Hidden Markov models (HMMs)

X_t is a single, discrete variable (often E_t is too)

Domain of X_t is $\{1, \dots, S\}$

Transition matrix $T_{ij} = P(X_t = j | X_{t-1} = i)$, e.g., $\begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$

Sensor matrix O_t for each time step, diagonal elements $P(e_t | X_t = i)$

e.g., with $U_1 = true$, $O_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}$

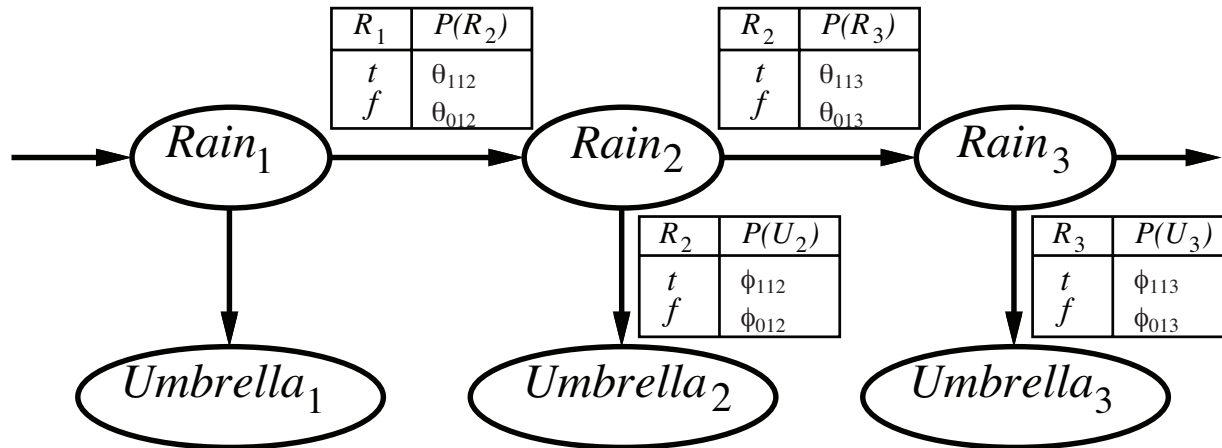
Forward and backward messages as column vectors:

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

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

Forward–backward algorithm needs time $O(S^2t)$ and space $O(St)$
(logarithmic or constant space is also possible)

Learning in HMMs



If parameters at each time step were separate, the EM update would be

$$\text{E step: } p_{ijkt} = P(X_t = k, X_{t-1} = j | \mathbf{e}^{(i)}, \boldsymbol{\theta}) \quad \forall i, j, k, t$$

$$\text{M step: } \theta'_{jkt} = \hat{N}_{jkt} / \sum_{k'} \hat{N}_{jk't} = \sum_i p_{ijkt} / \sum_i \sum_{k'} p_{ijk't}$$

Stationary parameters $\theta_{jkt} = \lambda_{jk} \forall t$, so EM \approx chain rule: $\frac{\partial L}{\partial \lambda_{jk}} = \sum_t \frac{\partial L}{\partial \theta_{jkt}} \frac{\partial \theta_{jkt}}{\partial \lambda_{jk}}$

$$\text{E step: } p_{ijkt} = P(X_t = k, X_{t-1} = j | \mathbf{e}^{(i)}, \boldsymbol{\lambda}) \quad \forall i, j, k, t$$

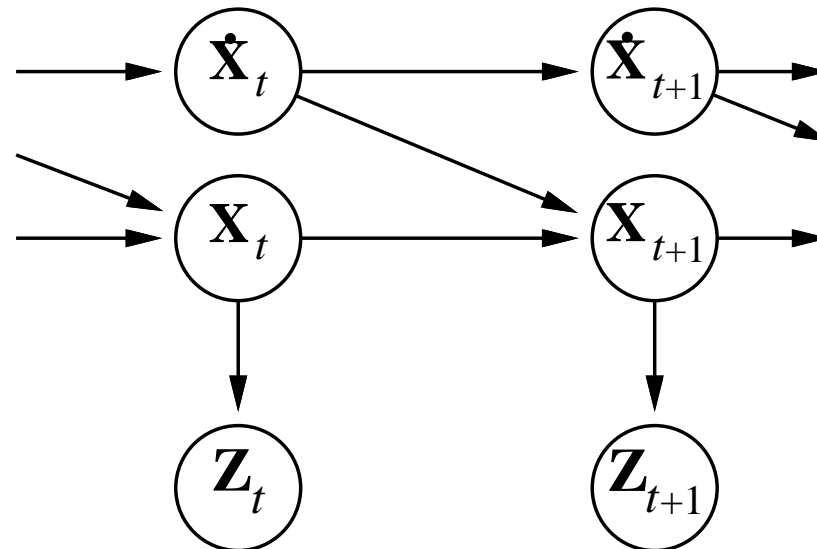
$$\text{M step: } \lambda'_{jk} = \hat{N}_{jk} / \sum_{k'} \hat{N}_{jk'} = \sum_i \sum_t p_{ijkt} / \sum_i \sum_t \sum_{k'} p_{ijk't}$$

Kalman filters

Modelling systems described by a set of continuous variables,

e.g., tracking a bird flying— $\mathbf{X}_t = X, Y, Z, \dot{X}, \dot{Y}, \dot{Z}$.

Airplanes, robots, ecosystems, economies, chemical plants, planets, ...



Gaussian prior, linear Gaussian transition model and sensor model

Updating Gaussian distributions

Prediction step: if $\mathbf{P}(\mathbf{X}_t | \mathbf{e}_{1:t})$ is Gaussian, then prediction

$$\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) = \int_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t) P(\mathbf{x}_t | \mathbf{e}_{1:t}) d\mathbf{x}_t$$

is Gaussian. If $\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t})$ is Gaussian, then the updated distribution

$$\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = \alpha \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \mathbf{P}(\mathbf{X}_{t+1} | \mathbf{e}_{1:t})$$

is Gaussian

Hence $\mathbf{P}(\mathbf{X}_t | \mathbf{e}_{1:t})$ is multivariate Gaussian $N(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ for all t

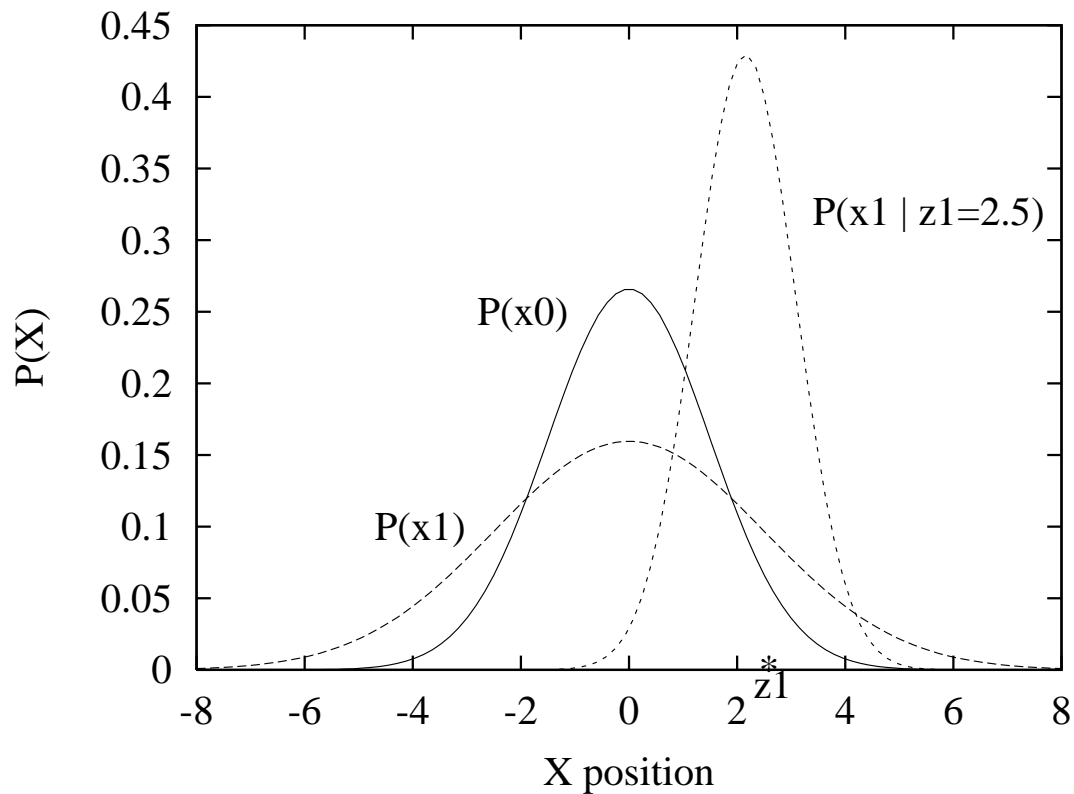
General (nonlinear, non-Gaussian) process: description of posterior grows **unboundedly** as $t \rightarrow \infty$

Simple 1-D example

Gaussian random walk on X -axis, s.d. σ_x , sensor s.d. σ_z

$$\mu_{t+1} = \frac{(\sigma_t^2 + \sigma_x^2)z_{t+1} + \sigma_z^2\mu_t}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2}$$

$$\sigma_{t+1}^2 = \frac{(\sigma_t^2 + \sigma_x^2)\sigma_z^2}{\sigma_t^2 + \sigma_x^2 + \sigma_z^2}$$



General Kalman update

Transition and sensor models:

$$P(\mathbf{x}_{t+1}|\mathbf{x}_t) = N(\mathbf{F}\mathbf{x}_t, \Sigma_x)(\mathbf{x}_{t+1})$$
$$P(\mathbf{z}_t|\mathbf{x}_t) = N(\mathbf{H}\mathbf{x}_t, \Sigma_z)(\mathbf{z}_t)$$

\mathbf{F} is the matrix for the transition; Σ_x the transition noise covariance

\mathbf{H} is the matrix for the sensors; Σ_z the sensor noise covariance

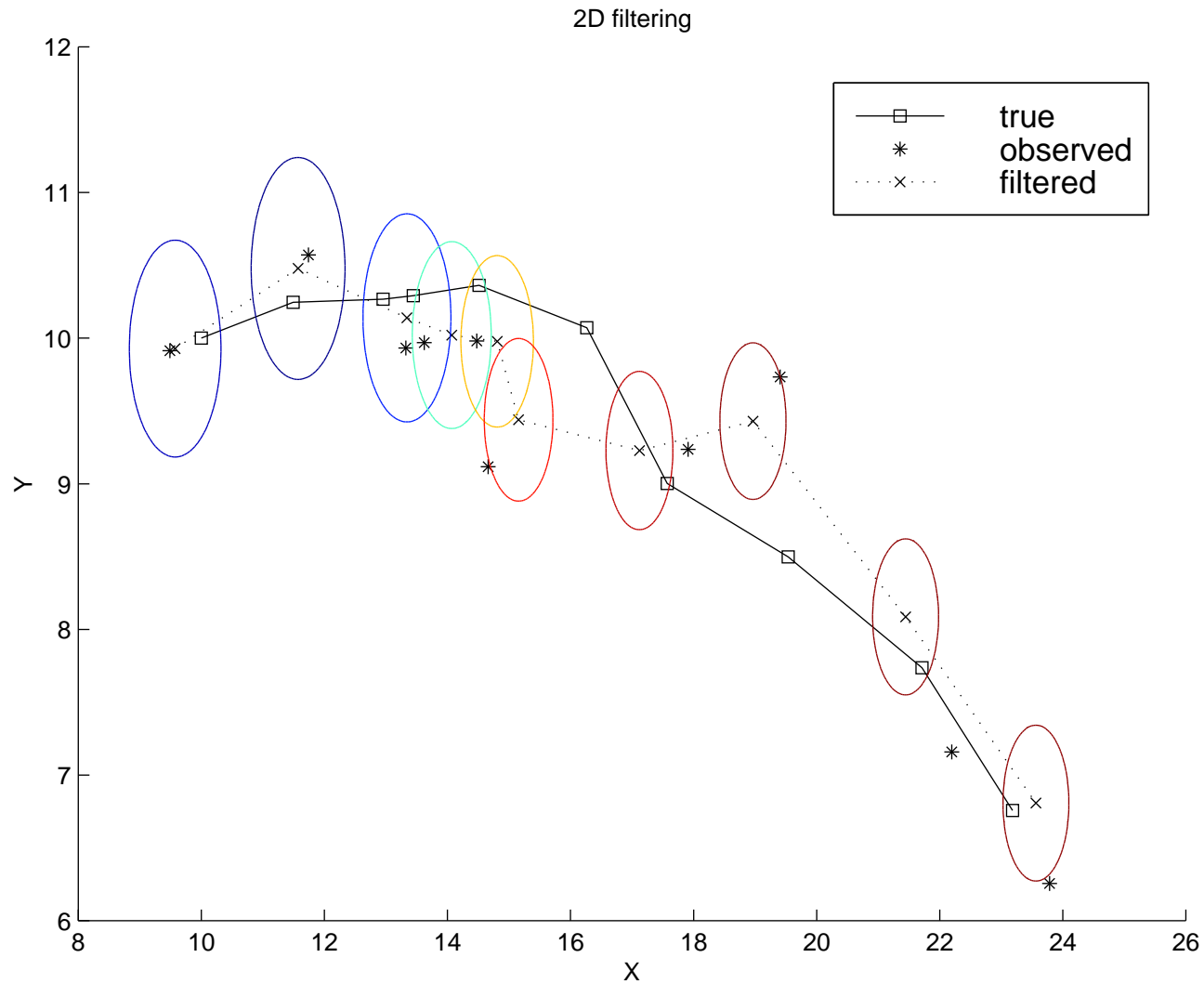
Filter computes the following update:

$$\boldsymbol{\mu}_{t+1} = \mathbf{F}\boldsymbol{\mu}_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t)$$
$$\Sigma_{t+1} = (\mathbf{I} - \mathbf{K}_{t+1}\mathbf{H})(\mathbf{F}\Sigma_t\mathbf{F}^\top + \Sigma_x) ,$$

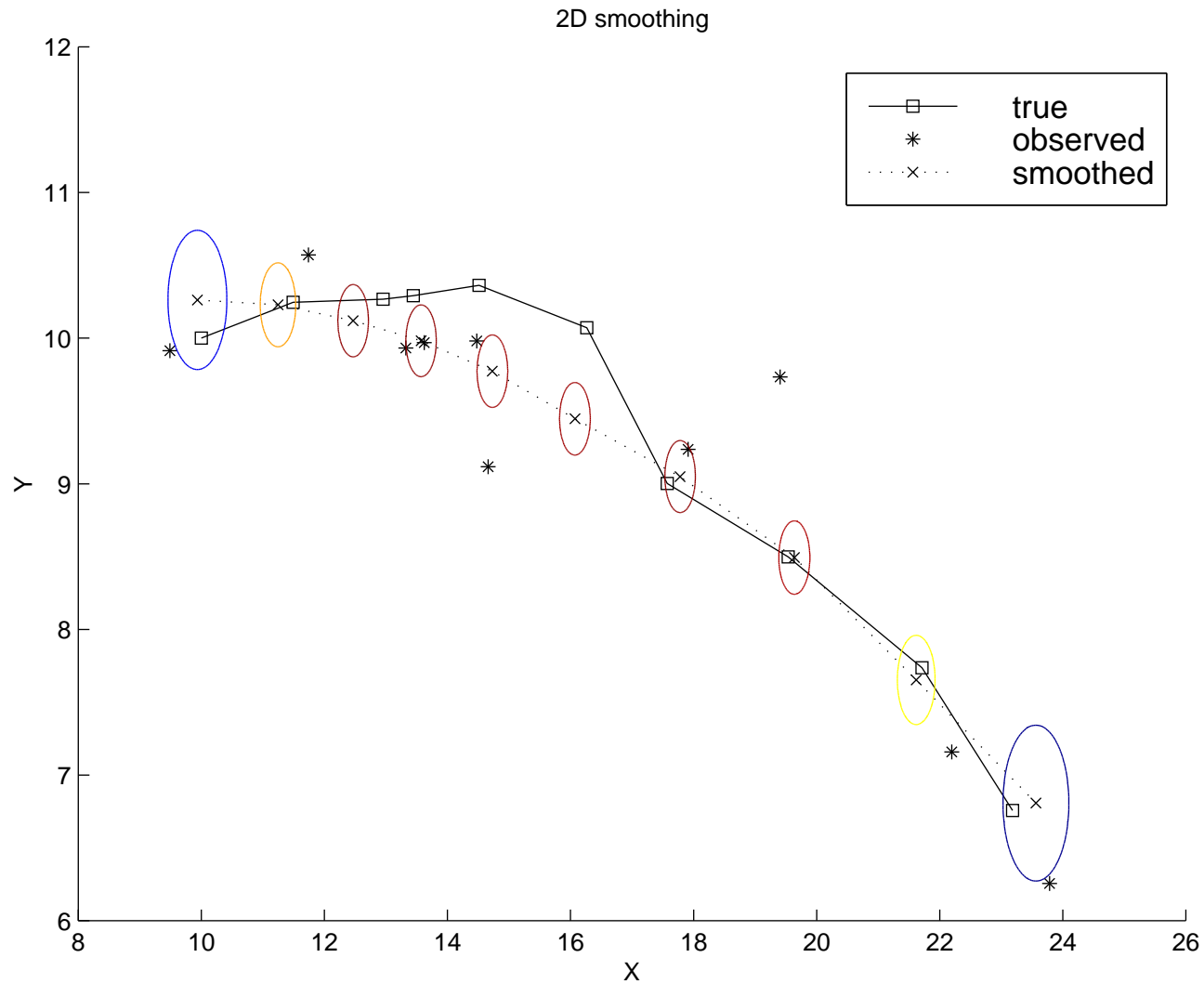
where $\mathbf{K}_{t+1} = (\mathbf{F}\Sigma_t\mathbf{F}^\top + \Sigma_x)\mathbf{H}^\top(\mathbf{H}(\mathbf{F}\Sigma_t\mathbf{F}^\top + \Sigma_x)\mathbf{H}^\top + \Sigma_z)^{-1}$
is the **Kalman gain matrix**

Σ_t and \mathbf{K}_t are independent of observation sequence, so compute offline

2-D tracking example: filtering

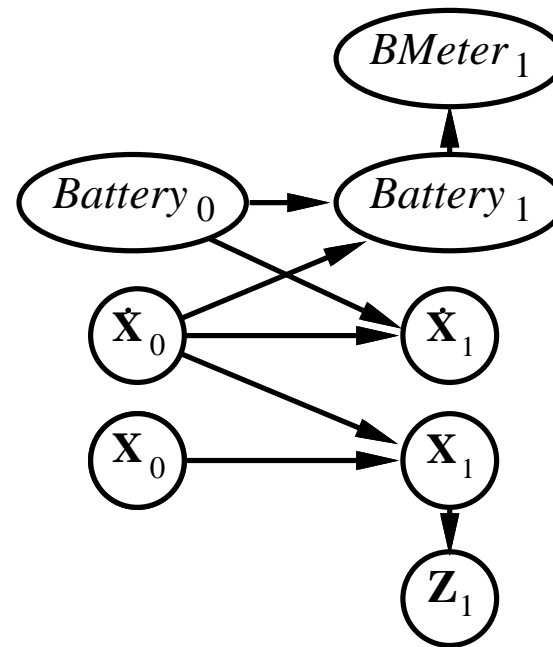
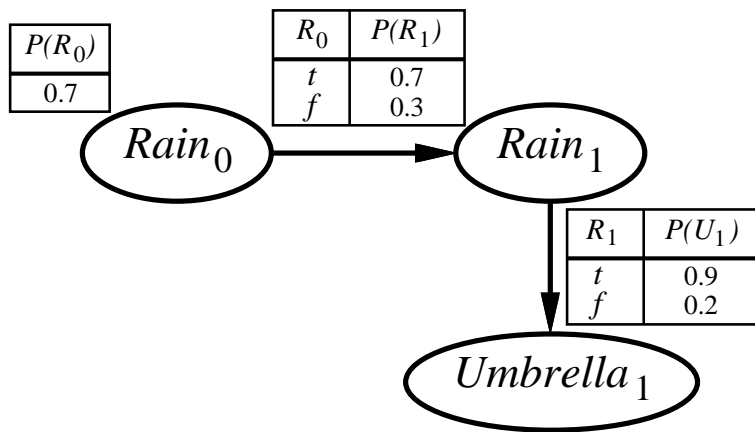


2-D tracking example: smoothing



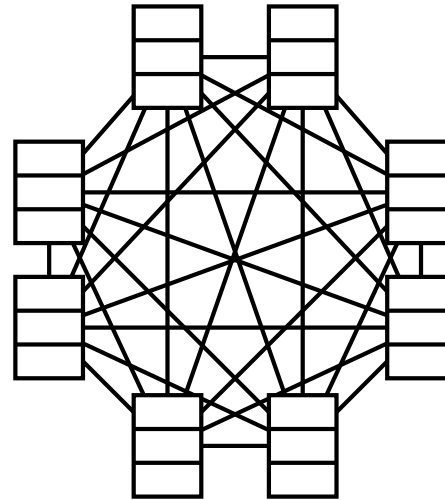
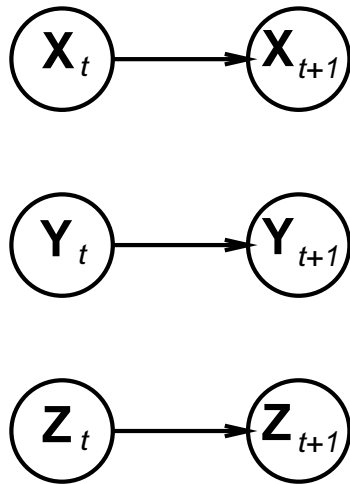
Dynamic Bayesian networks

$\mathbf{X}_t, \mathbf{E}_t$ contain arbitrarily many variables in a replicated Bayes net



DBNs vs. HMMs

Every HMM is a single-variable DBN; every discrete DBN is an HMM



Sparse dependencies \Rightarrow exponentially fewer parameters;

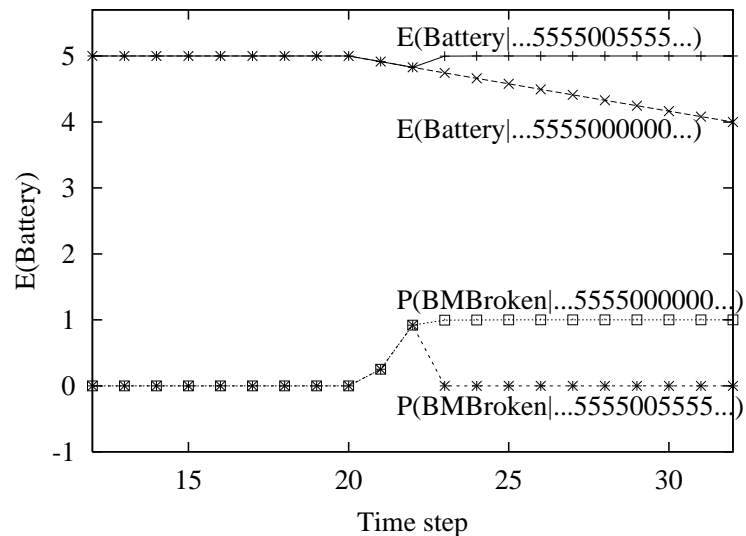
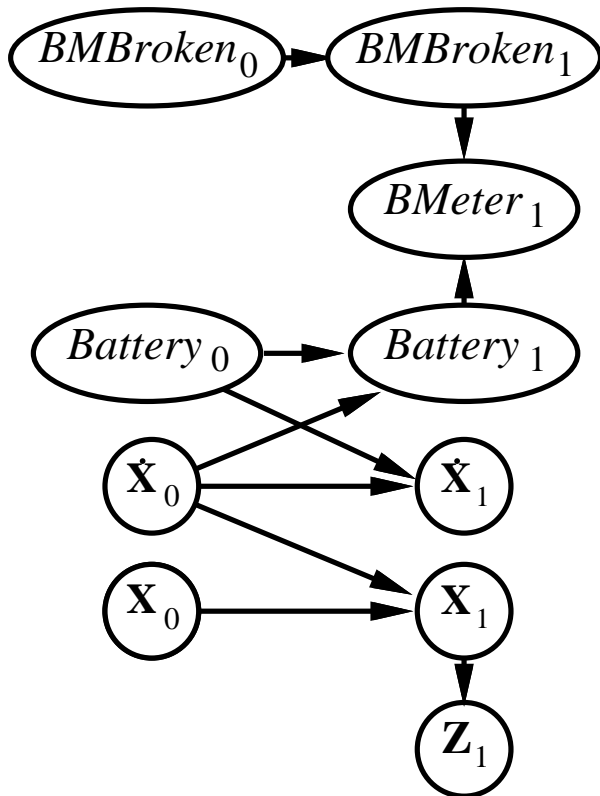
e.g., 20 state variables, three parents each

DBN has $20 \times 2^3 = 160$ parameters, HMM has $2^{20} \times 2^{20} \approx 10^{12}$

DBNs vs Kalman filters

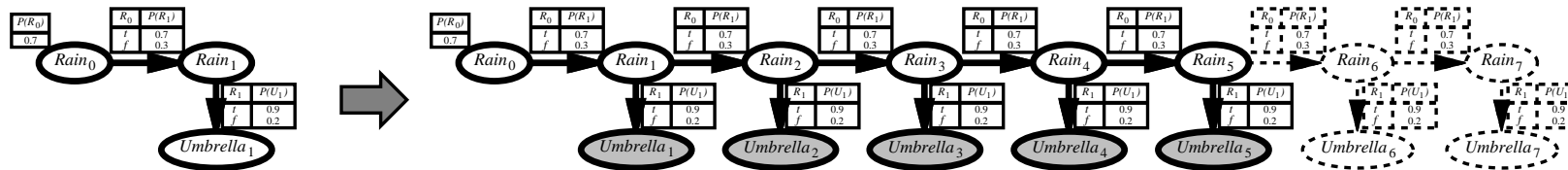
Every Kalman filter model is a DBN, but few DBNs are KFs;
real world requires non-Gaussian posteriors

E.g., where are my keys? What's the battery charge?



Exact inference in DBNs

Naive method: **unroll** the network and run any exact algorithm



Problem: inference cost for each update grows with t

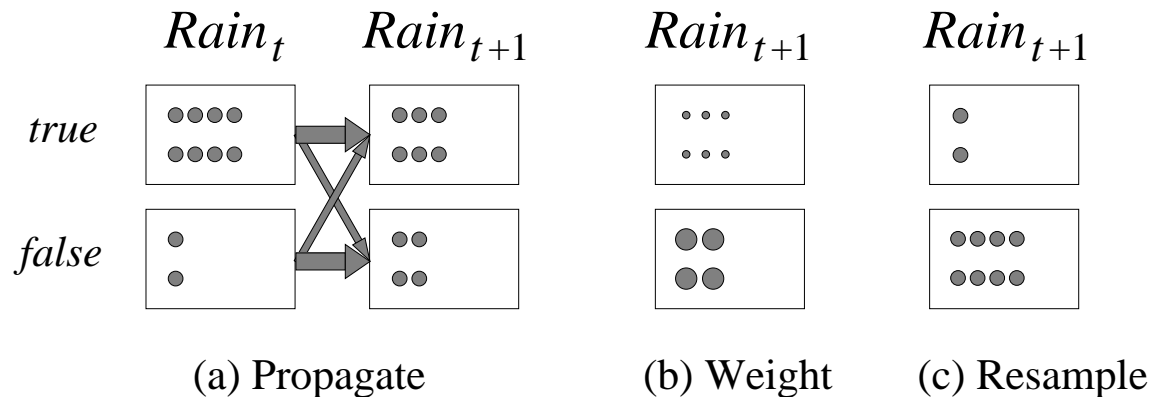
Rollup filtering: add slice $t + 1$, “sum out” slice t using variable elimination

Largest factor is $O(K^{D+L})$, update cost $O(K^{D+L+1})$
 (cf. HMM update cost $O(K^{2D})$)

Particle filtering

Basic idea: posterior at t represented by population of “particles”;
resample given evidence to track high-likelihood regions of the state-space

Replicate particles proportional to likelihood for e_t



Widely used for tracking nonlinear systems, esp. in vision

Also used for simultaneous localization and mapping in mobile robots

10^5 -dimensional state space

Particle filtering contd.

Assume consistent at time t : $N(\mathbf{x}_t|\mathbf{e}_{1:t})/N = P(\mathbf{x}_t|\mathbf{e}_{1:t})$

Propagate forward: populations of \mathbf{x}_{t+1} are

$$N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1}|\mathbf{x}_t)N(\mathbf{x}_t|\mathbf{e}_{1:t})$$

Weight samples by their likelihood for \mathbf{e}_{t+1} :

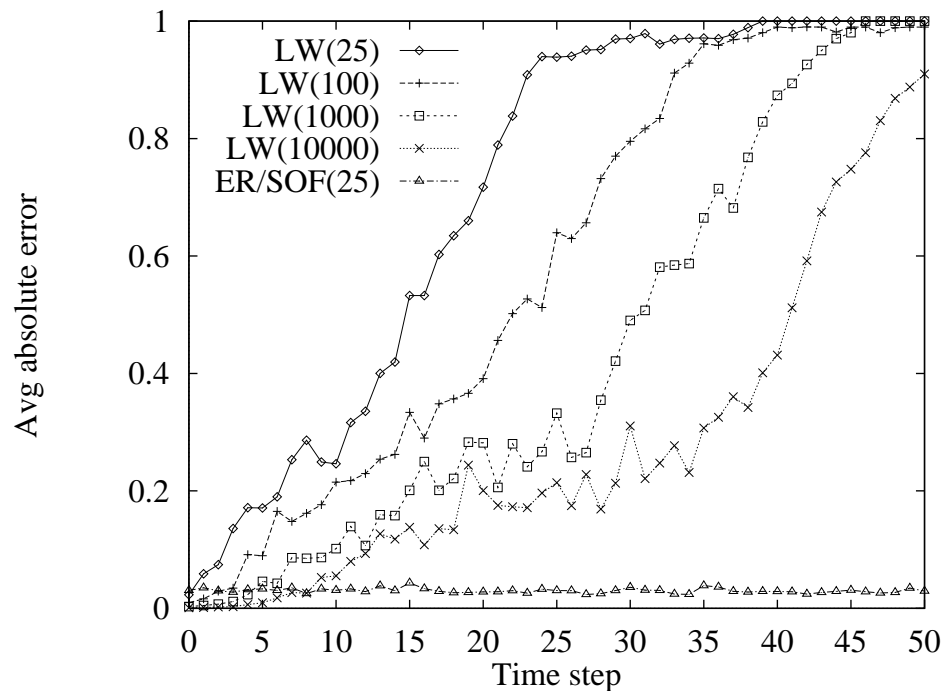
$$W(\mathbf{x}_{t+1}|\mathbf{e}_{1:t+1}) = P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t})$$

Resample to obtain populations proportional to W :

$$\begin{aligned} N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t+1})/N &= \alpha W(\mathbf{x}_{t+1}|\mathbf{e}_{1:t+1}) = \alpha P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1})N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t}) \\ &= \alpha P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1}|\mathbf{x}_t)N(\mathbf{x}_t|\mathbf{e}_{1:t}) \\ &= \alpha' P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1}) \sum_{\mathbf{x}_t} P(\mathbf{x}_{t+1}|\mathbf{x}_t)P(\mathbf{x}_t|\mathbf{e}_{1:t}) \\ &= P(\mathbf{x}_{t+1}|\mathbf{e}_{1:t+1}) \end{aligned}$$

Particle filtering performance

Approximation error of particle filtering remains bounded over time, if transition and sensor probabilities bounded away from 0 and 1



Summary

General stationary Markovian model composed of

- transition model $\mathbf{P}(\mathbf{X}_t | \mathbf{X}_{t-1})$
- sensor model $\mathbf{P}(\mathbf{E}_t | \mathbf{X}_t)$

Tasks are filtering, prediction, smoothing, most likely sequence;
all done recursively with constant cost per time step*

Hidden Markov models have a single discrete state variable

- EM training by forward–backward algorithm
- core model for speech recognition

Kalman filters allow n state variables, linear Gaussian, $O(n^3)$ update

Dynamic Bayes nets subsume HMMs, Kalman filters; exact update intractable

Particle filtering is a good approximate filtering algorithm for DBNs