Temporal probability models

Chapter 15

Outline

- ♢ Time and uncertainty
- ♢ Inference: filtering, prediction, smoothing
- ♢ Hidden Markov models
- ♢ Kalman filters (a brief mention)
- ♢ Dynamic Bayesian networks
- ♢ Particle filtering
- ♢ Speech recognition

Time and uncertainty

The world changes; we need to track and predict it

Diabetes management vs vehicle diagnosis

Basic idea: copy state and evidence variables for each time step

$\mathbf{X}_t$ = 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., $MeasuredBloodSugar_t$, $PulseRate_t$, $FoodEaten_t$

This assumes **discrete time**; step size depends on problem

Notation: $\mathbf{X}_{a:b} = \mathbf{X}_a, \mathbf{X}_{a+1}, \ldots, \mathbf{X}_{b-1}, \mathbf{X}_b$

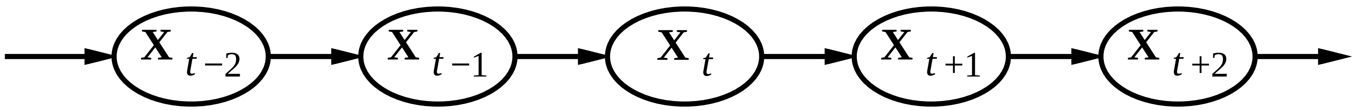
Markov processes (Markov chains)

Construct a Bayes net from these variables: parents?

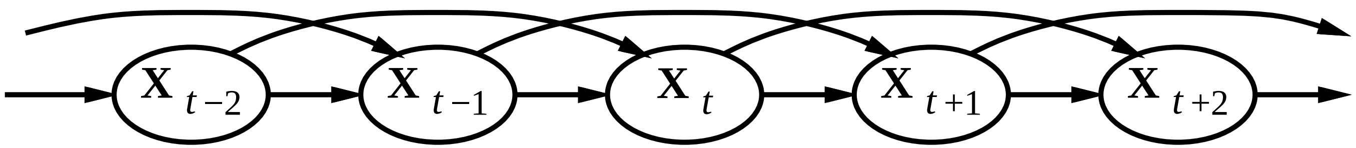
Markov assumption: $\mathbf{X}_t$ depends on **bounded** subset of $\mathbf{X}_{0:t-1}$

First-order Markov process: $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$
Second-order Markov process: $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-2}, \mathbf{X}_{t-1})$

First−order



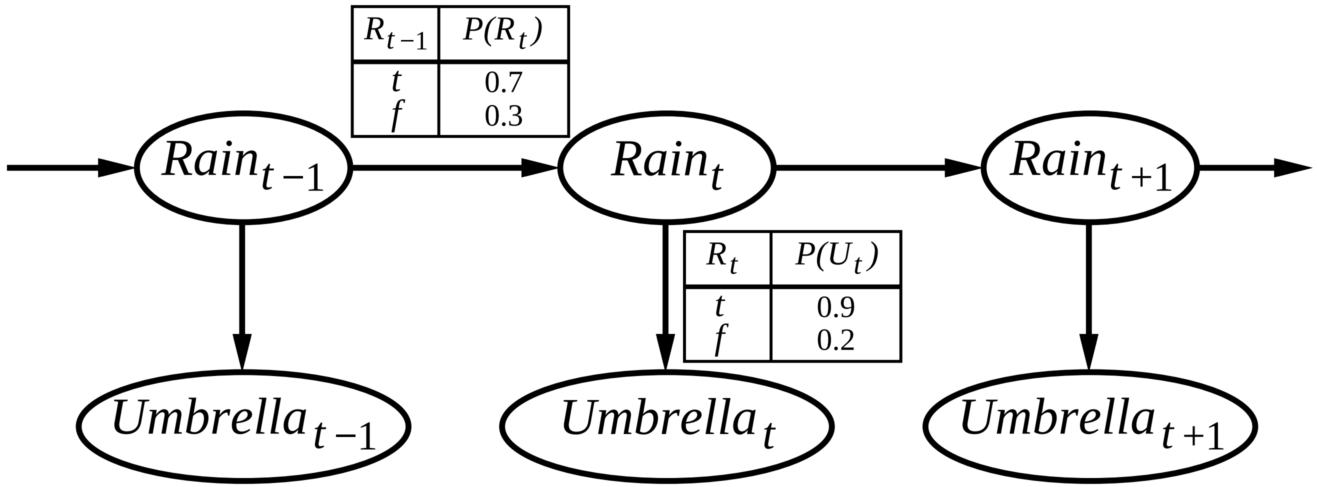
Second−order



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

Stationary process: transition model $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$ and sensor model $\mathbf{P}(\mathbf{E}_t|\mathbf{X}_t)$ fixed for all t

Example



Full joint distribution over all variables:

$$P(X_o, X_1, ..., X_t, E_1, ..., E_t) = P(X_0) \prod_{i=1}^{t} P(X_i | X_{i-1}) P(E_i | X_i)$$

Problems with Markov assumption

First-order Markov assumption is not always 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: $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$
belief state—input to the decision process of a rational agent

Prediction: $\mathbf{P}(\mathbf{X}_{t+k}|\mathbf{e}_{1:t})$ for $k > 0$
evaluation of possible action sequences;
like filtering without the evidence

Smoothing: $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ 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:

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

$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}, \mathbf{e}_{t+1})$ (dividing up evidence)
$= \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}, \mathbf{e}_{1:t})\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$ (using Bayes' rule)
$= \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$ (Markov property of evidence)

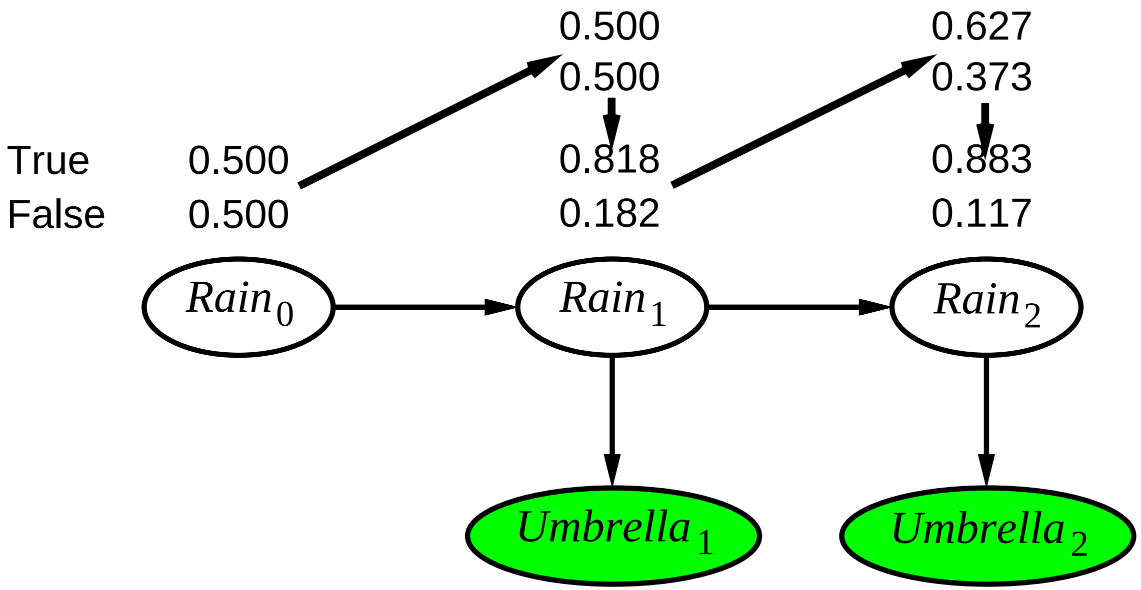
$$\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})\Sigma_{\mathbf{x}_t}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, \mathbf{e}_{1:t})P(\mathbf{x}_t|\mathbf{e}_{1:t})$$
$$= \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})\Sigma_{\mathbf{x}_t}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)P(\mathbf{x}_t|\mathbf{e}_{1: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

Within summation, 1st term: transition model; 2nd: curr. state distribution.

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

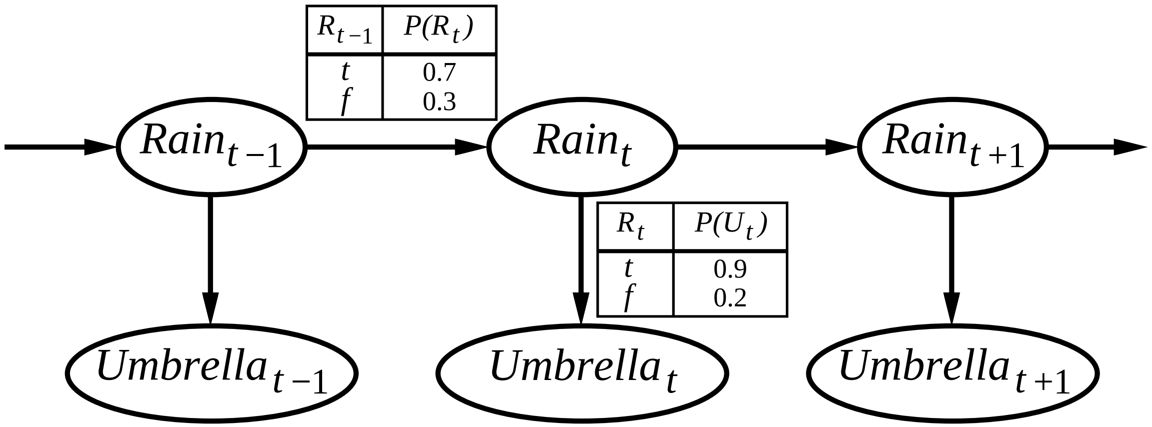
Time and space **constant** (independent of t)

Filtering example



Initially, prior belief of rain on day 0 is $P(R_0) =< 0.5, 0.5 >$.

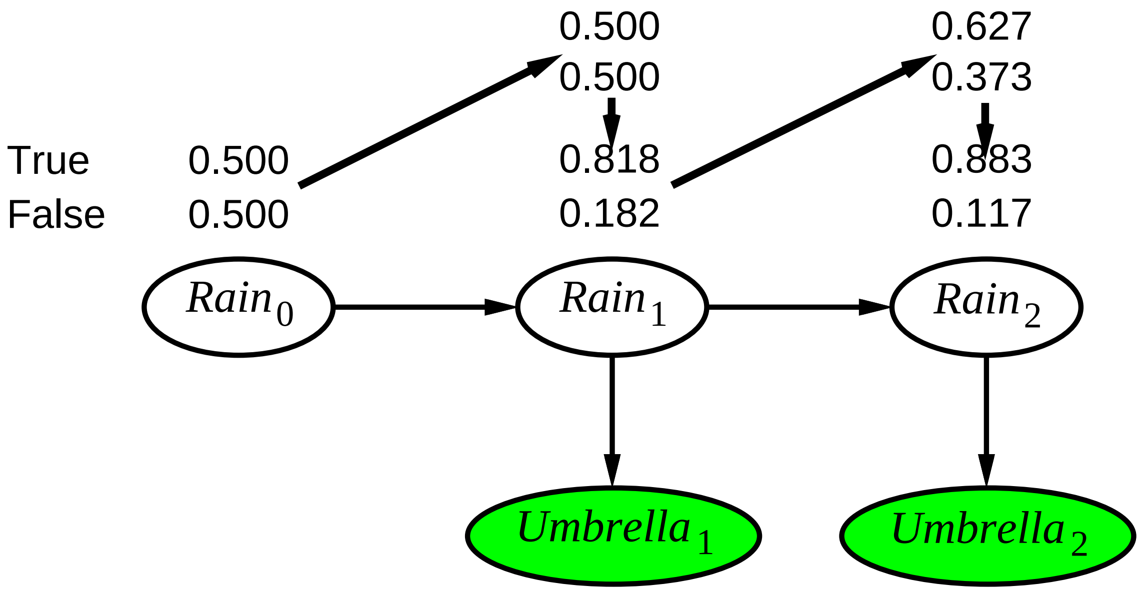
Recall CPTs for Example



Full joint distribution over all variables:

$$P(X_o, X_1, ..., X_t, E_1, ..., E_t) = P(X_0) \prod_{i=1}^{t} P(X_i|X_{i-1})P(E_i|X_i)$$

Filtering example

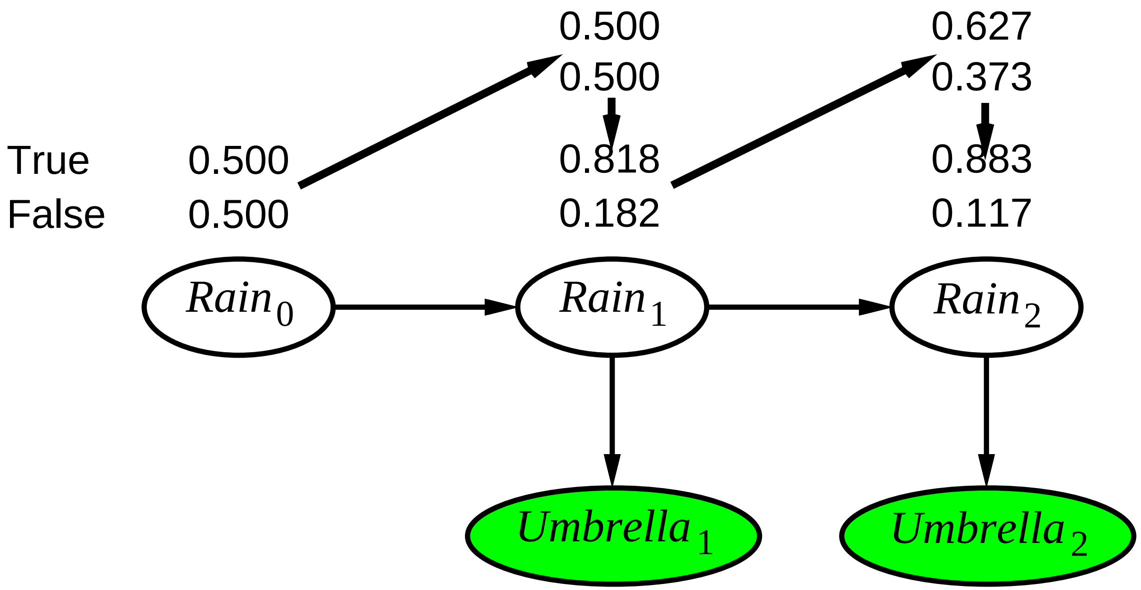


On day 1, umbrella appears, so $U_1 = \text{true}$.

The prediction from $t = 0$ to $t = 1$ is: $P(R_1) = \sum_{r_0} P(R_1|r_0)P(r_0)$

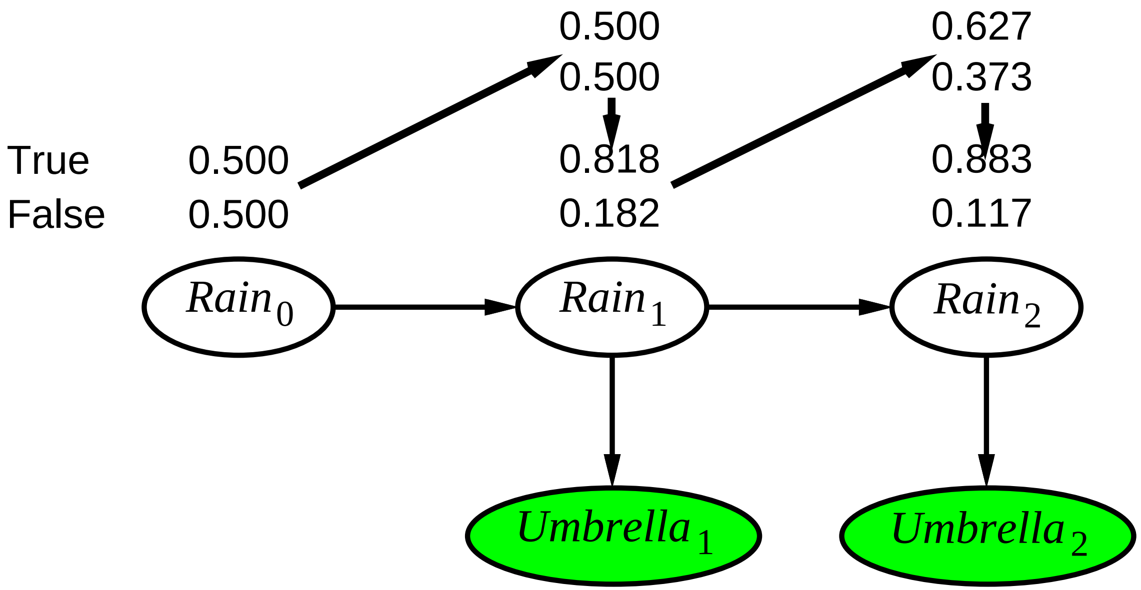
$= < 0.7, 0.3 > \times 0.5 + < 0.3, 0.7 > \times 0.5 = < 0.5, 0.5 >$

Filtering example



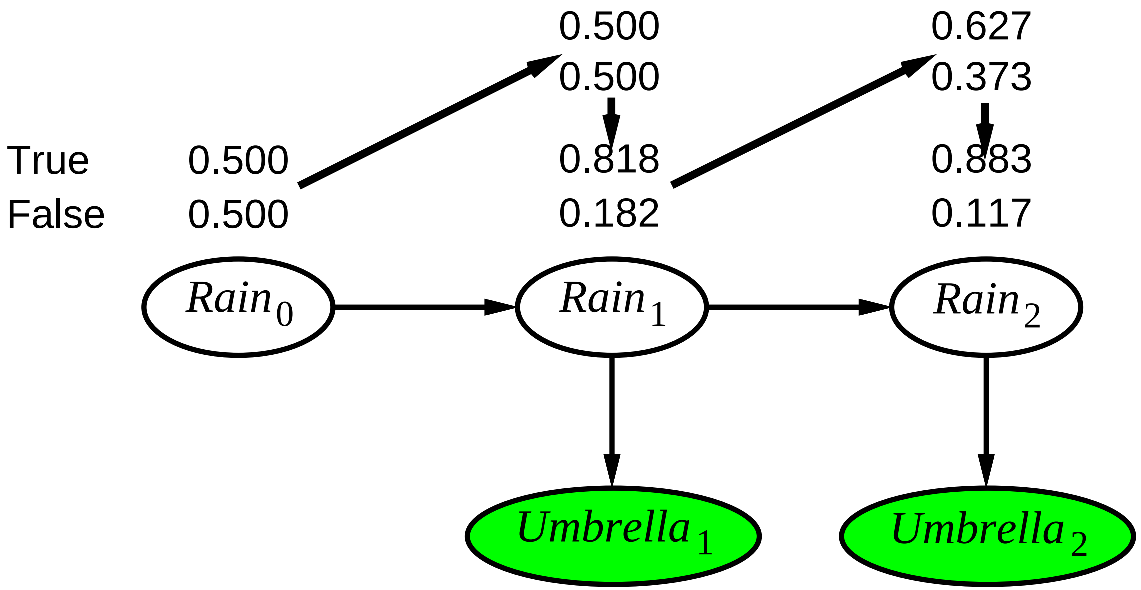
Updating this with evidence for $t = 1$ gives:
$P(R_1|u_1) = \alpha P(u_1|R_1)P(R_1)$
$= \alpha < 0.9, 0.2 >< 0.5, 0.5 >$
$= \alpha < 0.45, 0.1 >=< 0.818, 0.182 >$

Filtering example



On day 2, umbrella appears, so $U_2 =$ true.
The prediction from $t = 1$ to $t = 2$ is: $P(R_2|U_1) = \sum_{r_1} P(R_2|r_1)P(r_1|u_1)$
$= < 0.7, 0.3 > \times 0.818 + < 0.3, 0.7 > \times 0.182 = < 0.627, 0.373 >$

Filtering example



Updating this with evidence for $t = 2$ gives:

$P(R_2|u_1, u_2) = \alpha P(u_2|R_2)P(R_2|u_1)$

$= \alpha < 0.9, 0.2 >< 0.627, 0.373 >$

$= \alpha < 0.565, 0.075 >=< 0.883, 0.117 >$

Prediction

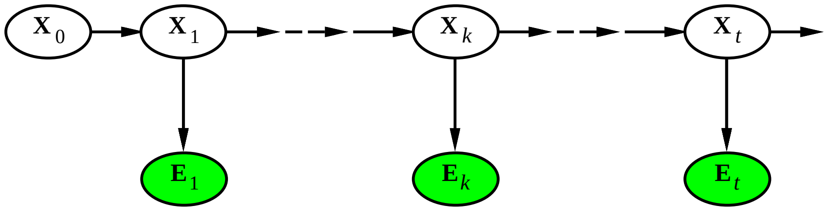
Prediction is same as filtering without addition of new evidence.

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

As $k \rightarrow \infty$, $P(\mathbf{x}_{t+k}|\mathbf{e}_{1:t})$ tends to the **stationary distribution** of the Markov chain

Mixing time depends on how **stochastic** the chain is

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:

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

Smoothing (con't)

Same as previous slide:

Backward message computed by a backwards recursion:

$$\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) = \Sigma_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}|\mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t}|\mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k)$$

Note that first and third terms come directly from model.

Middle term is recursive call: $\mathbf{b}_{k+1:t} = \textsc{Backward}(\mathbf{b}_{k+2:t}, \mathbf{e}_{k+1:t})$
where $\textsc{Backward}$ implements the update from the above equation.
Note that time and space needed for each update are constant, and independent of t.

Forward–backward algorithm: cache forward messages along the way

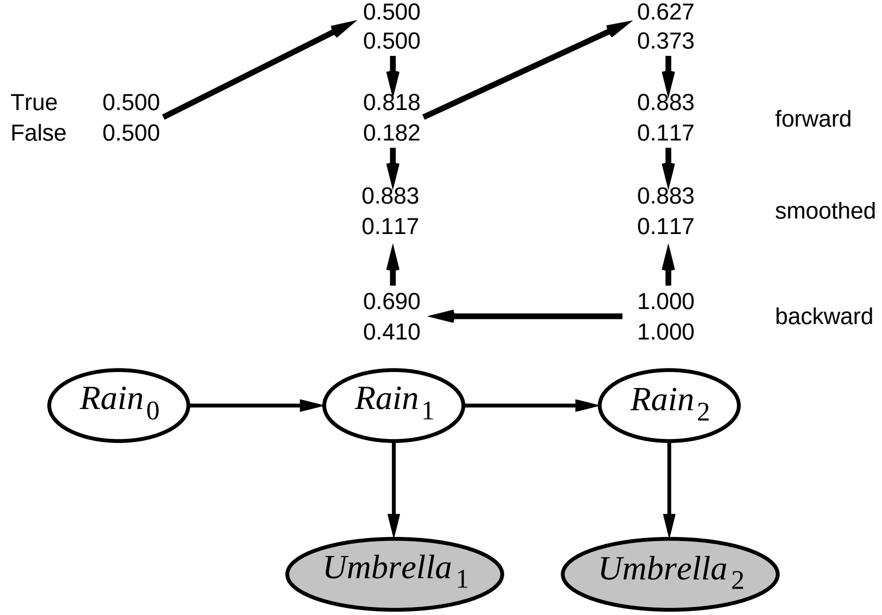
$\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t}) = \alpha \mathbf{f}_{1:k} \mathbf{b}_{k+1:t}$

Time linear in t (polytree inference), space $O(t|\mathbf{f}|)$

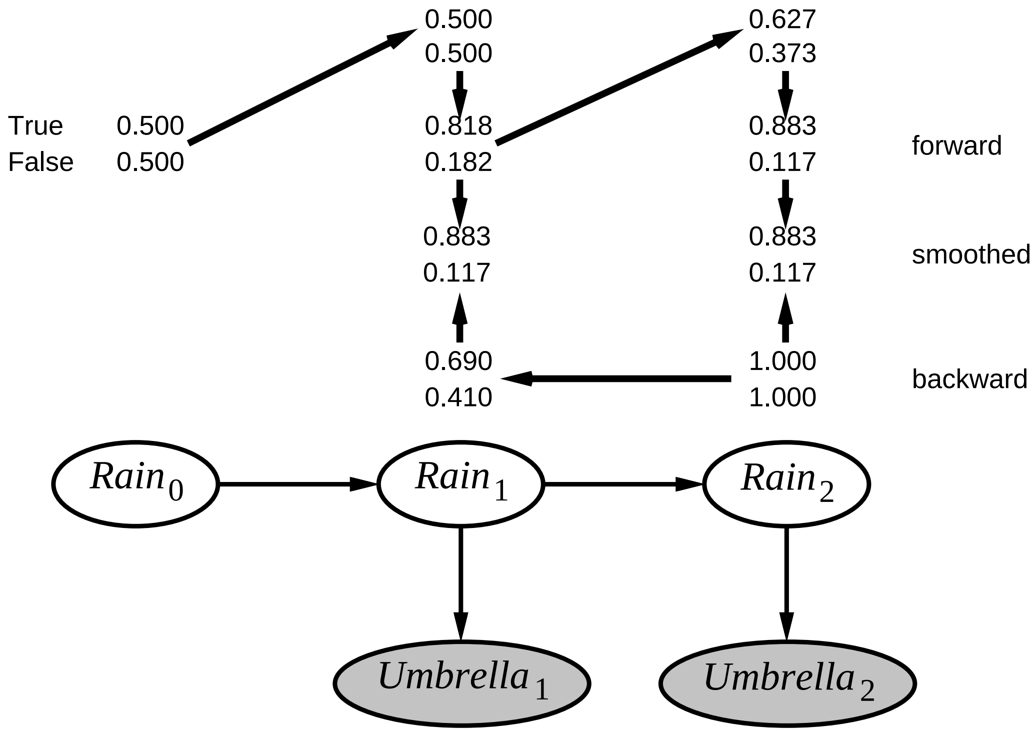
Smoothing example

We want to calculate probability of rain at $t{=}1$, given umbrella observations on days 1 and 2. This is given by: $P(R_1|u_1, u_2) = \alpha P(R_1|u_1)P(u_2|R_1)$

The first term we know (from last week) to be $< .818, .182 >$:



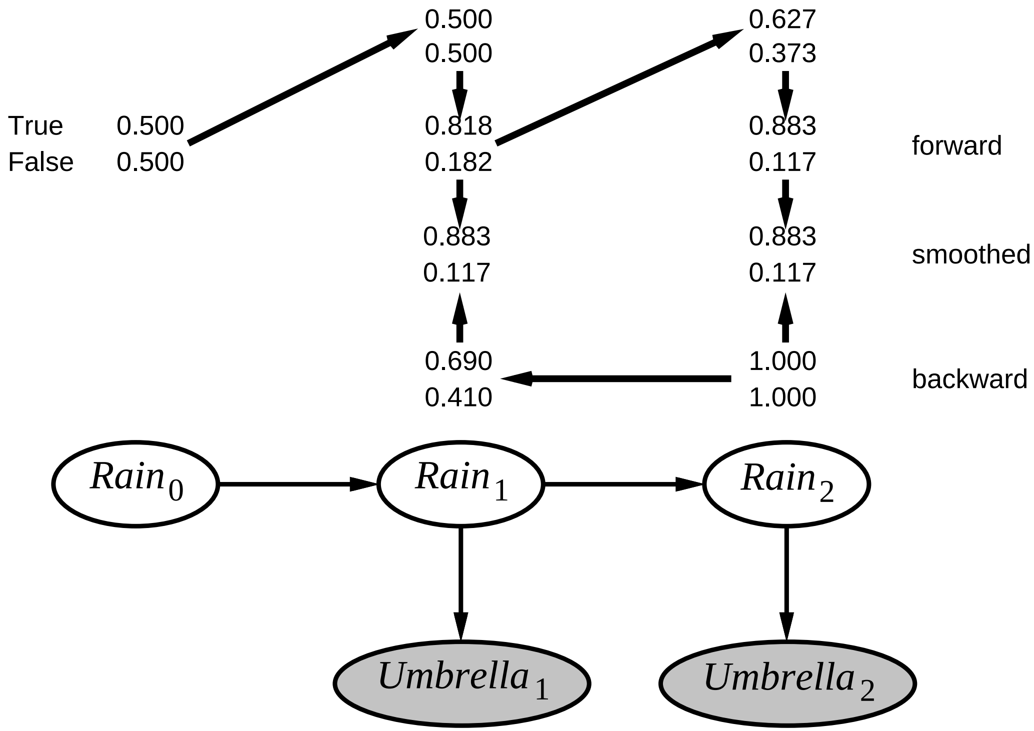
Smoothing example (con't.)



The second term can be computing by applying backward recursion:

$$P(u_2|R_1) = \Sigma_{r_2} P(u_2|r_2)P(|r_2)P(r_2|R_1)$$
$$= (0.9 \times 1 \times <0.7, 0.3>) + (0.2 \times 1 \times <0.3, 0.7>) =< 0.69, 0.41 >$$

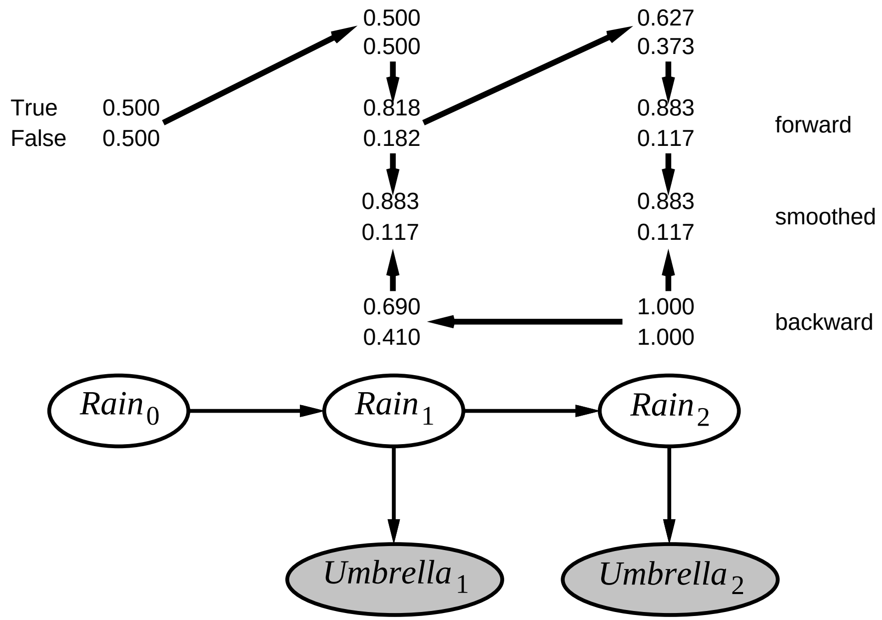
Smoothing example (con't.)



Plugging this into our previous equation, $P(R_1|u_1, u_2) = \alpha P(R_1|u_1)P(u_2|R_1)$, we find that the smoothed estimate for rain on day 1 is:

$$P(R_1|u_1, u_2) = \alpha < 0.818, 0.182 > \times < 0.690, 0.41 > = < 0.883, 0.117 >$$

Smoothing example (con't.)



Note that the smoothed estimate is higher than the filtered estimate (0.818), since the umbrella on day 2 makes it more likely to have rained on day 2, which then makes it more likely to have rained on day 1 (since rain tends to persist).

Time complexity

Both forward and backward recursion take a constant amount of time per-step.

Hence, time complexity of smoothing with respect to evidence $e_{1:t}$ is $O(t)$ for a particular time step k.

If we smooth a whole sequence, we can use dynamic programming to also achieve $O(t)$ complexity (rather than $O(t^2)$ without dynamic programming). The algorithm that implements this approach is called the Forward-Backward algorithm.

Space complexity $= O(|f|t)$, where $|f|$ is size of representation of forward message. This can be reduced to $O(|f| \log t)$, although it requires increasing the time complexity.

Recall: Inference tasks

Filtering: $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$
belief state—input to the decision process of a rational agent

Prediction: $\mathbf{P}(\mathbf{X}_{t+k}|\mathbf{e}_{1:t})$ for $k > 0$
evaluation of possible action sequences;
like filtering without the evidence

Smoothing: $\mathbf{P}(\mathbf{X}_k|\mathbf{e}_{1:t})$ 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

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

$$\max_{\mathbf{x}_1 \ldots \mathbf{x}_t} \mathbf{P}(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathbf{X}_{t+1} | \mathbf{e}_{1:t+1})$$
$$= \alpha \mathbf{P}(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \max_{\mathbf{x}_t} \left(\mathbf{P}(\mathbf{X}_{t+1} | \mathbf{x}_t) \max_{\mathbf{x}_1 \ldots \mathbf{x}_{t-1}} P(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \mathbf{x}_t | \mathbf{e}_{1:t}) \right)$$

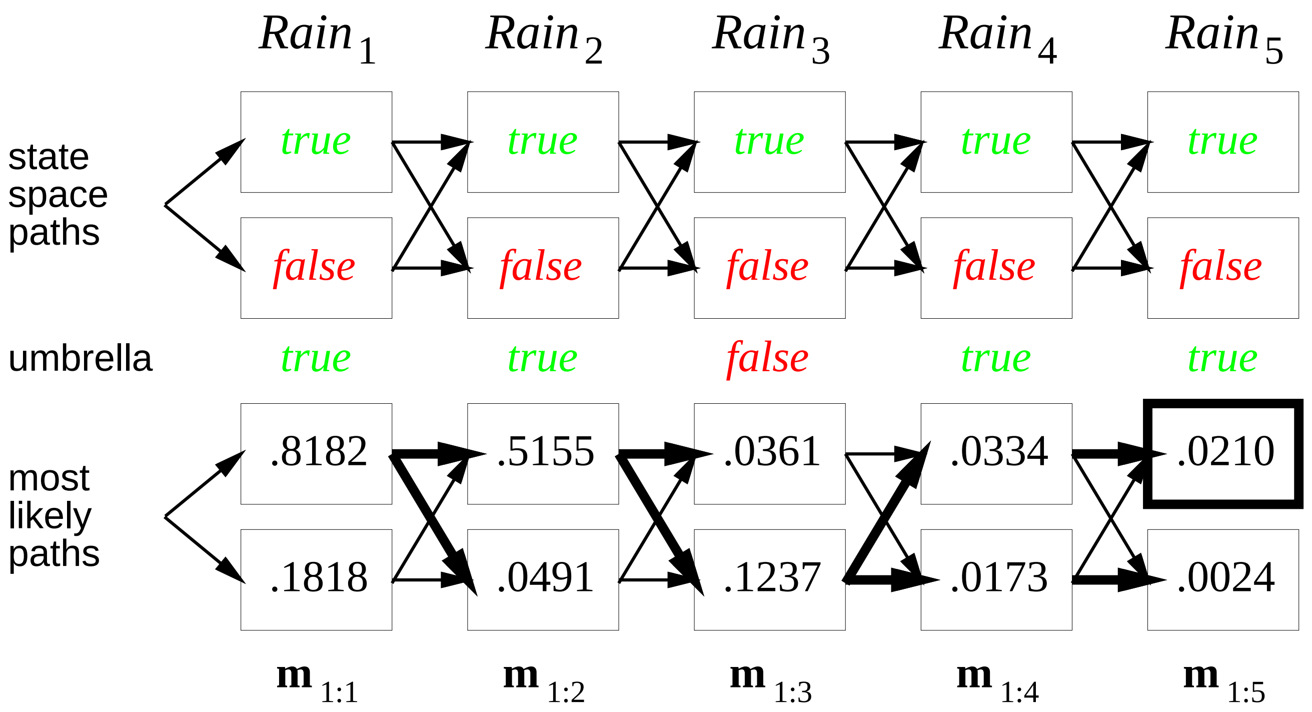
Identical to filtering, except $\mathbf{f}_{1:t}$ replaced by

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

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



Summarizing so far: forward/backward updates

Recall for filtering, we perform 2 calculations.
First, the current state distribution is projected forward from t to $t+1$.
Then, it is updated using new evidence at time $t+1$. This is written as:

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

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

Similarly, for smoothing, we have a backward calculation that works back from time t:

$$\mathbf{b}_{k+2:t} = \mathbf{P}(\mathbf{e}_{k+2:t}|\mathbf{X}_{k+1})$$

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

Hidden Markov models

$\mathbf{X}_t$ is a single, discrete variable (usually $\mathbf{E}_t$ is too)
Domain of X_t is $\{1, \ldots, S\}$ (representing states)

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

Sensor matrix $\mathbf{O}_t$ for each time step, diagonal elements $P(e_t | X_t = i)$
e.g., with $U_1 = true$, $\mathbf{O}_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}$

Forward and backward messages as column vectors:

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

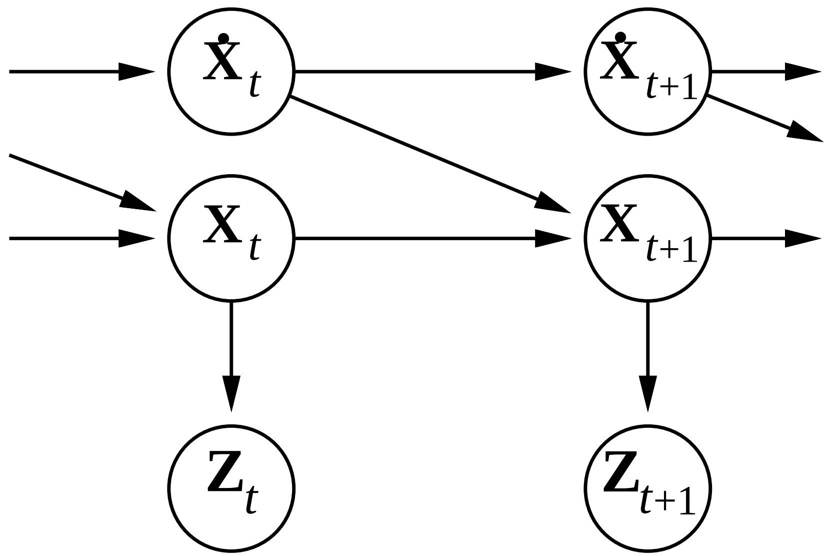
Forward-backward algorithm needs time $O(S^2 t)$ and space $O(St)$

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 (i.e., next state is linear function of current state, plus Gaussian noise), 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 also Gaussian. If $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$ is Gaussian, and the sensor model $\mathbf{P}(e_{t+1}|X_{t+1})$ is linear 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 also Gaussian.

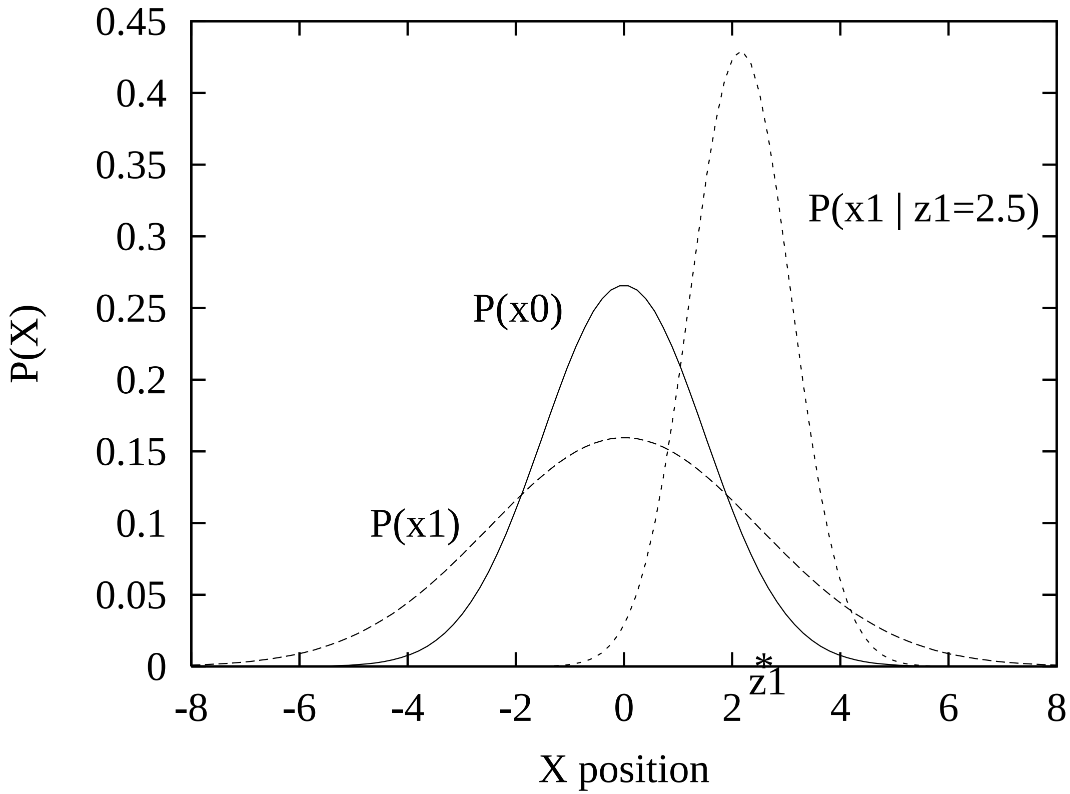
Hence the FORWARD operator for Kalman filtering takes Gaussian $\mathbf{f}_{1:t}$ specified $N(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ and produces a new multivariate Gaussian $\mathbf{f}_{1:t+1}$ specified by $N(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1})$

Important because: General (nonlinear, non-Gaussian) posterior state distribution has representation that grows unboundedly as $t \to \infty$

Simple 1-D example

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

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



General Kalman update

Transition and sensor models:

$$\begin{aligned} P(\mathbf{x}_{t+1}|\mathbf{x}_t) &= N(\mathbf{F}\mathbf{x}_t, \mathbf{\Sigma}_x)(\mathbf{x}_{t+1}) \\ P(\mathbf{z}_t|\mathbf{x}_t) &= N(\mathbf{H}\mathbf{x}_t, \mathbf{\Sigma}_z)(\mathbf{z}_t) \end{aligned}$$

$\mathbf{F}$ is the matrix for the transition; $\mathbf{\Sigma}_x$ the transition noise covariance
$\mathbf{H}$ is the matrix for the sensors; $\mathbf{\Sigma}_z$ the sensor noise covariance

Filter computes the following update:

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \mathbf{F}\boldsymbol{\mu}_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t) \\ \mathbf{\Sigma}_{t+1} &= (\mathbf{I} - \mathbf{K}_{t+1})(\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^\top + \mathbf{\Sigma}_x) \end{aligned}$$

where $\mathbf{K}_{t+1} = (\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^\top + \mathbf{\Sigma}_x)\mathbf{H}^\top(\mathbf{H}(\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^\top + \mathbf{\Sigma}_x)\mathbf{H}^\top + \mathbf{\Sigma}_z)^{-1}$
is the Kalman gain matrix

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

Interpreting General Kalman update

Filter updates (same as last slide):

$$\begin{aligned}\boldsymbol{\mu}_{t+1} &= \mathbf{F}\boldsymbol{\mu}_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t) \\ \boldsymbol{\Sigma}_{t+1} &= (\mathbf{I} - \mathbf{K}_{t+1})(\mathbf{F}\boldsymbol{\Sigma}_t\mathbf{F}^\top + \boldsymbol{\Sigma}_x)\end{aligned}$$

Explanation:

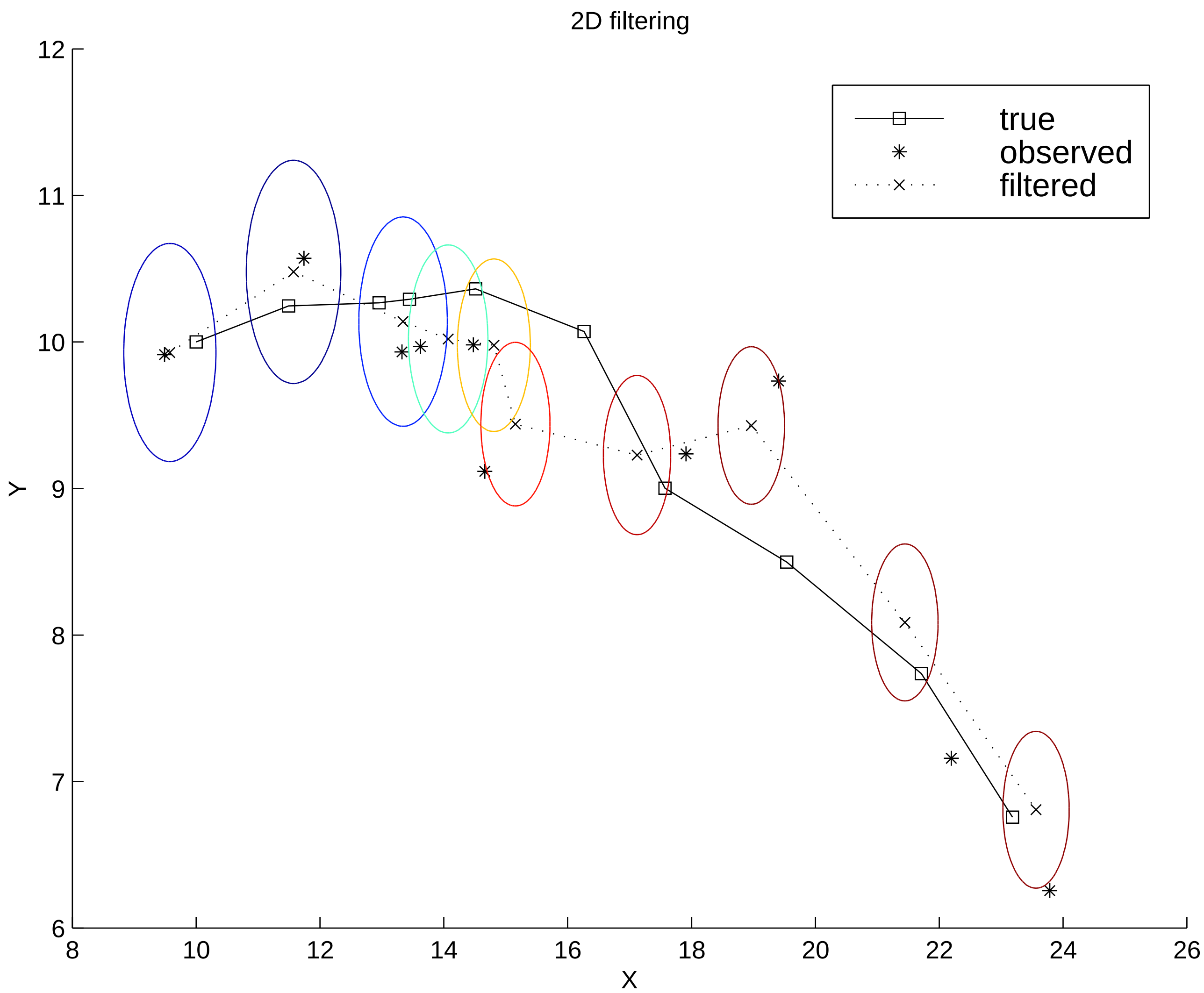
$\mathbf{F}\boldsymbol{\mu}_t$: predicted state at t+1

$\mathbf{H}\mathbf{F}\boldsymbol{\mu}_t$: predicted observation at t+1

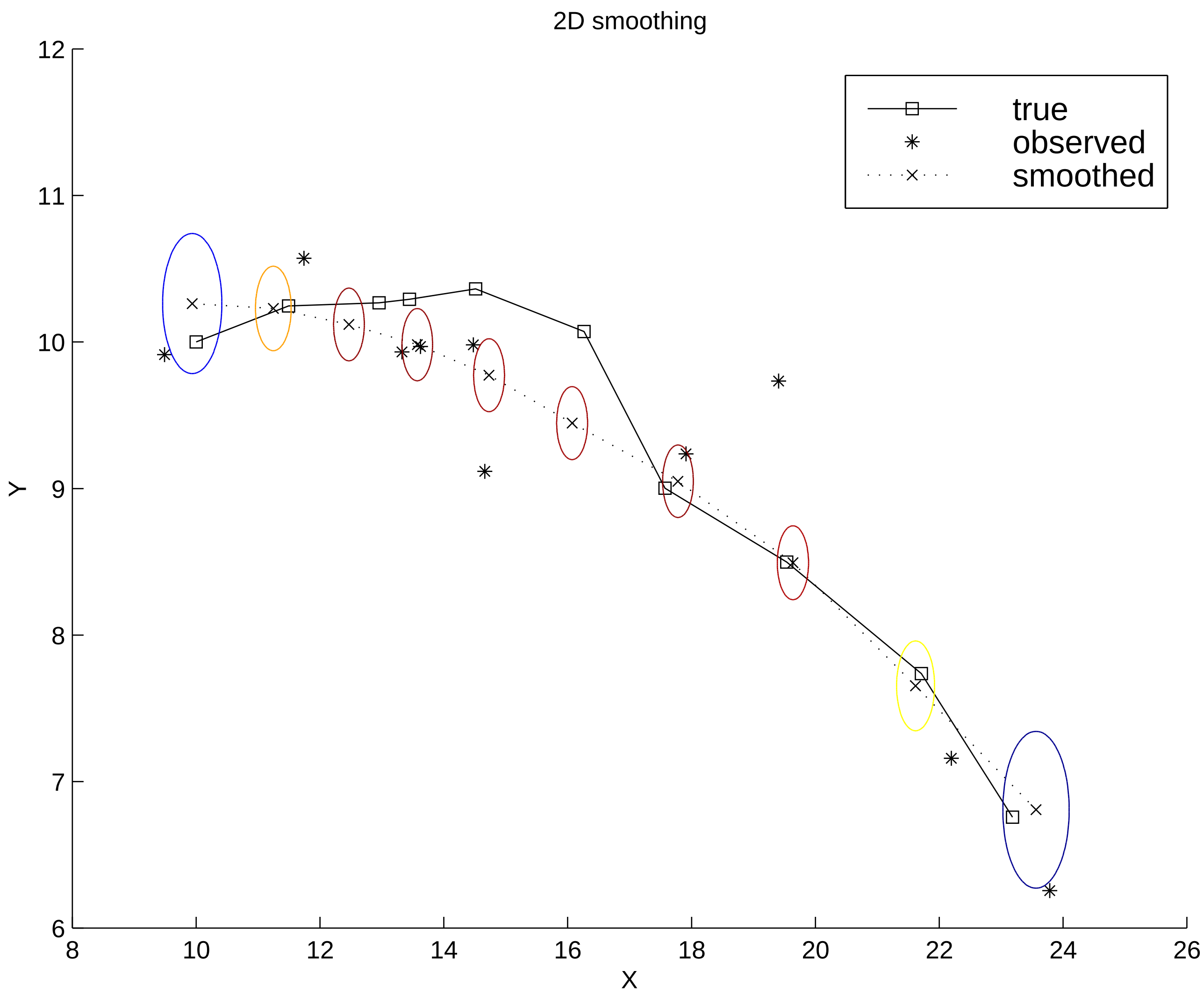
$\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t$: error in predicted observation

$\mathbf{K}_{t+1}$: measure of how seriously to take new observation

2-D tracking example: filtering



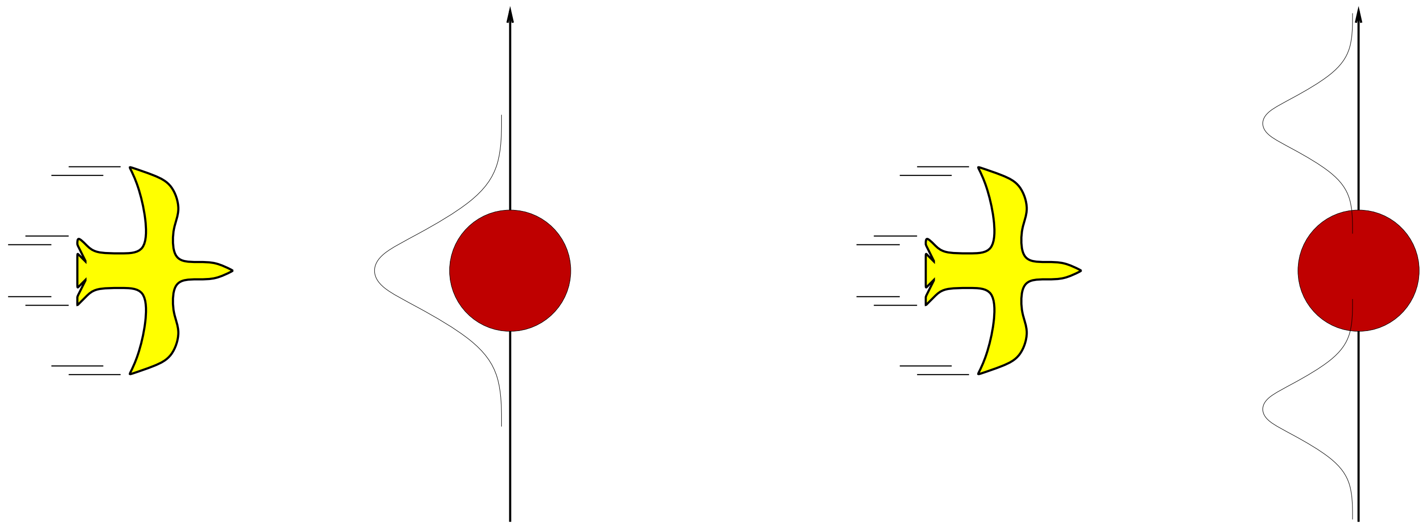
2-D tracking example: smoothing



Where it breaks

Cannot be applied if the transition model is nonlinear

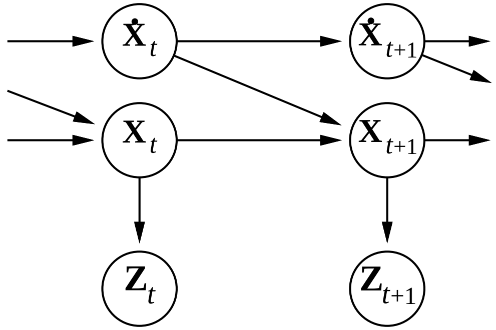
Extended Kalman Filter models transition as **locally linear** around $\mathbf{x}_t = \boldsymbol{\mu}_t$
Fails if systems is locally unsmooth



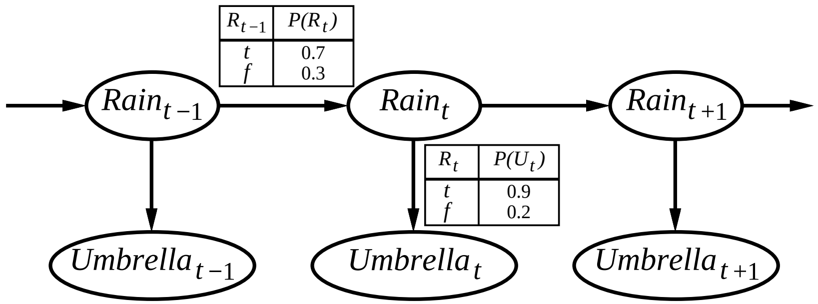
Dynamic Bayesian networks

A dynamic Bayesian network is a Bayesian network that represents a temporal probability model. Examples we've already seen:

Kalman filter NW:

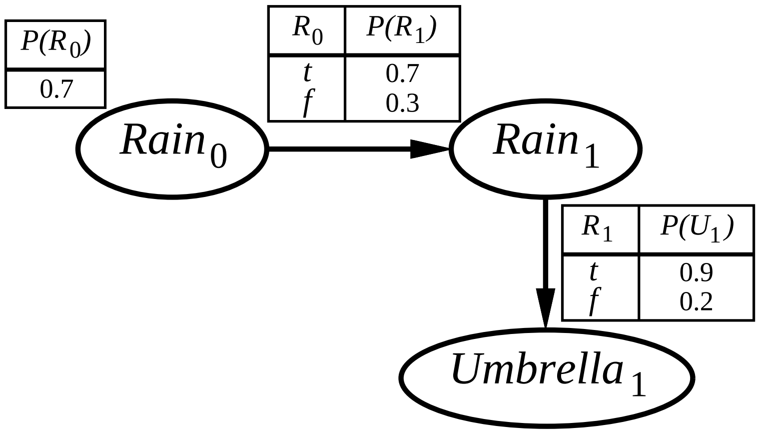


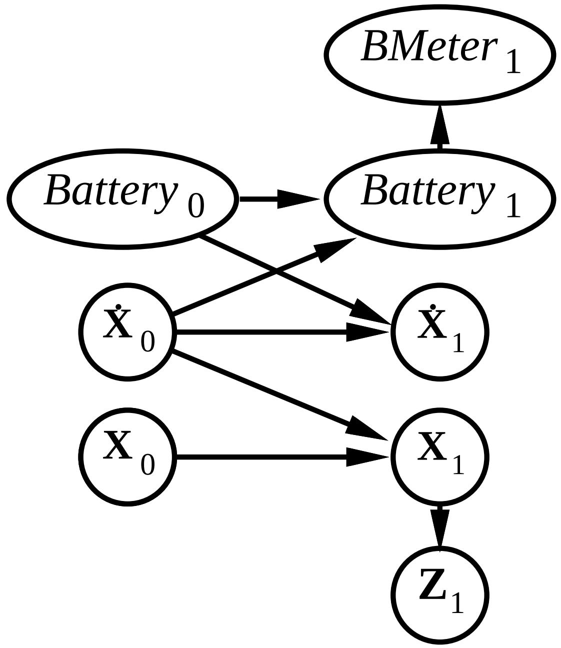
Umbrella NW:



Dynamic Bayesian networks

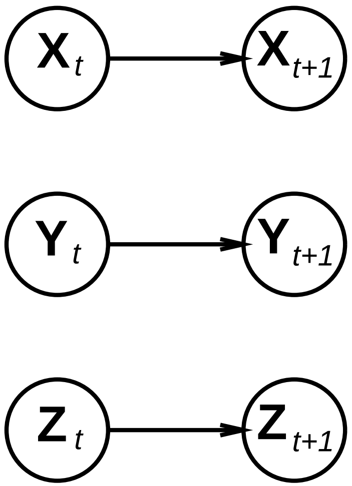
$\mathbf{X}_t$, $\mathbf{E}_t$ contain arbitrarily many variables in a replicated Bayes net

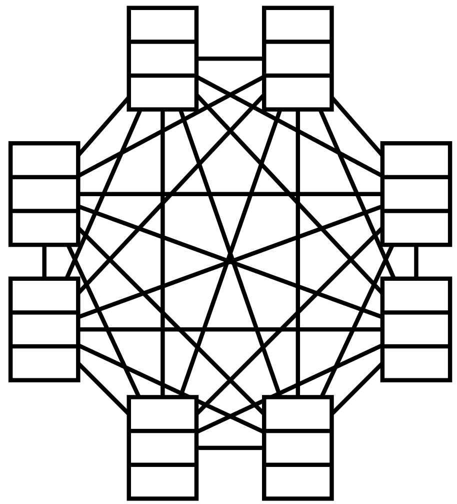




DBNs vs. HMMs

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





What's the difference?

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. HMMs

Implications:

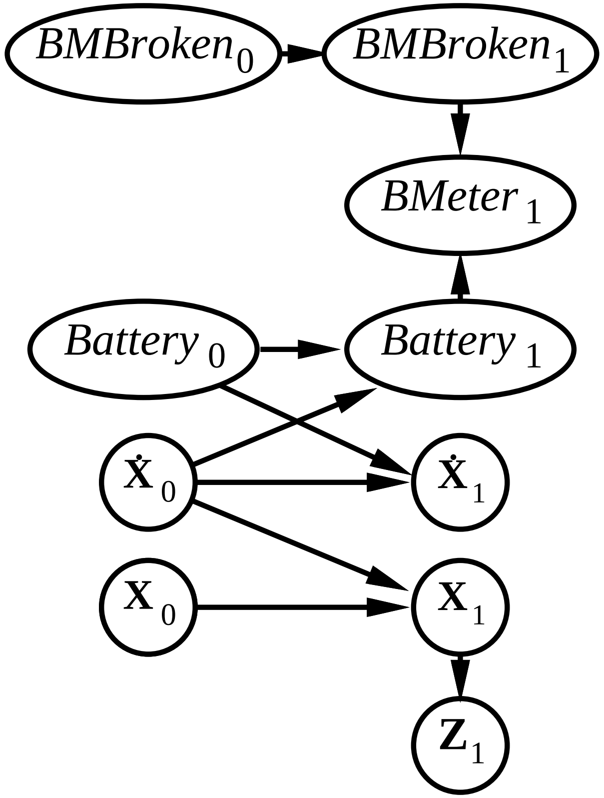
- HMM requires much more space
- HMM inference is much more expensive (due to huge transition matrix)
- Learning large number of parameters makes pure HMM unsuitable for large problems.

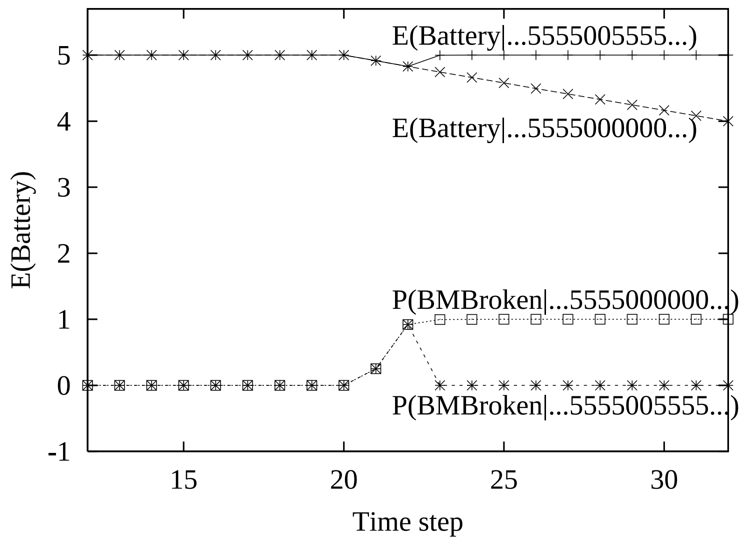
Relationship between DBNs and HMMs is roughly analogous to relationship between ordinary Bayesian networks and full tabulated joint distributions.

DBNs vs Kalman filters

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

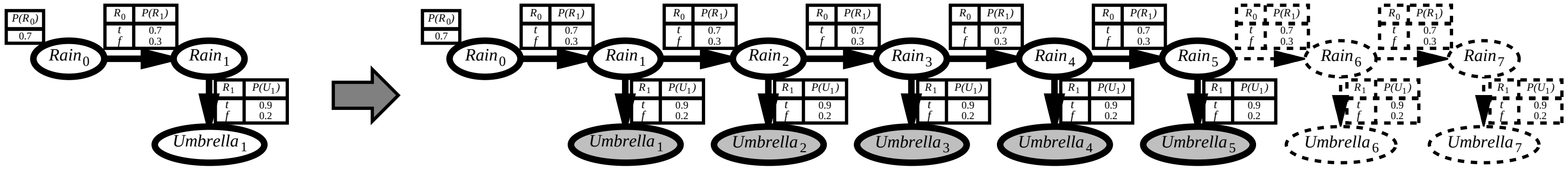
E.g., where are bin Laden and my keys? What's the battery charge?





Exact inference in DBNs

Naive method: unroll the network and run any exact algorithm



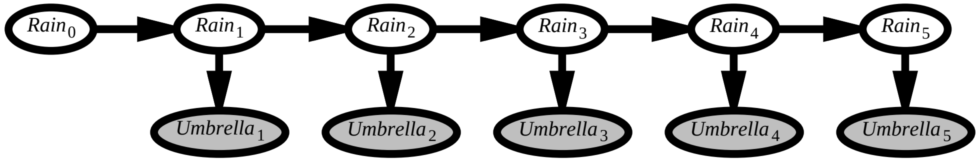
Problem: inference cost for each update grows with t

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

Largest factor is $O(d^{n+1})$, update cost $O(d^{n+2})$
(cf. HMM update cost $O(d^{2n})$)

Implication: Even though we can use DBNs to efficiently represent complex temporal processes with many sparsely connected variables, we cannot reason efficiently and exactly about those processes.

Likelihood weighting for DBNs



Could apply likelihood weighting directly to an unrolled DBN, but this would have problems in terms of increasing time and space requirements per update as observation sequence grows. (Remember, standard algorithm runs each sample in turn, all the way through the network.)

Instead, select N samples, and run all samples together through the network, one slice at a time.

The set of samples serves as approximate representation of the current belief state distribution.

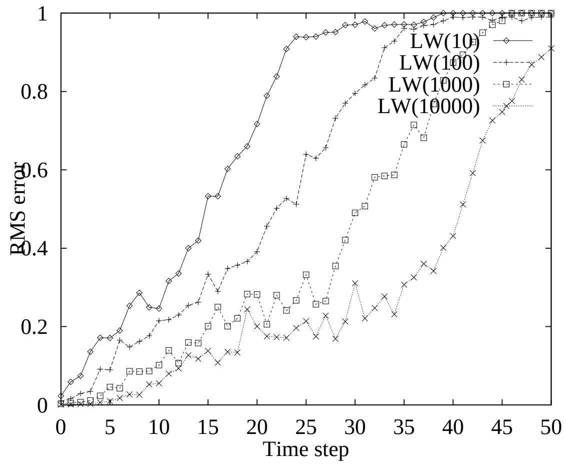
Update is now "constant" time (although dependent on number of samples required to maintain reasonable approximation to the true posterior distribution).

Likelihood weighting for DBNs (con't.)

LW samples pay no attention to the evidence!

⇒ fraction "agreeing" falls exponentially with t

⇒ number of samples required grows exponentially with t

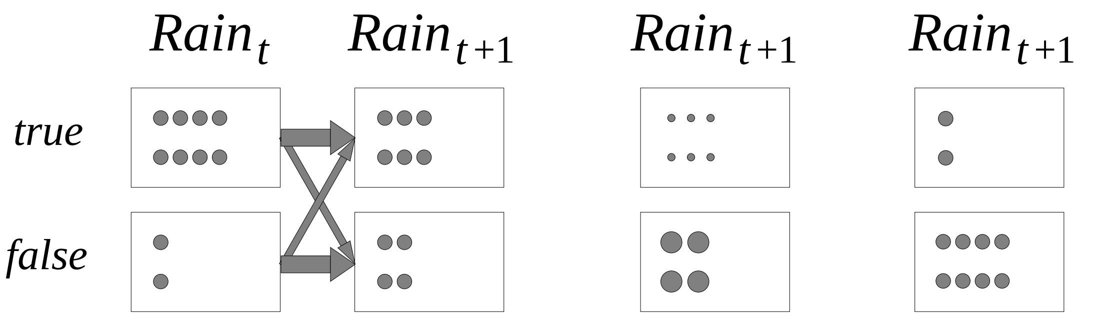


Solution: Focus the set of samples on the high-probability regions of the state space – throw away samples with very low probability, while multiplying those with high probability.

Particle filtering

Basic idea: ensure that the population of samples ("particles") tracks the high-likelihood regions of the state-space

Replicate particles proportional to likelihood for $\mathbf{e}_t$



(a) Propagate (b) Weight (c) Resample

Widely used for tracking nonlinear systems, esp. in vision

Also used for simultaneous localization and mapping in mobile robots.

Particle filtering (cont'd)

Let $N(\mathbf{x}_t|\mathbf{e}_{1:t})$ represent the number of samples occupying state $\mathbf{x}_t$ after observations $\mathbf{e}_{1:t}$. 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

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

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

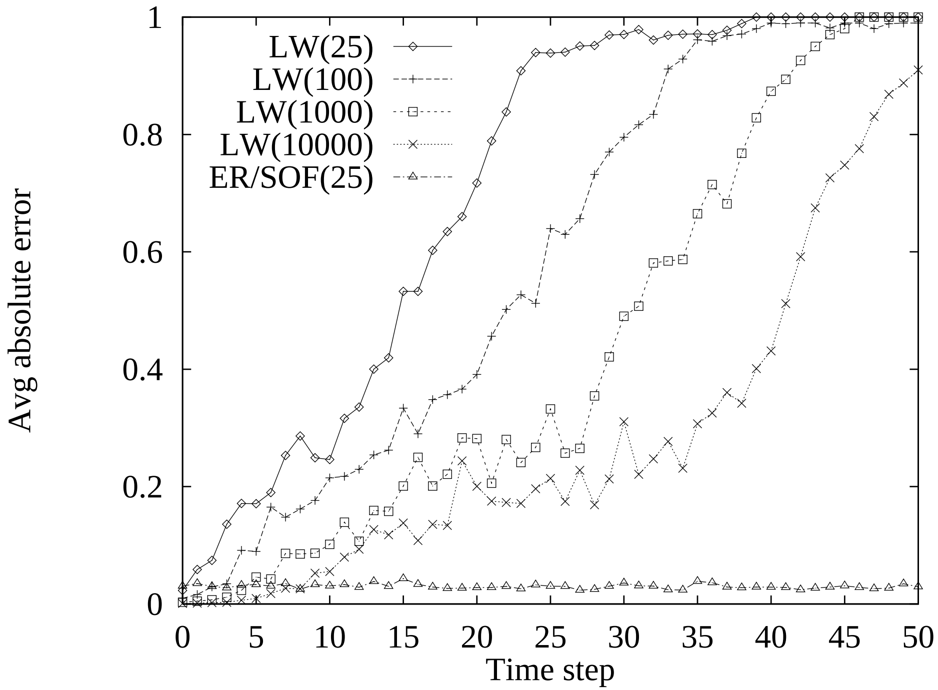
$$W(\mathbf{x}_{t+1}|\mathbf{e}_{1:t+1}) = P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1}) N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t})$$

Resample to obtain populations proportional to W:

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

Particle filtering performance

Approximation error of particle filtering remains bounded over time, at least empirically—theoretical analysis is difficult



Speech recognition – Outline

- ◊ Speech as probabilistic inference
- ◊ Speech sounds
- ◊ Word pronunciation
- ◊ Word sequences

Speech as probabilistic inference

Speech signals are noisy, variable, ambiguous

$Words$ = random variable ranging over all possible sequences of words that might be uttered.

What is the **most likely** word sequence, given the speech signal?
I.e., choose $Words$ to maximize $P(Words|signal)$

Use Bayes' rule:

$$P(Words|signal) = \alpha P(signal|Words)P(Words)$$

I.e., decomposes into acoustic model + language model

$Words$ are the hidden state sequence, $signal$ is the observation sequence

Phones

All human speech is composed from 40-50 phones, determined by the configuration of articulators (lips, teeth, tongue, vocal cords, air flow)

Form an intermediate level of hidden states between words and signal
$\Rightarrow$ acoustic model = pronunciation model + phone model

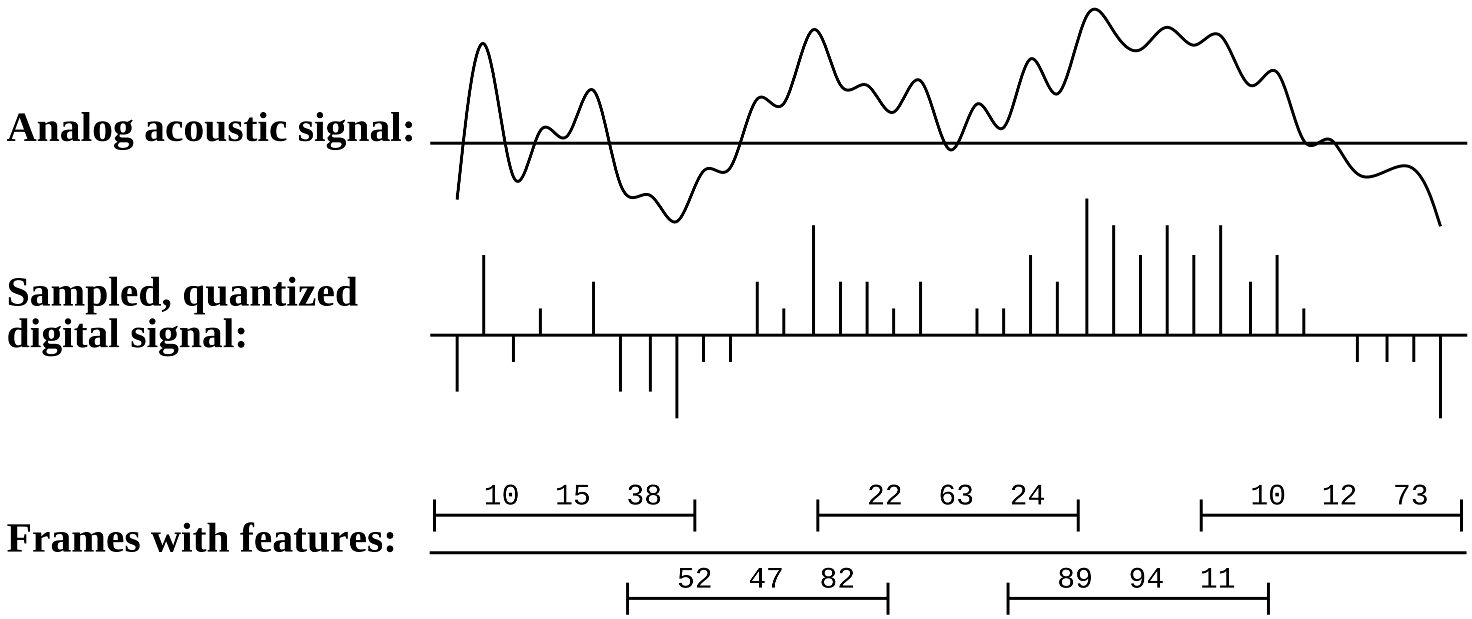
ARPAbet designed for American English

[iy]	b<u>ea</u>t	[b]	<u>b</u>et	[p]	<u>p</u>et
[ih]	b<u>i</u>t	[ch]	<u>Ch</u>et	[r]	<u>r</u>at
[ey]	b<u>e</u>t	[d]	<u>d</u>ebt	[s]	<u>s</u>et
[ao]	b<u>ough</u>t	[hh]	<u>h</u>at	[th]	<u>th</u>ick
[ow]	b<u>oa</u>t	[hv]	<u>h</u>igh	[dh]	<u>th</u>at
[er]	B<u>er</u>t	[l]	<u>l</u>et	[w]	<u>w</u>et
[ix]	ros<u>e</u>s	[ng]	si<u>ng</u>	[en]	butt<u>on</u>
⋮	⋮	⋮	⋮	⋮	⋮

E.g., "ceiling" is [s iy l ih ng] / [s iy l ix ng] / [s iy l en]

Speech sounds

Raw signal is the microphone displacement as a function of time;
processed into overlapping 30ms frames, each described by features



Frame features are typically formants—peaks in the power spectrum

Phone models

Problem: n features with 256 possible values gives us 256^n possible frames. So, we can't represent $P(features|phone)$ as look-up table.

Alternatives: Frame features in $P(features|phone)$ summarized by
– an integer in $[0 \ldots 255]$ (using vector quantization); or
– the parameters of a mixture of Gaussians

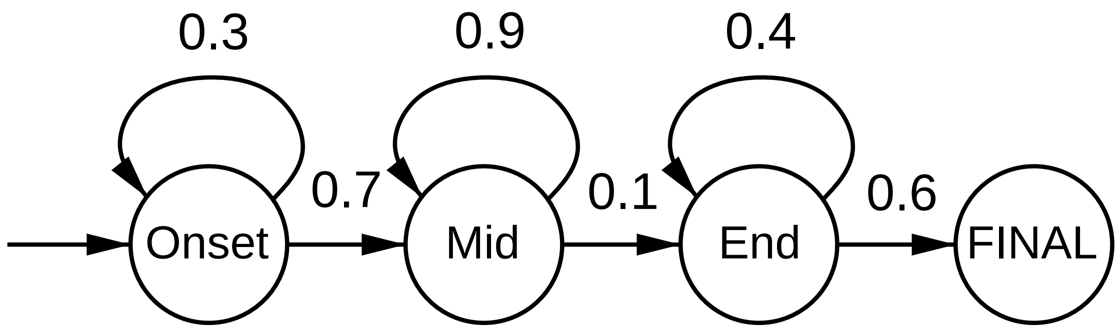
Three-state phones: each phone has three phases (Onset, Mid, End)
E.g., [t] has silent Onset, explosive Mid, hissing End
$\Rightarrow P(features|phone, phase)$

Triphone context: each phone becomes n^2 distinct phones, depending on the phones to its left and right
E.g., [t] in "star" is written [t(s,aa)] (different from "tar"!)

Triphones useful for handling coarticulation effects: the articulators have inertia and cannot switch instantaneously between positions
E.g., [t] in "eighth" has tongue against front teeth

Phone model example

Phone HMM for [m]:



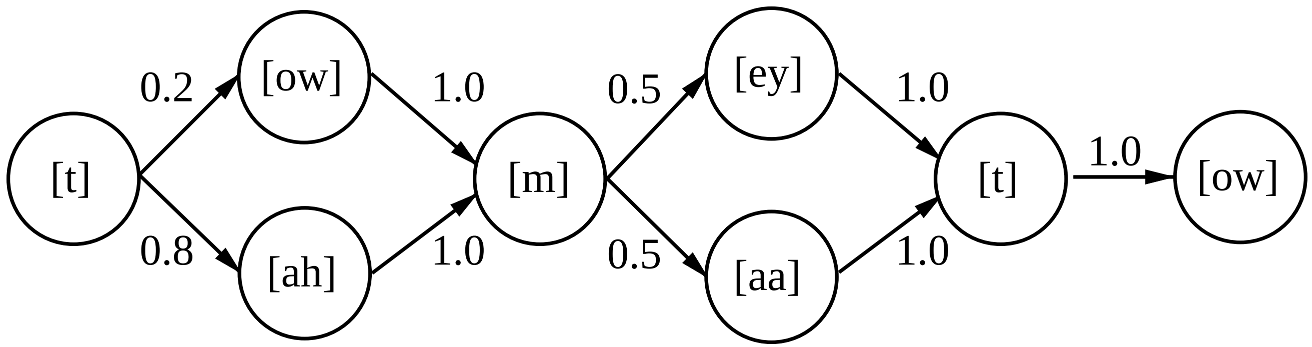
Output probabilities for the phone HMM:

Onset:	Mid:	End:
C1: 0.5	C3: 0.2	C4: 0.1
C2: 0.2	C4: 0.7	C6: 0.5
C3: 0.3	C5: 0.1	C7: 0.4

Word pronunciation models

Each word is described as a distribution over phone sequences

Distribution represented as an HMM transition model



$$P([towmeytow] | \text{“tomato”}) = P([towmaatow] | \text{“tomato”}) = 0.1$$
$$P([tahmeytow] | \text{“tomato”}) = P([tahmaatow] | \text{“tomato”}) = 0.4$$

Structure is created manually, transition probabilities learned from data

Isolated words

Phone models + word models fix likelihood $P(e_{1:t}|word)$ for any isolated word

$$P(word|e_{1:t}) = \alpha P(e_{1:t}|word)P(word)$$

Prior probability $P(word)$ obtained simply by counting word frequencies

$P(e_{1:t}|word)$ can be computed recursively: define

$$\boldsymbol{\ell}_{1:t} = \mathbf{P}(\mathbf{X}_t, \mathbf{e}_{1:t})$$

and use the recursive update

$$\boldsymbol{\ell}_{1:t+1} = \textsc{Forward}(\ell_{1:t}, \mathbf{e}_{t+1})$$

and then $P(e_{1:t}|word) = \sum_{\mathbf{x}_t} \boldsymbol{\ell}_{1:t}(\mathbf{x}_t)$

Isolated-word dictation systems with training reach 95–99% accuracy

Continuous speech

Not just a sequence of isolated-word recognition problems!

– Adjacent words highly correlated
– Sequence of most likely words $\neq$ most likely sequence of words
– Segmentation: there are few gaps in speech
– Cross-word coarticulation—e.g., "next thing"

Continuous speech systems manage 60–80% accuracy on a good day

Language model

Prior probability of a word sequence is given by chain rule:

$$P(w_1 \cdots w_n) = \prod_{i=1}^{n} P(w_i | w_1 \cdots w_{i-1})$$

Bigram model:

$$P(w_i | w_1 \cdots w_{i-1}) \approx P(w_i | w_{i-1})$$

Train by counting all word pairs in a large text corpus

More sophisticated models (trigrams, grammars, etc.) help a little bit

Combined HMM for Continuous Speech

States of the combined language+word+phone model are labelled by the word we're in + the phone in that word + the phone state in that phone

E.g., $[m]^{tomato}_{Onset}$ of $[ey]^{money}_{Mid}$

If each word has average of p three-state phones in its pronunciation model, and there are W words, then the continuous-speech HMM has $3pW$ states.

Transitions can occur:

- Between phone states within a given phone
- Between phones in a given word
- Between final state of one word and initial state of the next

Transitions between words occur with probabilities specified by bigram model.

Solving HMM

Viterbi algorithm (eqn. 15.9) finds the most likely **phone state** sequence.

From the state sequence, we can extract word sequence by just reading off word labels from the states.

Does segmentation by considering all possible word sequences and boundaries

Doesn't always give the most likely word sequence because
each word sequence is the sum over many state sequences

Jelinek invented A* in 1969 a way to find most likely word sequence
where "step cost" is $-\log P(w_i|w_{i-1})$

Summary

Temporal models use state and sensor variables replicated over time

Markov assumptions and stationarity assumption, so we need
– 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; used 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