# Temporal probability models

## Chapter 15, Sections 4–5

# Outline

◊ Inference: filtering, prediction, smoothing

◊ Hidden Markov models

# Hidden Markov models

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

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

Sensor(Emission) 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 $O(St)$

# Improve Inference 1: Country dance algorithm

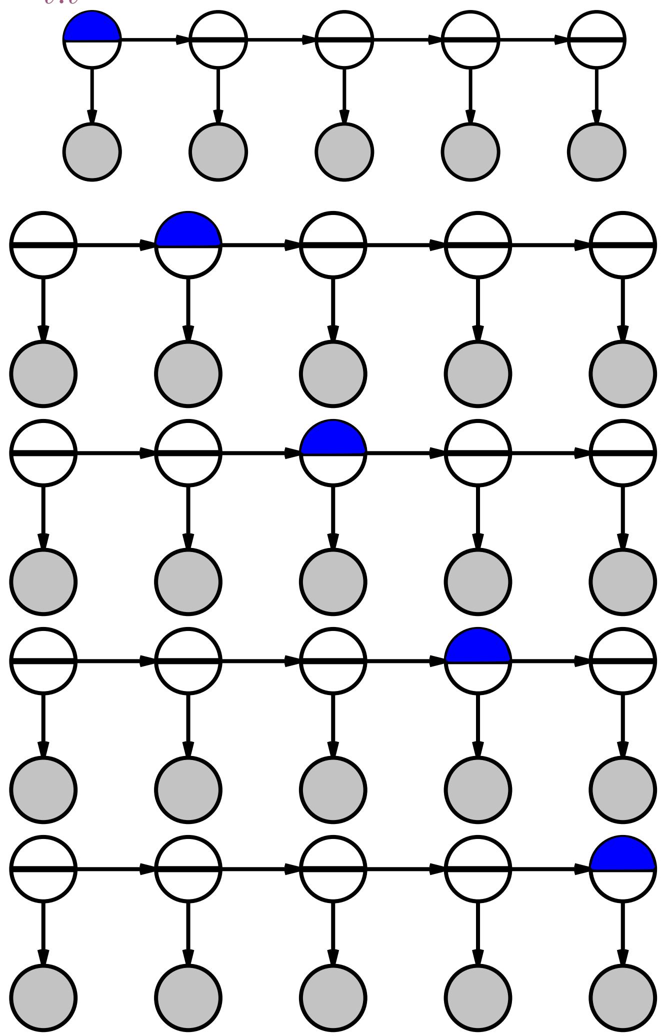
Allows smoothing to be carried out in constant space, independently of sequence length. Can avoid storing all forward messages in smoothing by running
forward algorithm backwards:

$$\begin{aligned}
\mathbf{f}_{1:t+1} &= \alpha \mathbf{O}_{t+1} \mathbf{T}^\top \mathbf{f}_{1:t} \\
\mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \mathbf{T}^\top \mathbf{f}_{1:t} \\
\alpha' (\mathbf{T}^\top)^{-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t}
\end{aligned}$$

Algorithm: for a sequence of length $t$, the forward pass computes $\mathbf{f}_{t:t}$ (forgetting all intermediate results), backward pass computes $\mathbf{f}_i$, $\mathbf{b}_i$ simultaneously

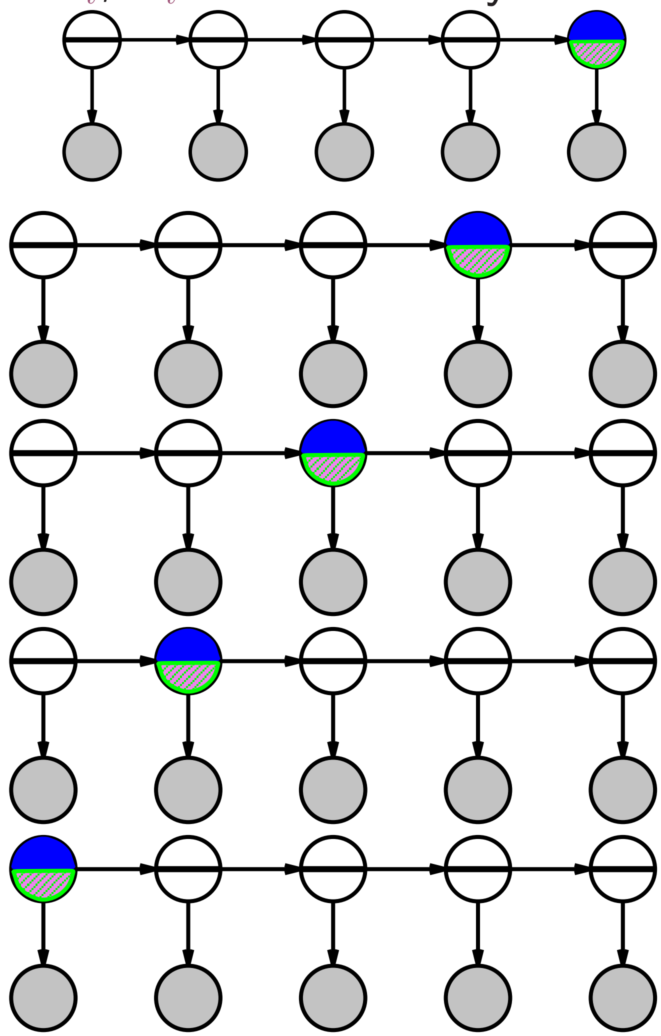
# Country dance algorithm

forward pass computes $\mathbf{f}_{t:t}$

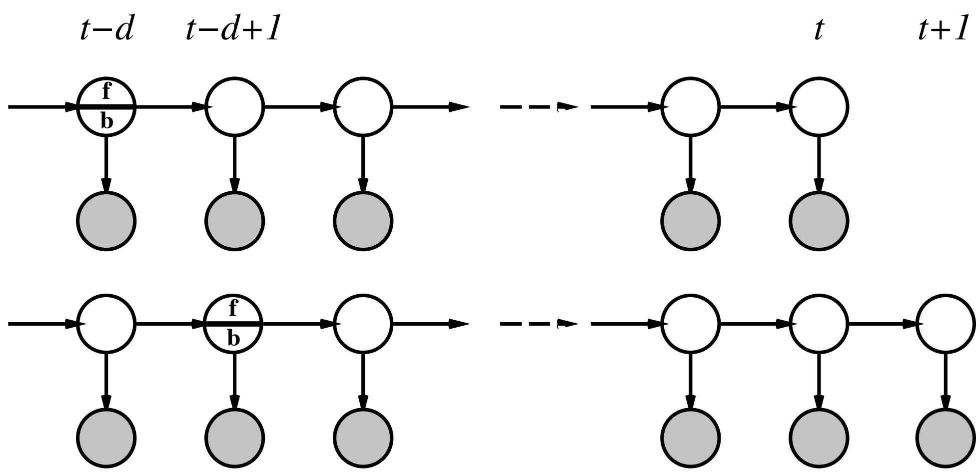


# Country dance algorithm

backward pass computes $\mathbf{f}_i$, $\mathbf{b}_i$ simultaneously



# Improve Inference 2: Fixed-lag smoothing




Obvious method runs forward–backward for $d$ steps each time

When new observatio arrives, recursively compute $\alpha\mathbf{f}_{1:t-d+1}$x$\mathbf{b}_{t-d+2:t+1}$ for slice $t-d+1$ from $\alpha\mathbf{f}_{1:t-d}$x$\mathbf{b}_{t-d+1:t}$.

Forward message $\mathbf{f}_{1:t-d+1}$ from, $\mathbf{f}_{1:t-d}$ using standard filtering process.
Backward message not directly obtainable

# Online fixed-lag smoothing contd.

Define $\mathbf{B}_{j:k} = \prod_{i=j}^{k} \mathbf{T}\mathbf{O}_i$, so

$$\begin{aligned} \mathbf{b}_{t-d+1:t} &= \mathbf{B}_{t-d+1:t}\mathbf{1} \\ \mathbf{b}_{t-d+2:t+1} &= \mathbf{B}_{t-d+2:t+1}\mathbf{1} \end{aligned}$$

Now we can get a recursive update for $\mathbf{B}$:

$$\mathbf{B}_{t-d+2:t+1} = \mathbf{O}_{t-d+1}^{-1}\mathbf{T}^{-1}\mathbf{B}_{t-d+1:t}\mathbf{T}\mathbf{O}_{t+1}$$

Hence update cost is constant, independent of lag $d$

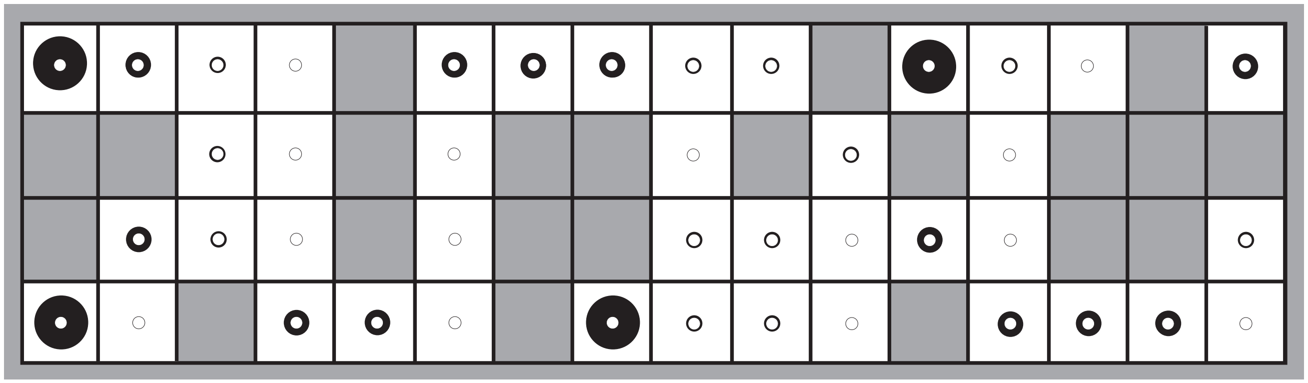
# Online fixed-lag smoothing algorithm

**function** FIXED-LAG-SMOOTHING($e_t$, $hmm$, $d$) **returns** a distribution over $\mathbf{X}_{t-d}$
  **inputs**: $e_t$, the current evidence for time step $t$
    $hmm$, a hidden Markov model with $S \times S$ transition matrix $\mathbf{T}$
    $d$, the length of the lag for smoothing
  **persistent**: $t$, the current time, initially 1
    **f**, the forward message $\mathbf{P}(X_t | e_{1:t})$, initially $hmm$.PRIOR
    **B**, the $d$-step backward transformation matrix, initially the identity matrix
    $e_{t-d:t}$, double-ended list of evidence from $t-d$ to $t$, initially empty
  **local variables**: $\mathbf{O}_{t-d}$, $\mathbf{O}_t$, diagonal matrices containing the sensor model information

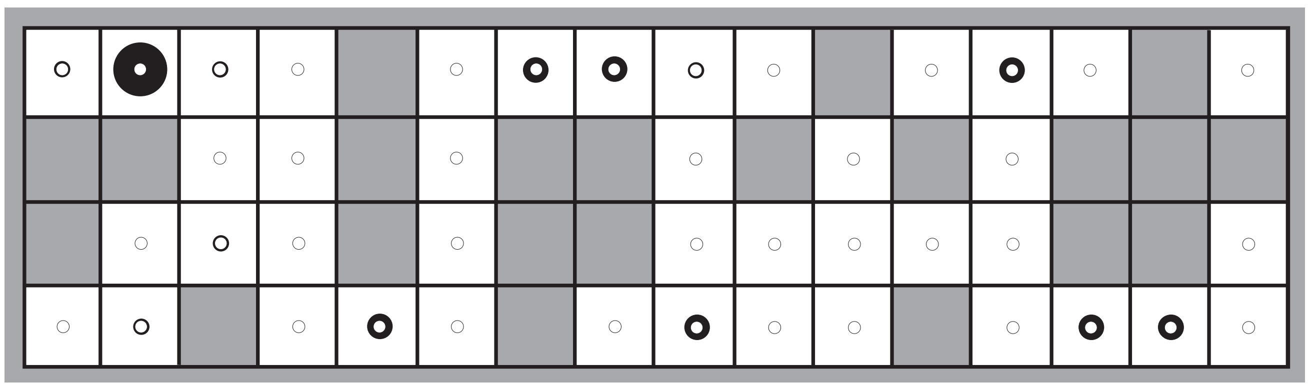
  add $e_t$ to the end of $e_{t-d:t}$
  $\mathbf{O}_t \leftarrow$ diagonal matrix containing $\mathbf{P}(e_t | X_t)$
  **if** $t > d$ **then**
    $\mathbf{f} \leftarrow$ FORWARD($\mathbf{f}$, $e_t$)
    remove $e_{t-d-1}$ from the beginning of $e_{t-d:t}$
    $\mathbf{O}_{t-d} \leftarrow$ diagonal matrix containing $\mathbf{P}(e_{t-d} | X_{t-d})$
    $\mathbf{B} \leftarrow \mathbf{O}^{-1}_{t-d}\mathbf{T}^{-1}\mathbf{B}\mathbf{T}\mathbf{O}_t$
  **else** $\mathbf{B} \leftarrow \mathbf{B}\mathbf{T}\mathbf{O}_t$
  $t \leftarrow t + 1$
  **if** $t > d$ **then return** NORMALIZE($\mathbf{f} \times \mathbf{B1}$) **else return** null

**Figure 15.6** An algorithm for smoothing with a fixed time lag of $d$ steps, implemented as an online algorithm that outputs the new smoothed estimate given the observation for a new time step. Notice that the final output NORMALIZE($\mathbf{f} \times \mathbf{B1}$) is just $\alpha\, \mathbf{f} \times \mathbf{b}$, by Equation (15.14)

# HMM example: Localization



(a) Posterior distribution over robot location after $E_1 = NSW$



(b) Posterior distribution over robot location after $E_1 = NSW, E_2 = NS$

# Outline

◇ Kalman filters

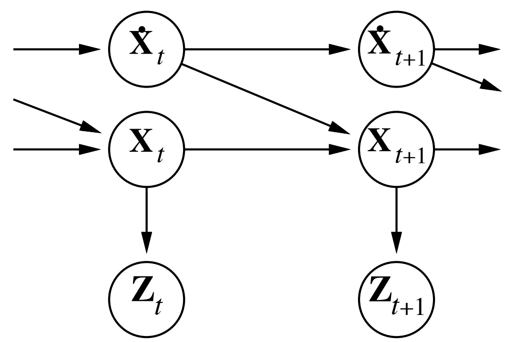
◇ Dynamic Bayes network

◇ Partical filtering

# Kalman filters

"The Kalman filter, also known as linear quadratic estimation (LQE), is an algorithm which uses a series of measurements observed over time, that containing noise, and produces estimates of unknown variables that tend to be more precise than single measurement alone." (wikipedia)

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 current $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$ is Gaussian and transition model $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)$ is linear Gaussian, then one step 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 prediction $\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$ is Gaussian and the sensor mdoel $\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})$ 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 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 $t \to \infty$

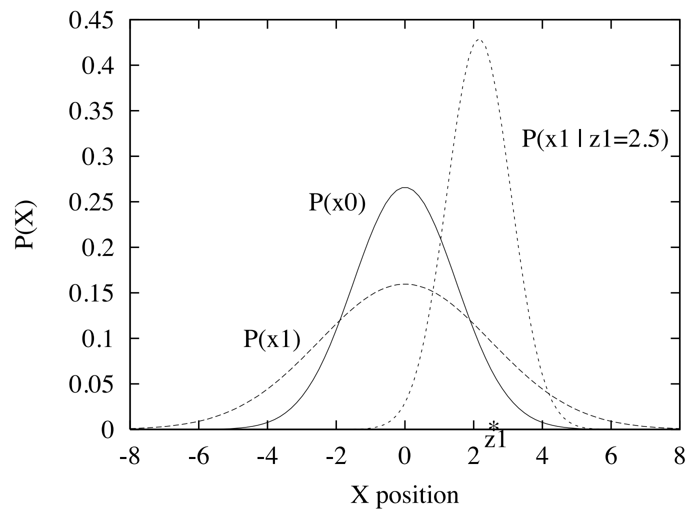
* linear Gaussian: linear model with Gaussian noise $Y = aX + N(\mu, \sigma)$

# Simple 1-D example

Gaussian random walk on $X$–axis, s.d. $\sigma_x$, sensor s.d. $\sigma_z$

Prior: $\mathbf{P}(x_0) = N(\mu_0, \sigma_0)$
Transition model: $\mathbf{P}(x_{t+1}|x_t) = N(\mathbf{x}_t, \sigma_x)$
Sensor model: $\mathbf{P}(z_{t+1}|x_{t+1}) = N(\mathbf{x}_{t+1}, \sigma_z)$

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



# 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

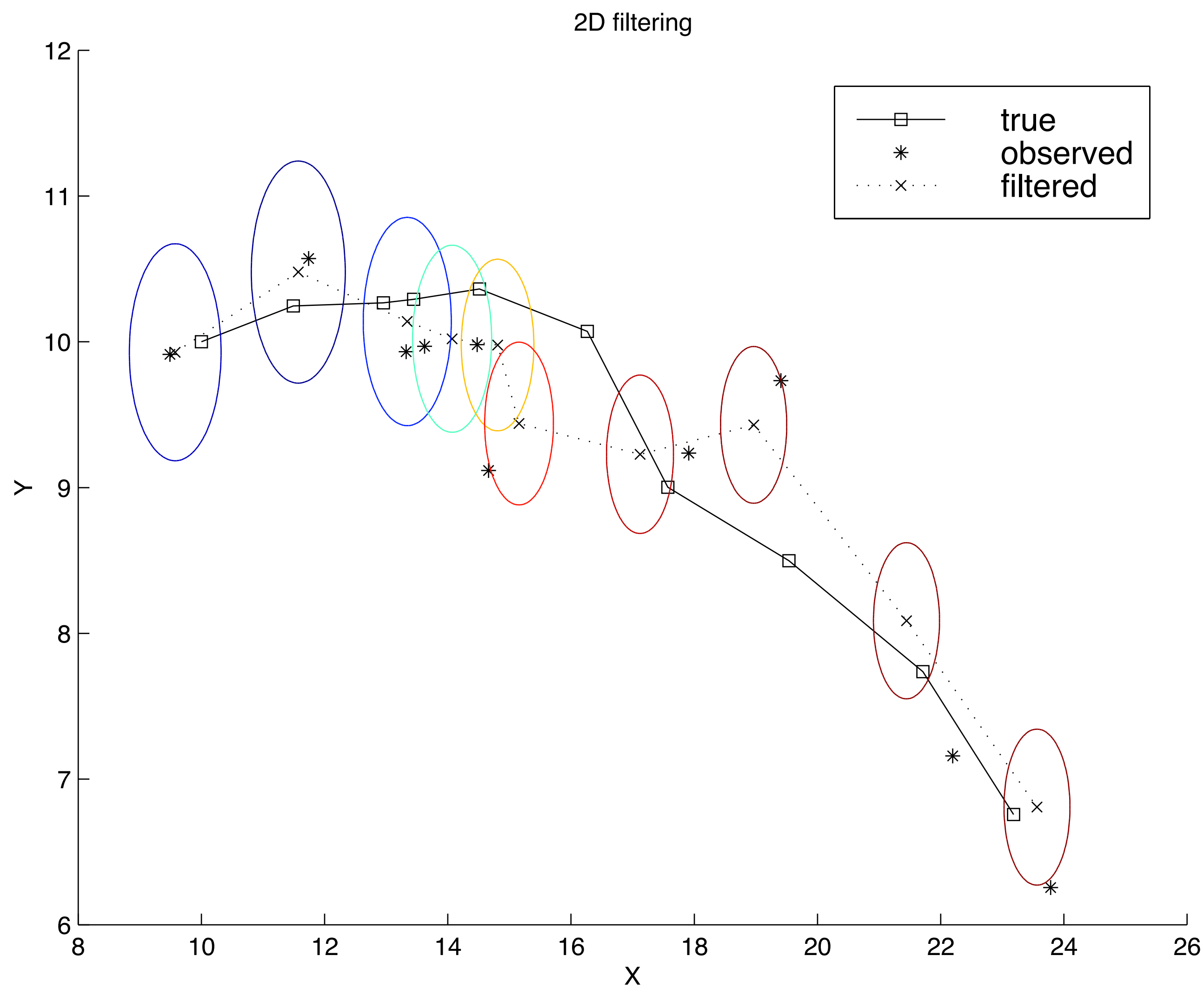
Filter computes the following update:

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

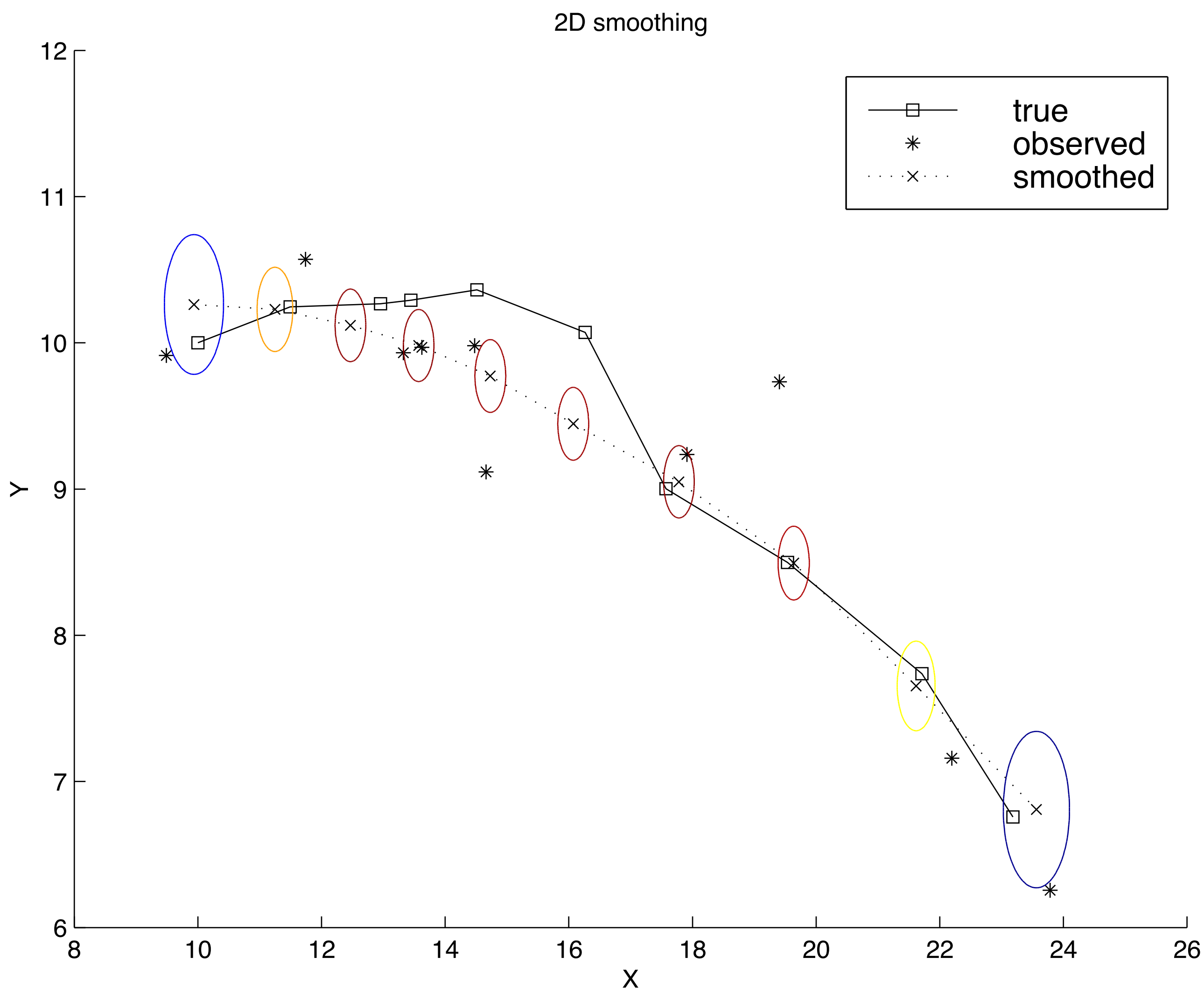
where $\mathbf{K}_{t+1} = (\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^\top + \mathbf{\Sigma}_x)\mathbf{H}^\top(\mathbf{H}(\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^\top + \mathbf{\Sigma}_x)\mathbf{H}^\top + \mathbf{\Sigma}_z)^{-1}$
is the Kalman gain matrix

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

# 2-D tracking example: filtering



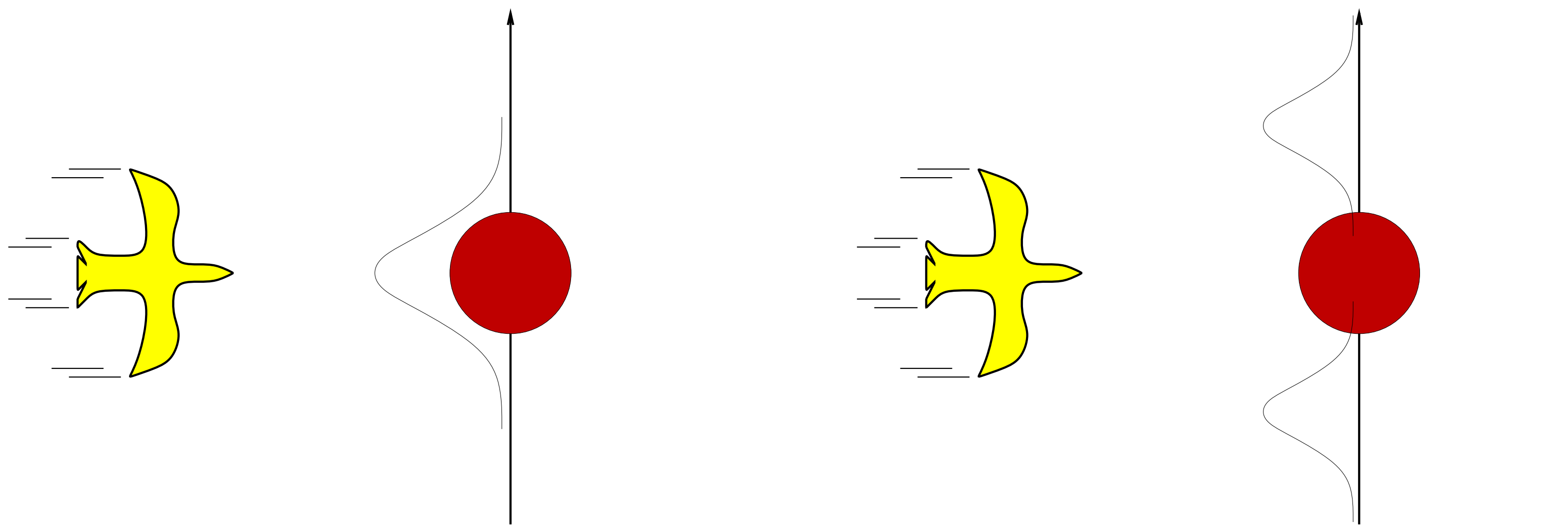
# 2-D tracking example: smoothing



# Where it breaks

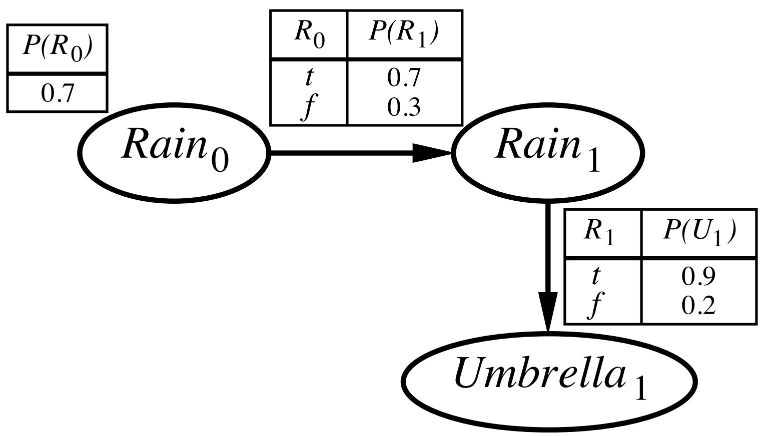
Cannot be applied if the transition model is nonlinear

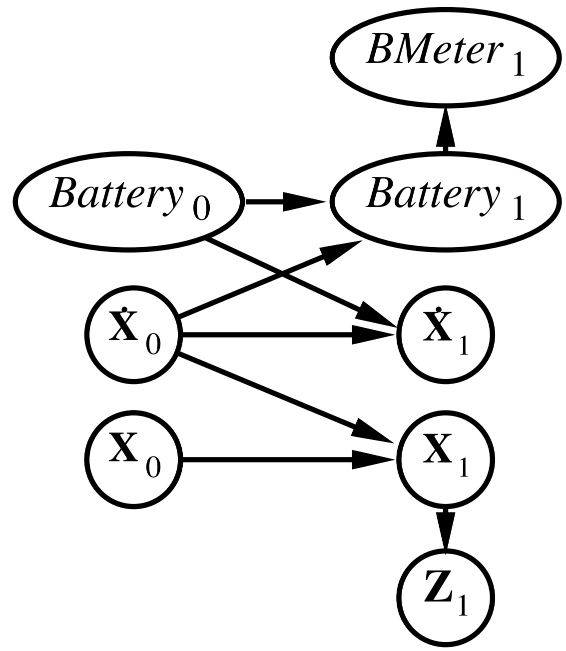
Extended Kalman Filter models transition as **locally linear** around $\mathbf{x}_t = \boldsymbol{\mu}_t$
Fails if systems is locally unsmooth



# Dynamic Bayesian networks

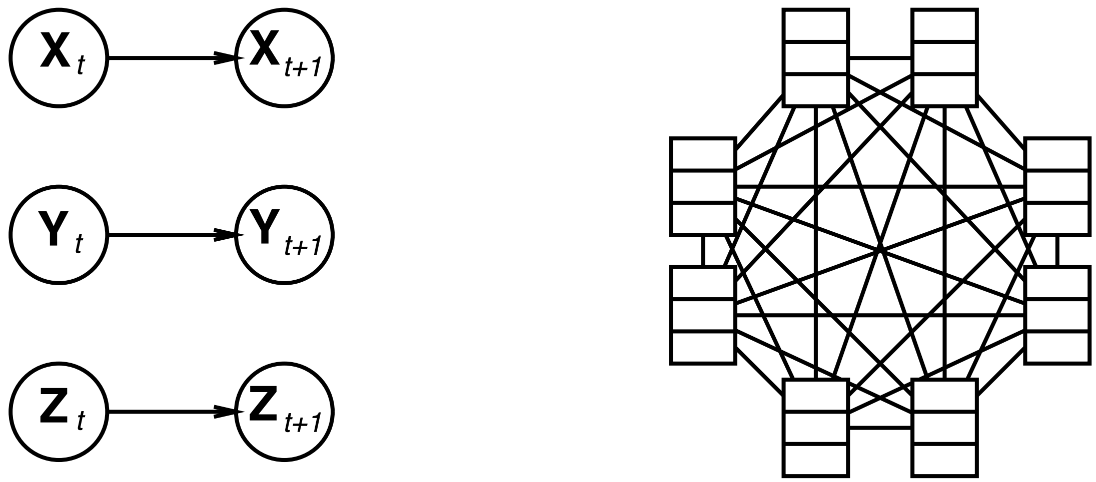
$\mathbf{X}_t$, $\mathbf{E}_t$ contain arbitrarily many variables in a replicated Bayes net



# DBNs vs. HMMs

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




Sparse dependencies $\Rightarrow$ exponentially fewer parameters;

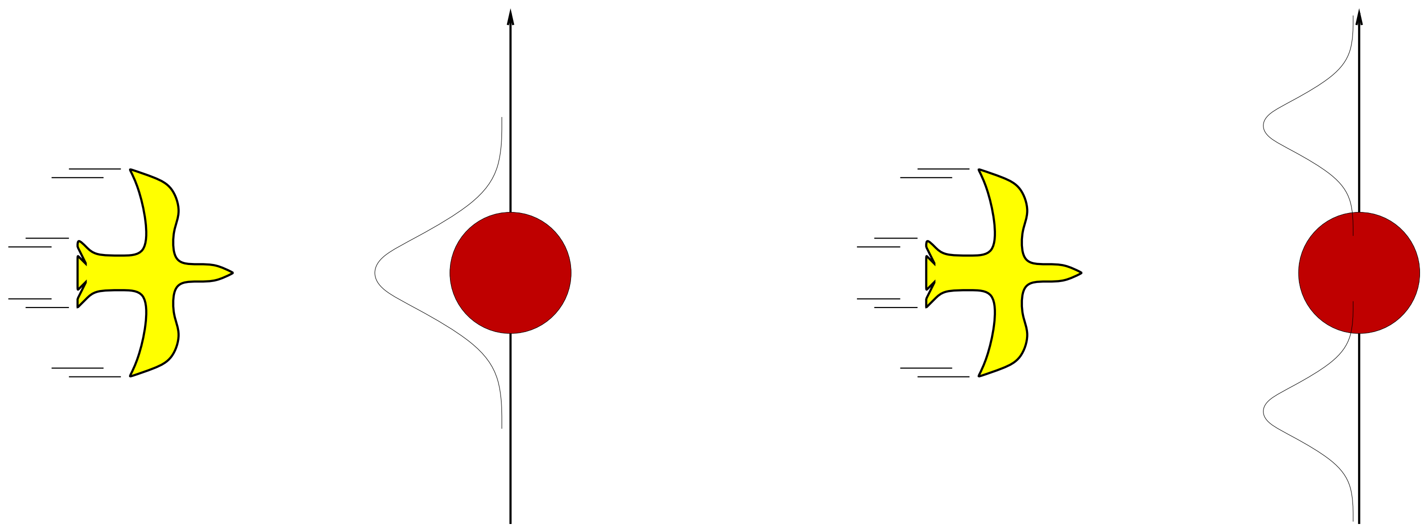
e.g., 20 state variables, three parents each

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?



# Constructing DBNs

requires:

- ♢ prior distribution over stat variables: $\mathbf{P}(X_0)$
- ♢ transition model: $\mathbf{P}(X_{t+1}|X_t)$
- ♢ sensor model: $\mathbf{P}(E_t|X_t)$

* If we assume the models are stationary, they need to be specified only once.

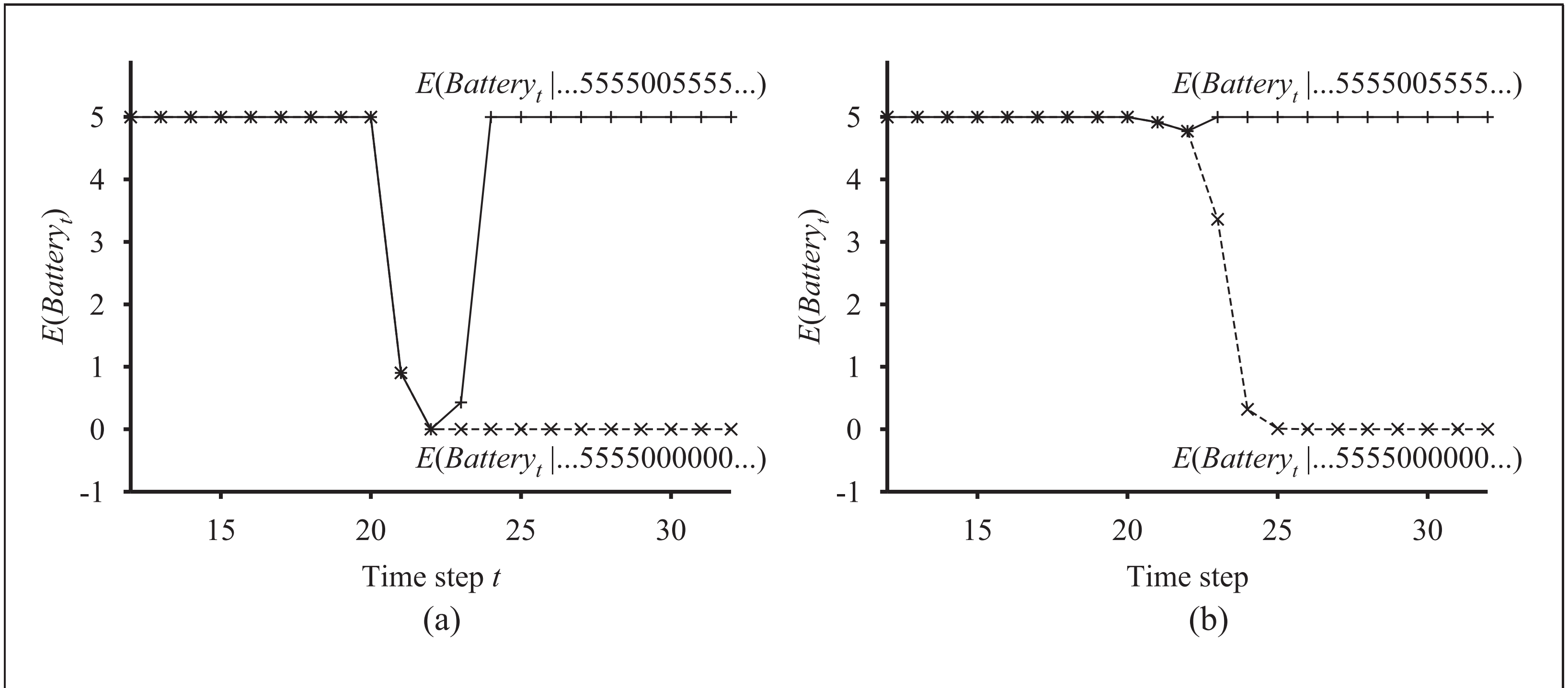
Sensor model in more detail:

- ♢ Perfect sensor
- ♢ Sensor with noisy reading: Gaussian error model
- ♢ Temporal failure in sensor: Transient failure model

For the system to handle sensor failur properly , the sensor model must include the possibility of failure: ex. $\mathbf{P}(BMeter_t = 0)|Battery_t = 5) = 0.03$ prob. larger than prob. of Gaussian error model

# Constructing DBNs cont.

Gaussian error model vs Transient failure model

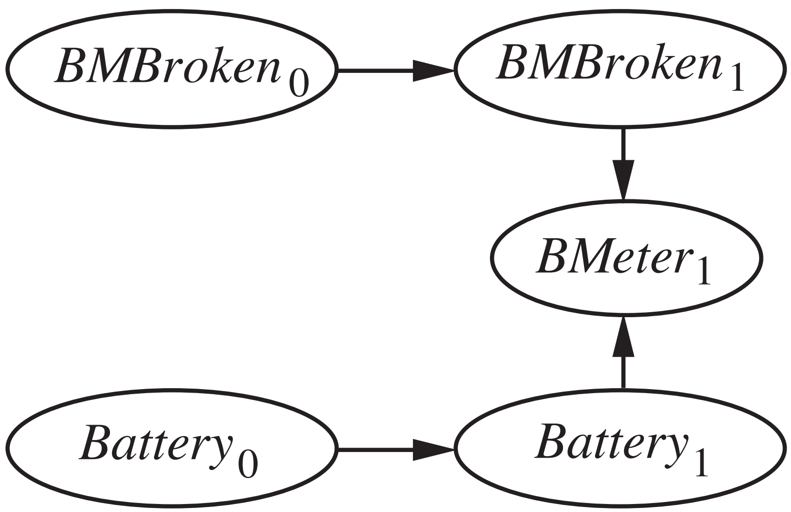


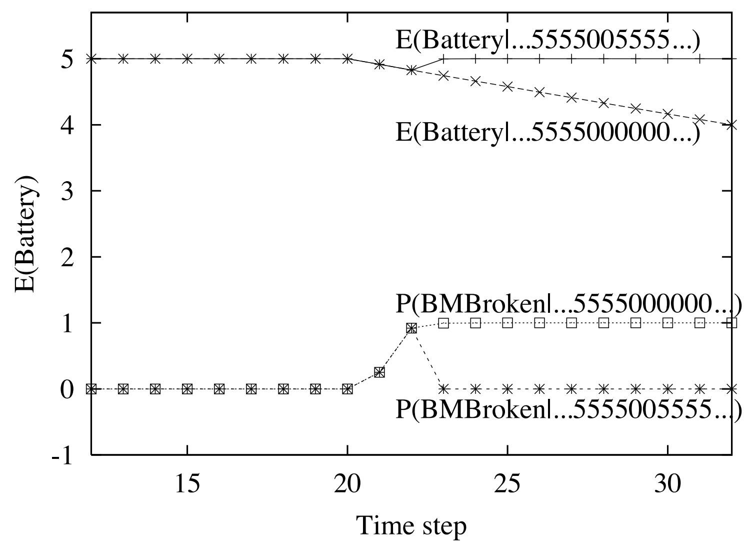
# Constructing DBNs cont.

◇ Persistant failure in sensor: Persistant failure model

Transient failure model vs Persistant failure model

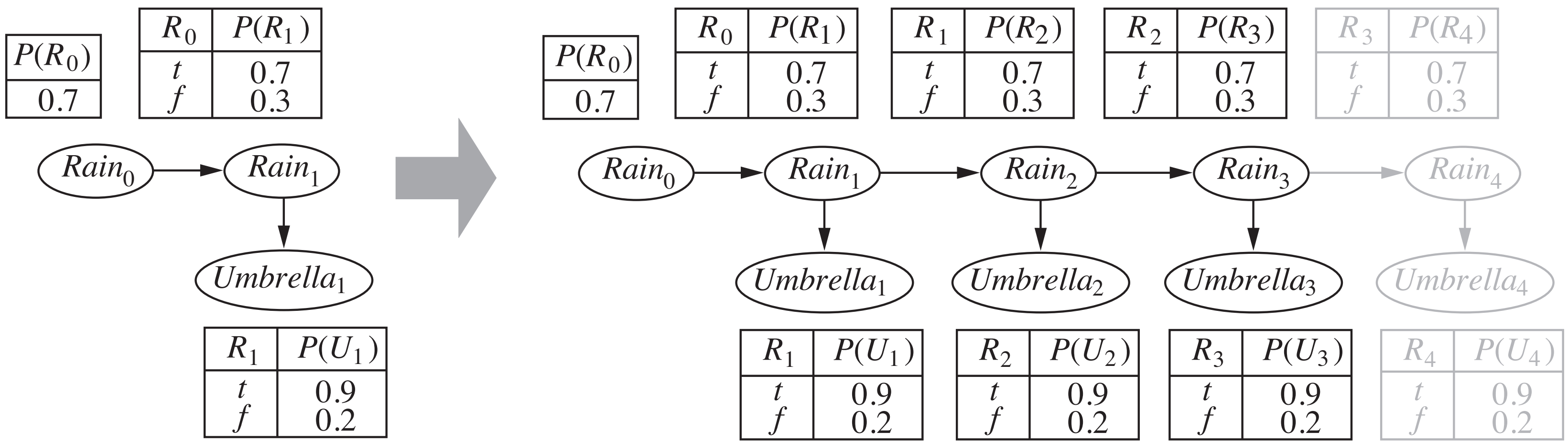
| $B_0$ | $P(B_1)$ |
|---|---|
| $t$ | 1.000 |
| $f$ | 0.001 |



# Exact inference in DBNs

Naive method: unroll the network and run any exact algorithm



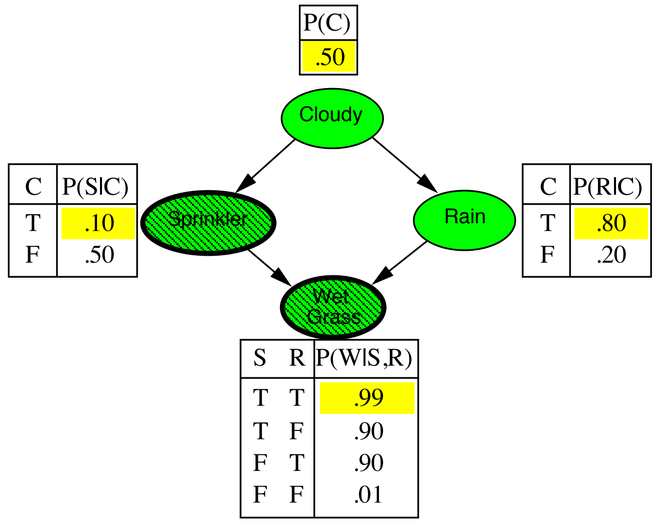

problem: inference cost for each update grows with $t$

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 $O(d^{2n})$)

# Likelihood weighting analysis review(14.5)

Sample the nonevidence nodes of the network in topological order, weighting each sample by the likelihood it accords to the observed evidence variables.



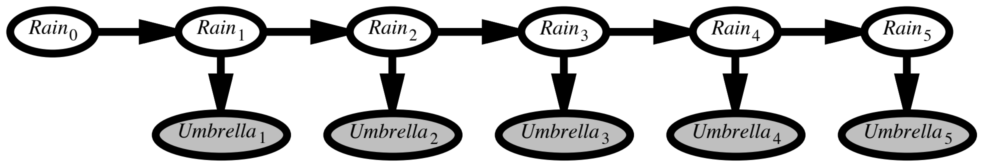

$w = 1.0 \times 0.1 \times 0.99 = 0.099$

Weighted sampling probability is

$$S_{WS}(\mathbf{z}, \mathbf{e}) w(\mathbf{z}, \mathbf{e})$$
$$= \prod_{i=1}^{l} P(z_i | parents(Z_i)) \;\; \prod_{i=1}^{m} P(e_i | parents(E_i))$$

# Likelihood weighting for DBNs

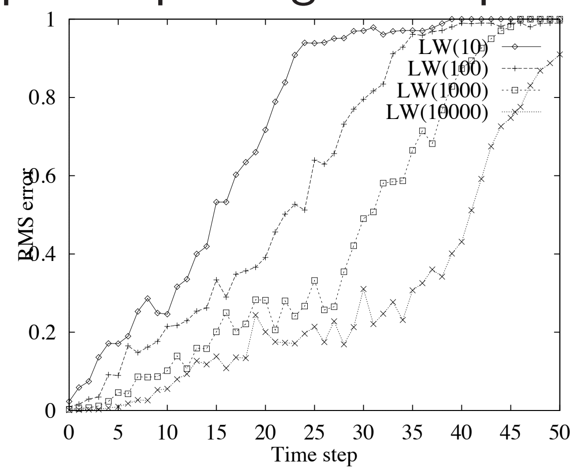
Set of weighted samples approximates the belief state




LW samples pay no attention to the evidence!

$\Rightarrow$ fraction "agreeing" falls exponentially with $t$

$\Rightarrow$ number of samples required grows exponentially with $t$



# Moditication to likelihood weight

Basic idea:

♢ 1) run all N samples together thorugh the DBN , one slice at at time

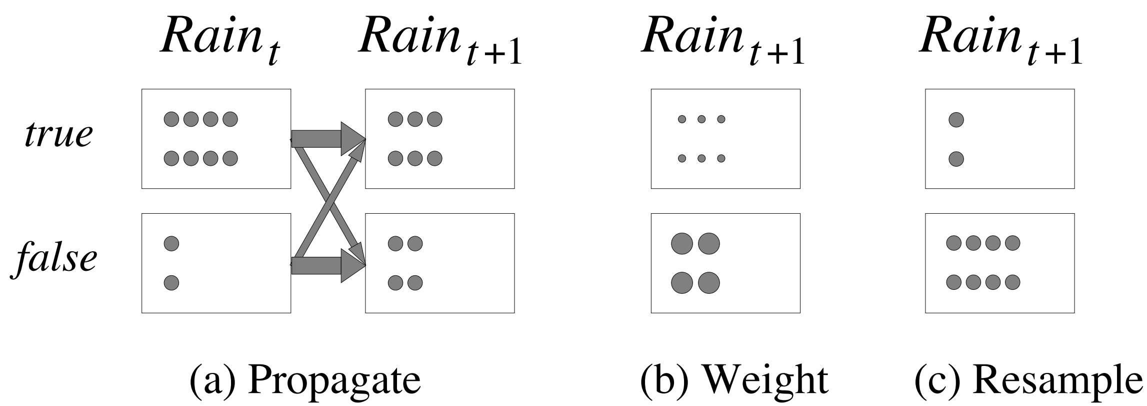
♢ 2) use the samples themselves as an approximate reprresenation of the current state distribution.

♢ 3) ensure that the population of samples (“particles”) tracks the high-likelihood regions of the state-space

by focusing the set of samples in the high-probability regions of the state space.

# Particle filtering

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




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

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

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 algorithm

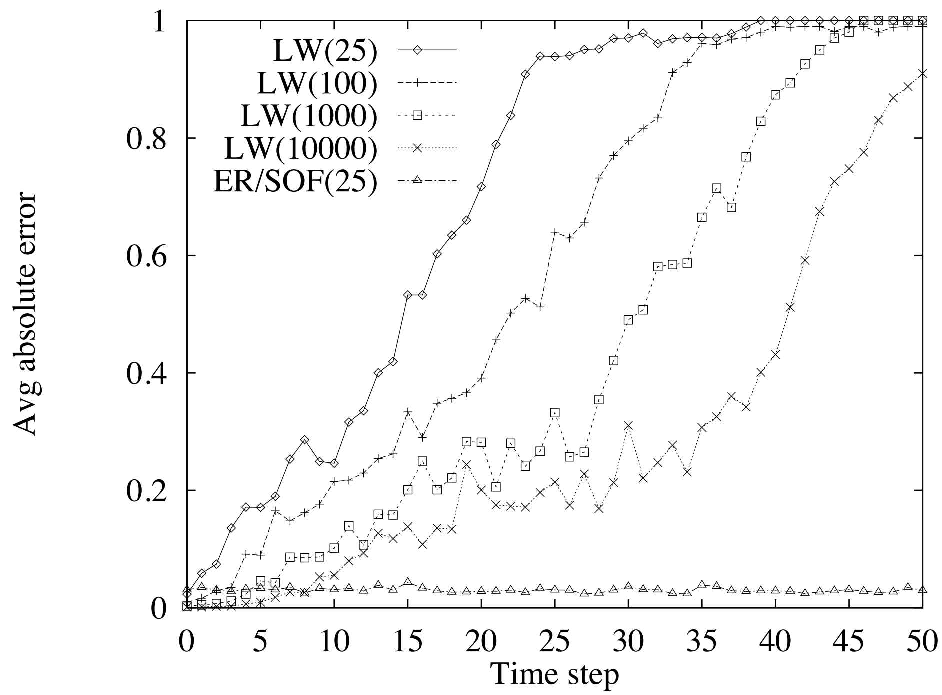
```
function PARTICLE-FILTERING(e, N, dbn) returns a set of samples for the next time step
   inputs: e, the new incoming evidence
           N, the number of samples to be maintained
           dbn, a DBN with prior P(X_0), transition model P(X_1|X_0), sensor model P(E_1|X_1)
   persistent: S, a vector of samples of size N, initially generated from P(X_0)
   local variables: W, a vector of weights of size N

   for i = 1 to N do
       S[i] ← sample from P(X_1 | X_0 = S[i])   /* step 1 */
       W[i] ← P(e | X_1 = S[i])                /* step 2 */
   S ← WEIGHTED-SAMPLE-WITH-REPLACEMENT(N, S, W)      /* step 3 */
   return S
```

**Figure 15.17** The particle filtering algorithm implemented as a recursive update operation with state (the set of samples). Each of the sampling operations involves sampling the relevant slice variables in topological order, much as in PRIOR-SAMPLE. The WEIGHTED-SAMPLE-WITH-REPLACEMENT operation can be implemented to run in $O(N)$ expected time. The step numbers refer to the description in the text.

# Particle filtering performance

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



# 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