

CS 561: Artificial Intelligence

Instructor: Sofus A. Macskassy, macskass@usc.edu

TAs: Nadeesha Ranashinghe (nadeeshr@usc.edu)

William Yeoh (wyeoh@usc.edu)

Harris Chiu (chiciu@usc.edu)

Lectures: MW 5:00-6:20pm, OHE 122 / DEN

Office hours: By appointment

Class page: <http://www-rcf.usc.edu/~macskass/CS561-Spring2010/>

This class will use <http://www.uscden.net/> and class webpage

- Up to date information
- Lecture notes
- Relevant dates, links, etc.

Course material:

[AIMA] Artificial Intelligence: A Modern Approach,
by Stuart Russell and Peter Norvig. (2nd ed)

Temporal Probability Models [Ch 15]

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

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

X_t = set of unobservable state variables at time t

e.g., $BloodSugar_t$, $StomachContents_t$, etc.

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: $X_{a:b} = X_a, X_{a+1}, \dots, X_{b-1}, X_b$

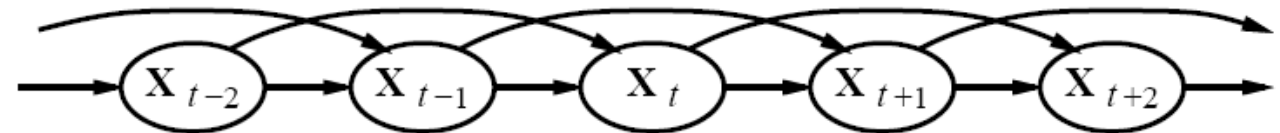
Markov processes (Markov chains)

- 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:

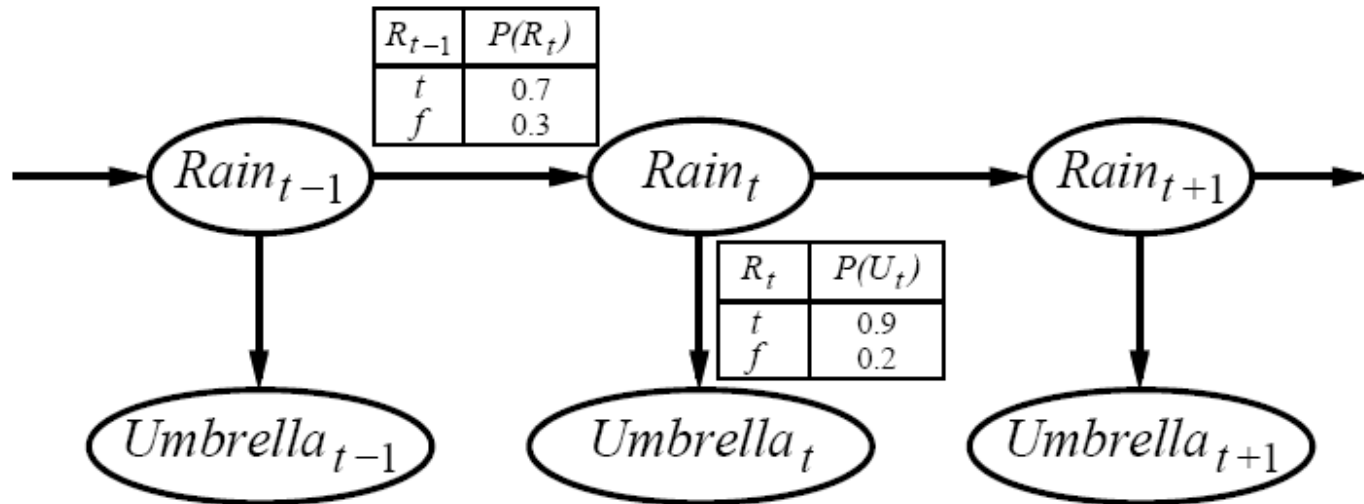


• Second-order:



- 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



- 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 $0 < k < t$
 - 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:

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

$$P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t+1}) = P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}, \mathbf{e}_{t+1}) \quad \text{Divide up evidence}$$

$$= \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}, \mathbf{e}_{1:t}) P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) \quad \text{Using Bayes rule}$$

$$= \alpha P(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) P(\mathbf{X}_{t+1} | \mathbf{e}_{1:t}) \quad \text{By Markov property}$$

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

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

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

- $\mathbf{f}_{1:t+1} = \alpha \text{Forward}(\mathbf{f}_{1:t}, \mathbf{e}_{t+1})$ where $\mathbf{f}_{1:t} = P(\mathbf{X}_t | \mathbf{e}_{1:t})$
Time and space **constant** (independent of t)

Filtering example

Day 0:

$$P(R_0) = \langle 0.5, 0.5 \rangle$$

Day 1 (Umbrella appears $\rightarrow U_1 = \text{true}$)

$$\begin{aligned} P(R_1) &= \sum_{r_0} P(R_1 | r_0) P(r_0) \\ &= \langle 0.7, 0.3 \rangle \times 0.5 + \langle 0.3, 0.7 \rangle \times 0.5 \\ &= \langle \mathbf{0.5}, \mathbf{0.5} \rangle \end{aligned}$$

updating with evidence for $t=1$ gives:

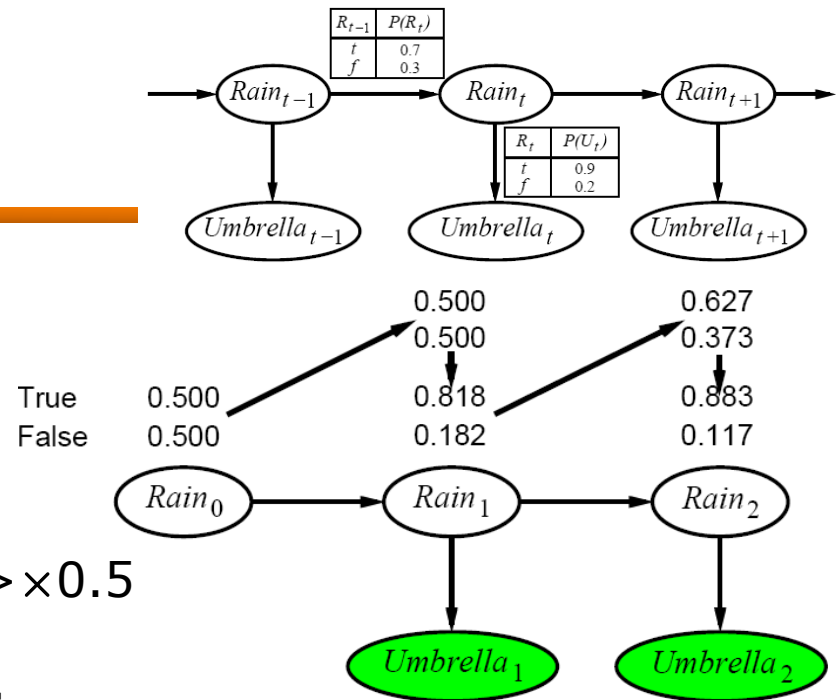
$$\begin{aligned} P(R_1 | u_1) &= \alpha P(u_1 | R_1) P(R_1) = \alpha \langle 0.9, 0.2 \rangle \times \langle 0.5, 0.5 \rangle \\ &= \alpha \langle 0.45, 0.1 \rangle \approx \langle \mathbf{0.818}, \mathbf{0.182} \rangle \end{aligned}$$

Day 2 (Umbrella appears $\rightarrow U_2 = \text{true}$)

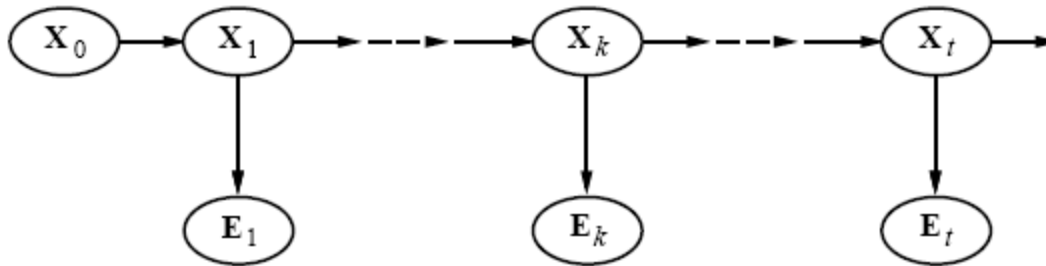
$$\begin{aligned} P(R_2 | u_1) &= \sum_{r_1} P(R_2 | r_1) P(r_1 | u_1) \\ &= \langle 0.7, 0.3 \rangle \times 0.818 + \langle 0.3, 0.7 \rangle \times 0.182 \approx \langle \mathbf{0.627}, \mathbf{0.373} \rangle \end{aligned}$$

updating with evidence for $t=2$ gives:

$$\begin{aligned} P(R_2 | u_1, u_2) &= \alpha P(u_2 | R_2) P(R_2 | u_1) = \alpha \langle 0.9, 0.2 \rangle \times \langle 0.627, 0.373 \rangle \\ &= \alpha \langle 0.565, 0.075 \rangle \approx \langle \mathbf{0.883}, \mathbf{0.117} \rangle \end{aligned}$$



Smoothing



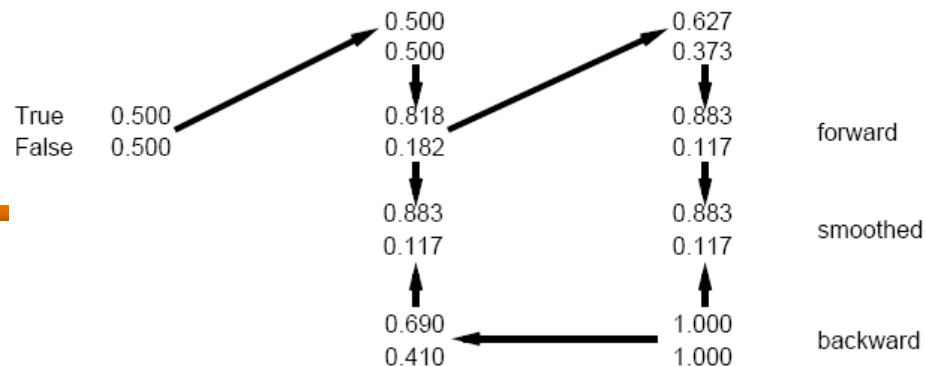
- Divide evidence $\mathbf{e}_{1:t}$ into $\mathbf{e}_{1:k}$, $\mathbf{e}_{k+1:t}$:

$$\begin{aligned} P(\mathbf{X}_k | \mathbf{e}_{1:t}) &= P(\mathbf{X}_k | \mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\ &= \alpha P(\mathbf{X}_k | \mathbf{e}_{1:k}) P(\mathbf{e}_{k+1:t} | \mathbf{X}_k, \mathbf{e}_{1:k}) \quad \text{Using Bayes rule} \\ &= \alpha P(\mathbf{X}_k | \mathbf{e}_{1:k}) P(\mathbf{e}_{k+1:t} | \mathbf{X}_k) \quad \text{Using conditional independence} \\ &= \alpha f_{1:k} b_{k+1:t} \end{aligned}$$

- Backward message computed by a backwards recursion:

$$\begin{aligned} P(\mathbf{e}_{k+1:t} | \mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} | \mathbf{X}_k, \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{X}_k) \quad \text{Conditioning on } X_{k+1} \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{X}_k) \quad \text{Using conditional independence} \\ &= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1} | \mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t} | \mathbf{x}_{k+1}) P(\mathbf{x}_{k+1} | \mathbf{X}_k) = \text{BACKWARD}(b_{k+2:t}, \mathbf{e}_{k+1:t}) \end{aligned}$$

Smoothing example



Compute estimate for rain at $t=1$

$$P(R_1 | u_1, u_2) = \alpha P(R_1 | u_1) P(u_2 | R_1)$$

$$P(R_1 | u_1) = \langle 0.818, 0.182 \rangle$$

$$P(u_2 | R_1) = \sum_{r_2} P(u_2 | r_2) P(r_2 | R_1)$$

$$= (0.9 \times 1 \times \langle 0.7, 0.3 \rangle) + (0.2 \times 1 \times \langle 0.3, 0.7 \rangle)$$

$$= \langle 0.69, 0.41 \rangle$$

Smoothed estimate:

$$P(R_1 | u_1, u_2) = \alpha \langle 0.818, 0.182 \rangle \times \langle 0.69, 0.41 \rangle \approx \langle \mathbf{0.883}, \mathbf{0.117} \rangle$$

- **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 x_{t+1}
= most likely path to **some** x_t plus one more step

$$\begin{aligned} \max_{x_1 \dots x_t} P(x_1, \dots, x_t, X_{t+1} | e_{1:t+1}) \\ = P(e_{t+1} | X_{t+1}) \max_{x_t} (P(X_{t+1} | x_t) \max_{x_1 \dots x_{t-1}} P(x_1, \dots, x_{t-1}, x_t | e_{1:t})) \end{aligned}$$

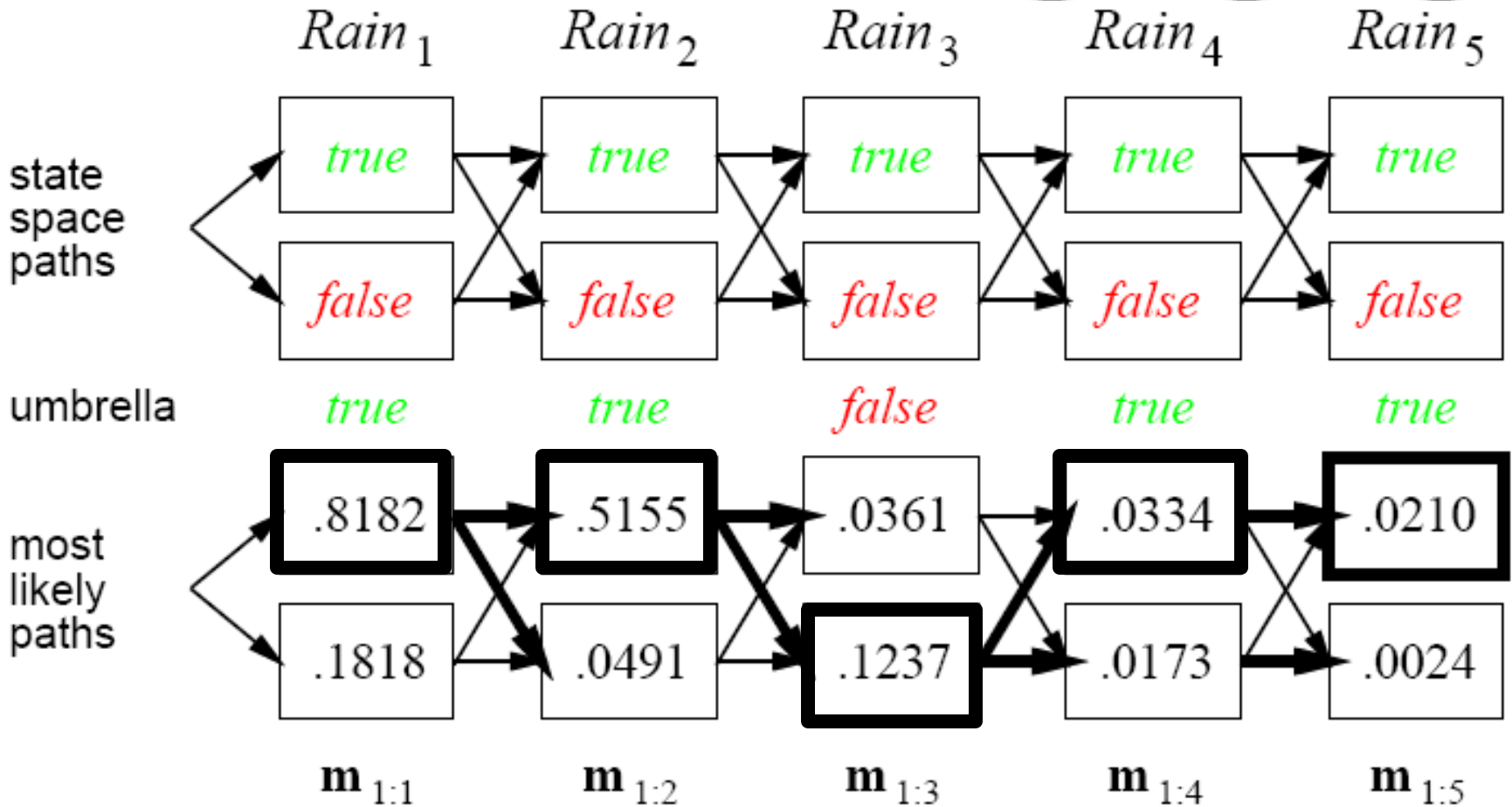
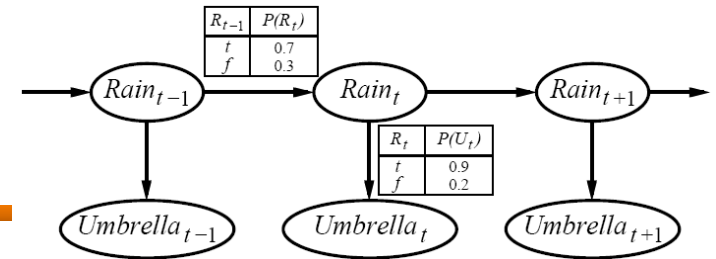
- Identical to filtering, except $f_{1:t}$ replaced by

$$m_{1:t} = \max_{x_1 \dots x_{t-1}} P(x_1, \dots, x_{t-1}, X_t | e_{1:t})$$

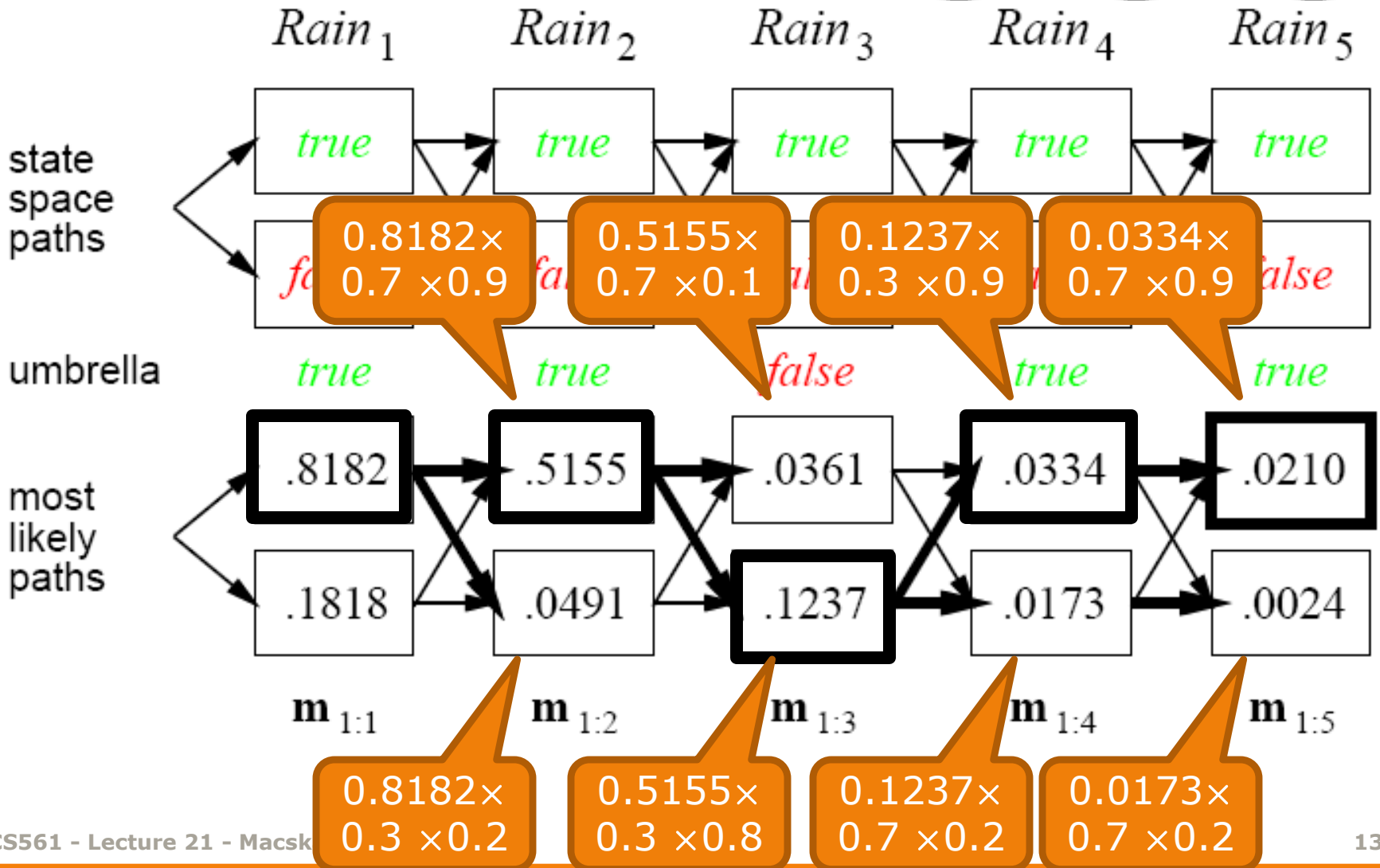
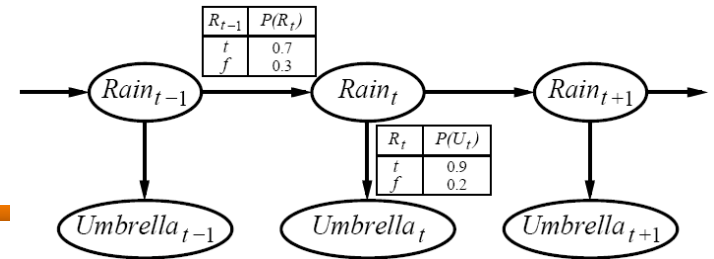
- I.e., $m_{1:t}(i)$ gives the probability of the most likely path to state i .
- Update has sum replaced by max, giving the **Viterbi algorithm**:

$$m_{1:t+1} = P(e_{1:t+1} | X_{t+1}) \max_{x_t} (P(X_{t+1} | x_t) m_{1:t})$$

Viterbi Example



Viterbi Example



Hidden Markov models

- X_t is a single, discrete variable (usually E_t is too)
- Domain of X_t is $\{1, \dots, S\}$
- **Transition matrix** $T_{ij} = P(X_t = j \mid X_{t-1} = i)$, e.g., $\begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$
- **Sensor matrix** O_t for each time step, diagonal elements $P(e_t \mid X_t = i)$
- e.g., with $U_1 = \text{true}$, $O_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}$
- Forward and backward messages as column vectors:
$$\mathbf{f}_{1:t+1} = O_{t+1} \mathbf{T}^\top \mathbf{f}_{1:t}$$
$$\mathbf{b}_{k+1:t} = \mathbf{T} O_{k+1} \mathbf{b}_{k+2:t}$$
- Forward-backward algorithm needs time $O(S^2t)$ and space $O(St)$

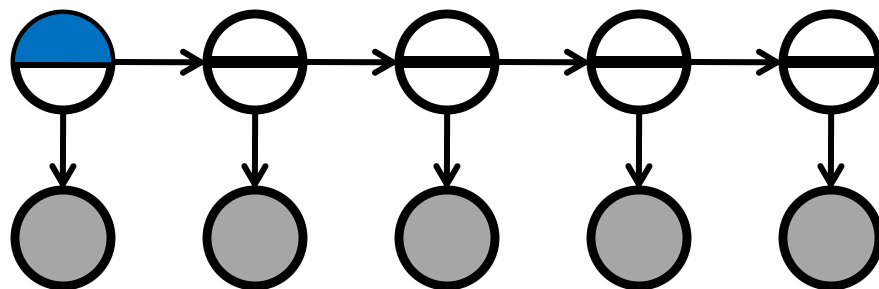
Country dance algorithm

- Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

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

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

- Algorithm: forward pass computes \mathbf{f}_t , backward pass does $\mathbf{f}_i, \mathbf{b}_i$



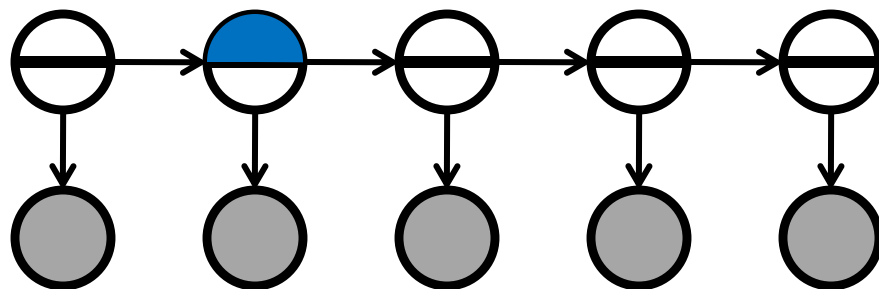
Country dance algorithm

- Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

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

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

- Algorithm: forward pass computes \mathbf{f}_t , backward pass does $\mathbf{f}_i, \mathbf{b}_i$



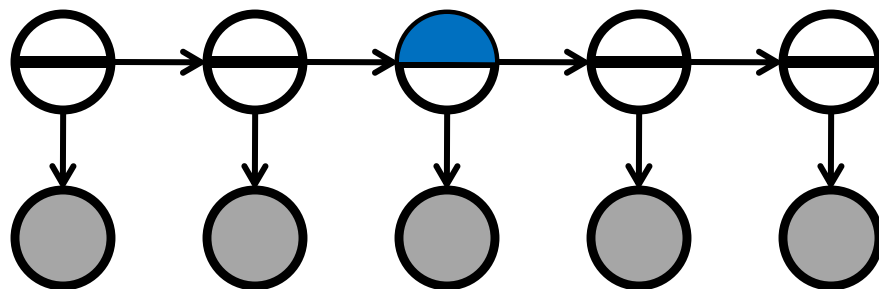
Country dance algorithm

- Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

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

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

- Algorithm: forward pass computes \mathbf{f}_t , backward pass does $\mathbf{f}_i, \mathbf{b}_i$



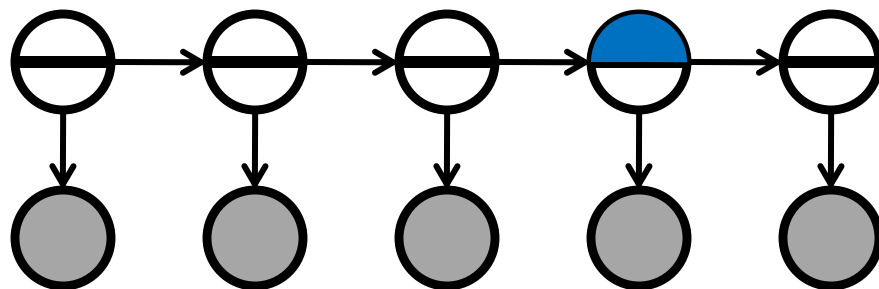
Country dance algorithm

- Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

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

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

- Algorithm: forward pass computes \mathbf{f}_t , backward pass does $\mathbf{f}_i, \mathbf{b}_i$



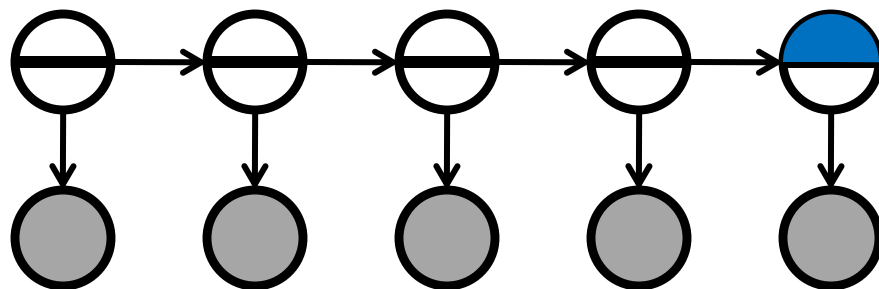
Country dance algorithm

- Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

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

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

- Algorithm: forward pass computes \mathbf{f}_t , backward pass does $\mathbf{f}_i, \mathbf{b}_i$



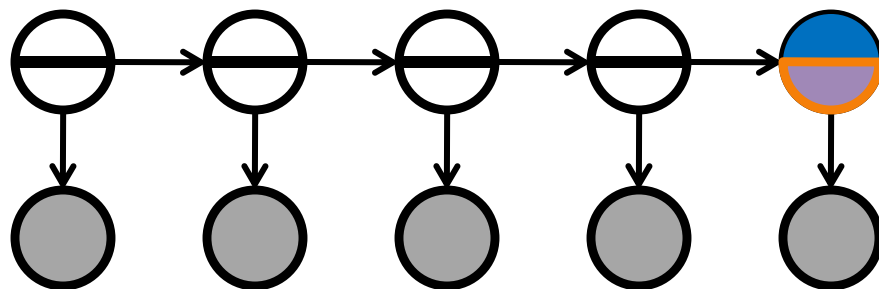
Country dance algorithm

- Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

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

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

- Algorithm: forward pass computes \mathbf{f}_t , backward pass does $\mathbf{f}_i, \mathbf{b}_i$



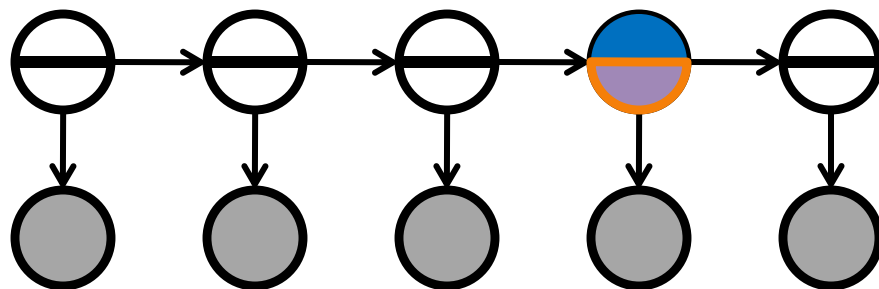
Country dance algorithm

- Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

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

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

- Algorithm: forward pass computes \mathbf{f}_t , backward pass does $\mathbf{f}_i, \mathbf{b}_i$



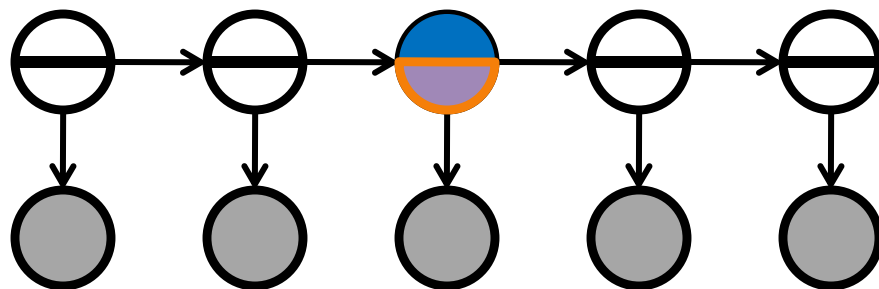
Country dance algorithm

- Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

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

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

- Algorithm: forward pass computes \mathbf{f}_t , backward pass does $\mathbf{f}_i, \mathbf{b}_i$



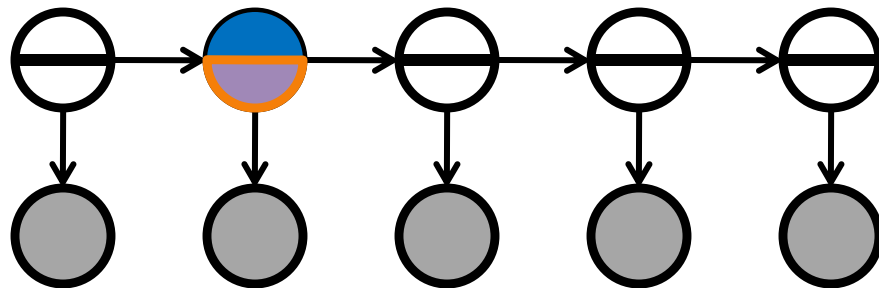
Country dance algorithm

- Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

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

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

- Algorithm: forward pass computes \mathbf{f}_t , backward pass does $\mathbf{f}_i, \mathbf{b}_i$



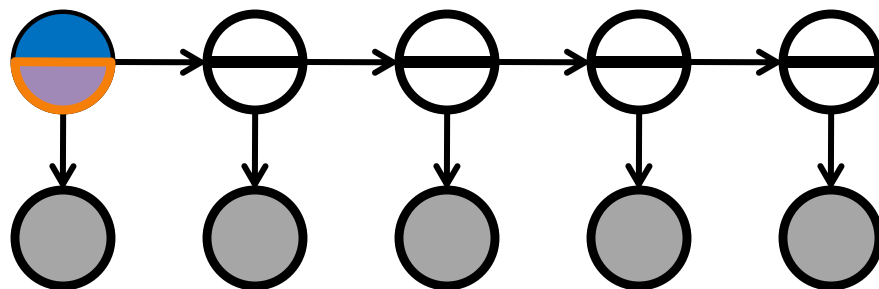
Country dance algorithm

- Can avoid storing all forward messages in smoothing by running forward algorithm backwards:

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

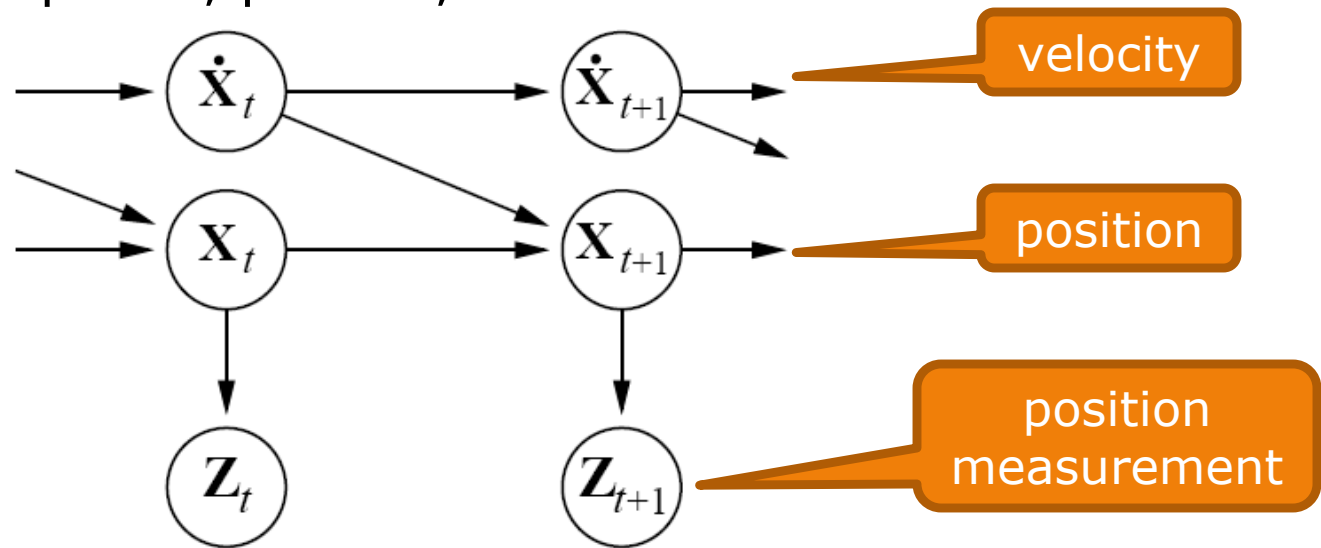
$$\mathbf{b}_{k+1:t} = \mathbf{T} \mathbf{O}_{k+1} \mathbf{b}_{k+2:t}$$

- Algorithm: forward pass computes \mathbf{f}_t , backward pass does $\mathbf{f}_i, \mathbf{b}_i$



Kalman filters

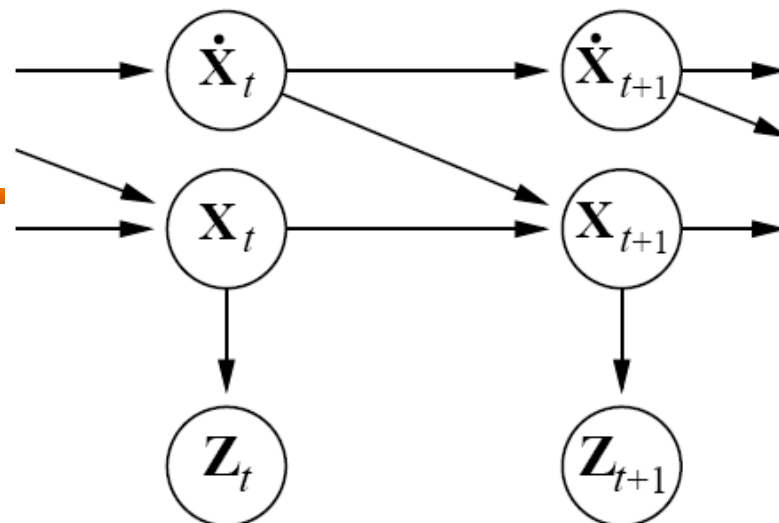
Modeling systems described by a set of continuous variables, e.g., tracking a bird flying – $X_t = X, Y, Z, \dot{X}, \dot{Y}, \dot{Z}$.
Airplanes, robots, ecosystems, economies, chemical plants, planets, ...



Kalman filters

- Use linear Gaussian distributions
- X-coordinate update, assuming constant velocity is:

$$X_{t+\Delta} = X_t + \dot{X}\Delta.$$



- Adding Gaussian noise, gives the following linear Gaussian transition model:

$$P(X_{t+\Delta} = x_{t+\Delta} \mid X_t = x_t, \dot{X}_t = \dot{x}_t) = N(x_t + \dot{x}_t\Delta, \sigma)(x_{t+\Delta}).$$

- Multivariate Gaussians are straightforward extensions of this.

Updating Gaussian distributions

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

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

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

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

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

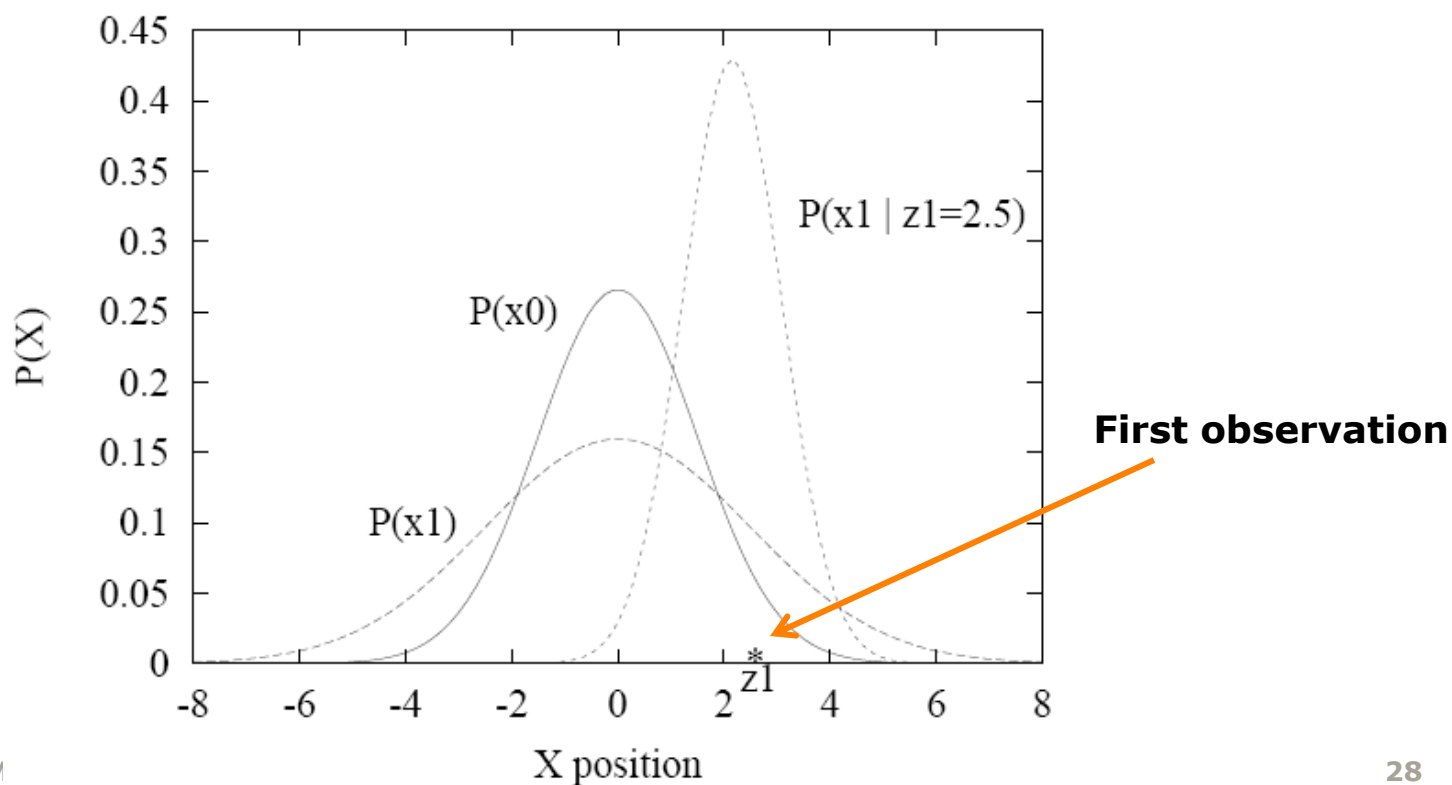
FORWARD operator for Kalman filters takes a Gaussian forward message $f_{1:t}$ specified by $N(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ and produces $N(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1})$

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:

$$\mu_{t+1} = \mathbf{F}\mu_t + \mathbf{K}_{t+1} (\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\mu_t)$$

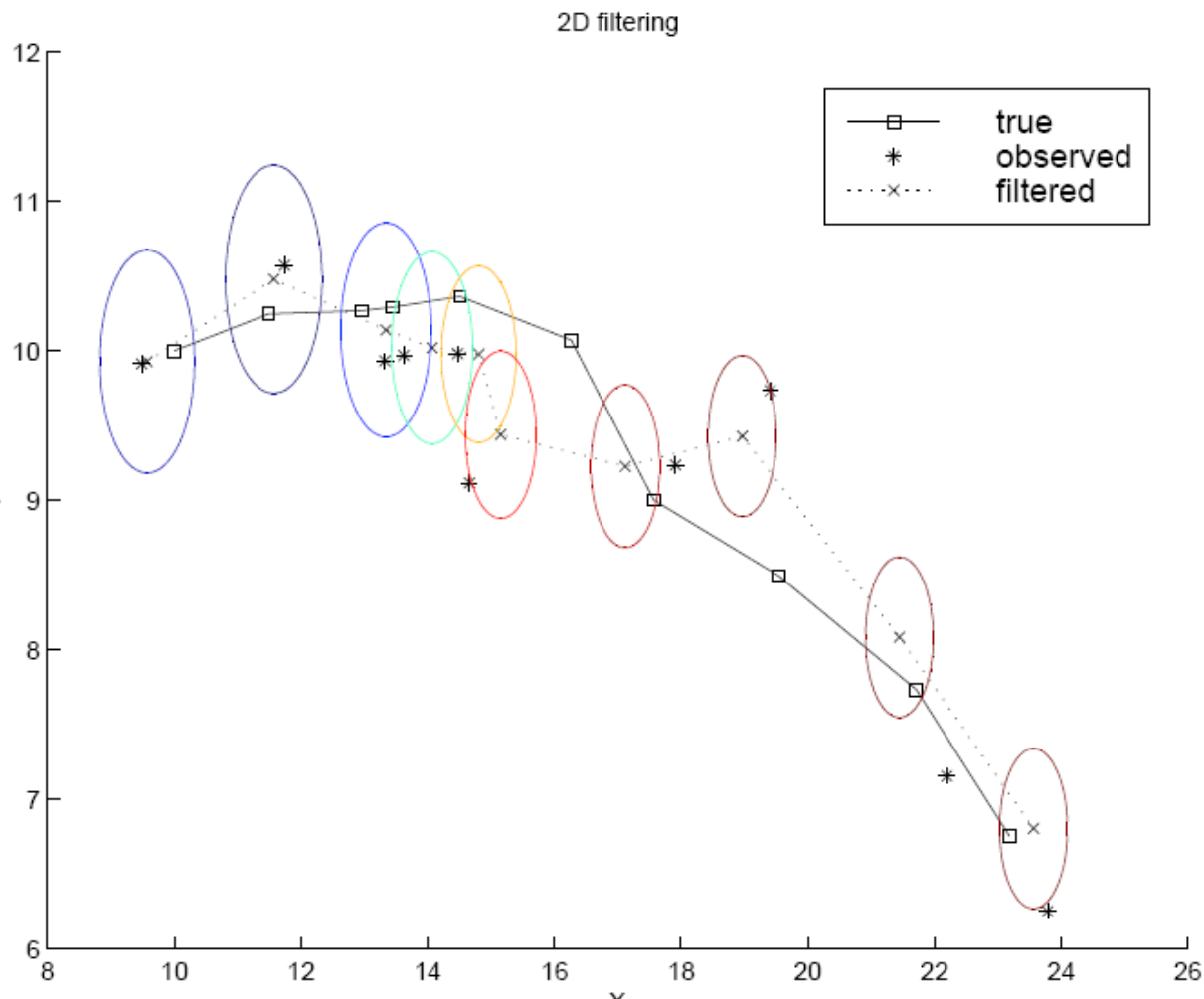
$$\Sigma_{t+1} = (\mathbf{I} - \mathbf{K}_{t+1}) (\mathbf{F}\Sigma_t\mathbf{F}^\top + \Sigma_x)$$

where $\mathbf{K}_{t+1} = (\mathbf{F}\Sigma_t\mathbf{F}^\top + \Sigma_x) \mathbf{H}^\top (\mathbf{H}(\mathbf{F}\Sigma_t\mathbf{F}^\top + \Sigma_x)\mathbf{H}^\top + \Sigma_z)^{-1}$

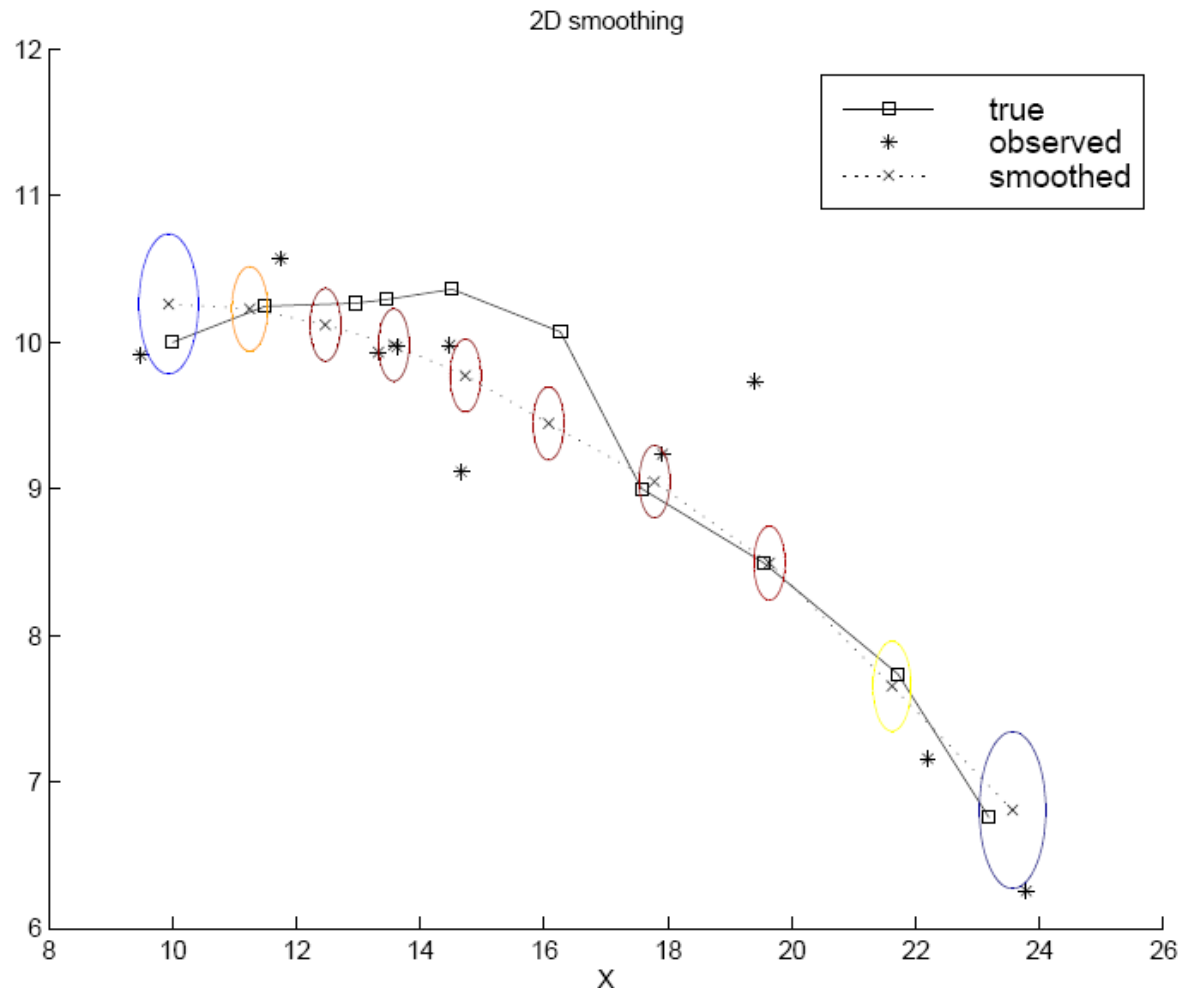
is the Kalman gain matrix

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

2-D tracking example: filtering



2-D tracking example: smoothing

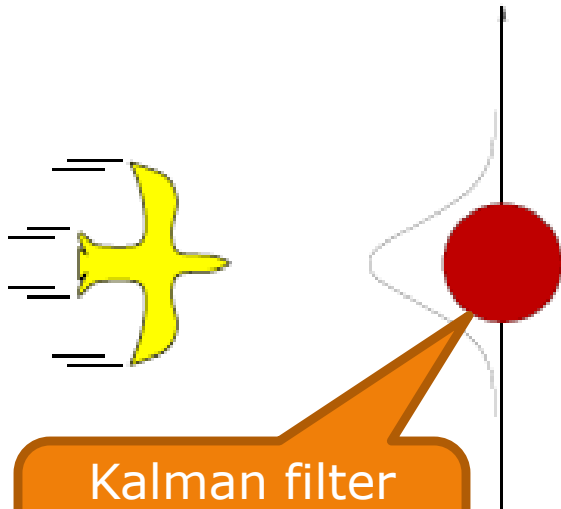


Where it breaks

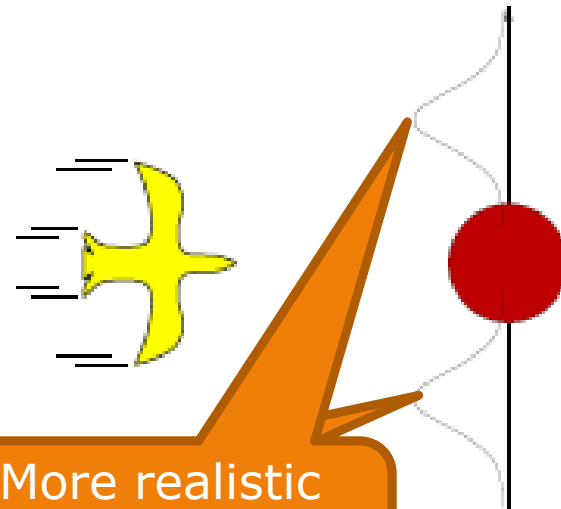
Cannot be applied if the transition model is nonlinear

Extended Kalman Filter models transition as **locally linear** around $x_t = \mu_t$

Fails if systems is locally unsmooth



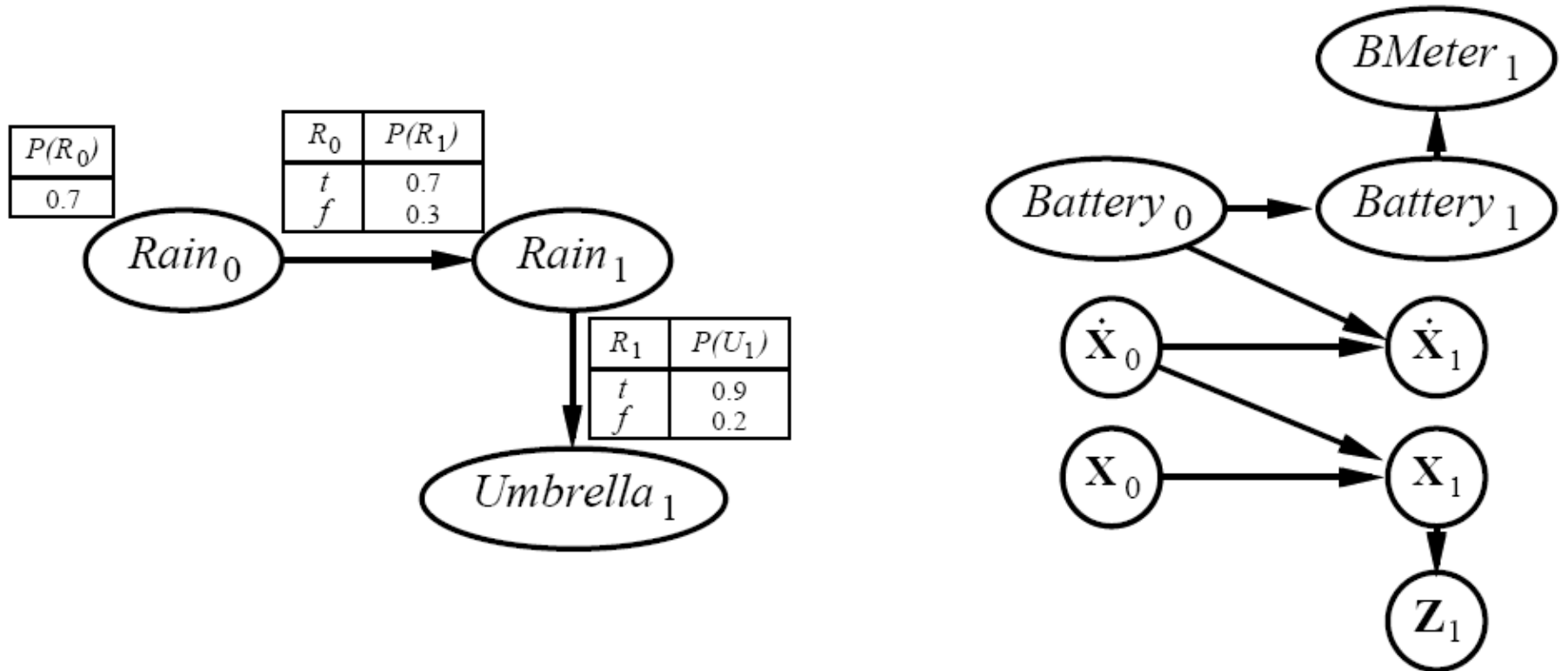
Kalman filter predicts the bird flies stright



More realistic model considers obstacle

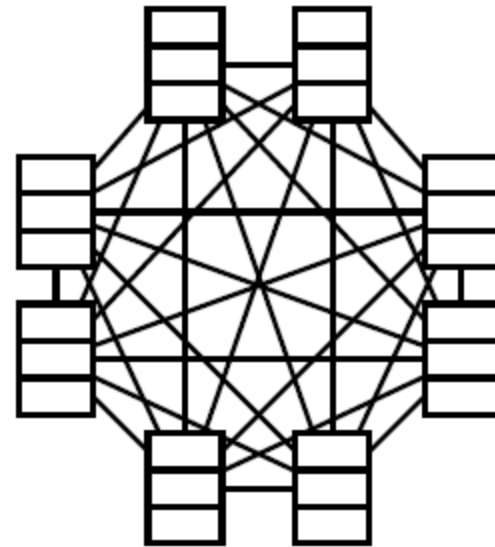
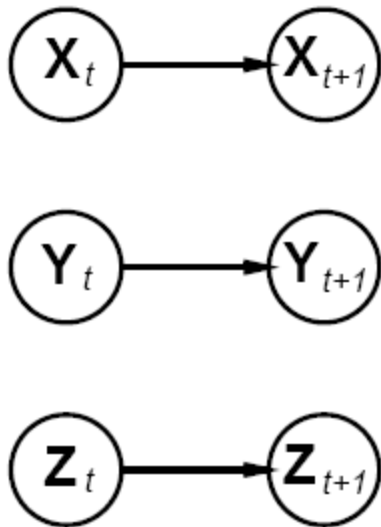
Dynamic Bayesian networks

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



DBNs vs. HMMs

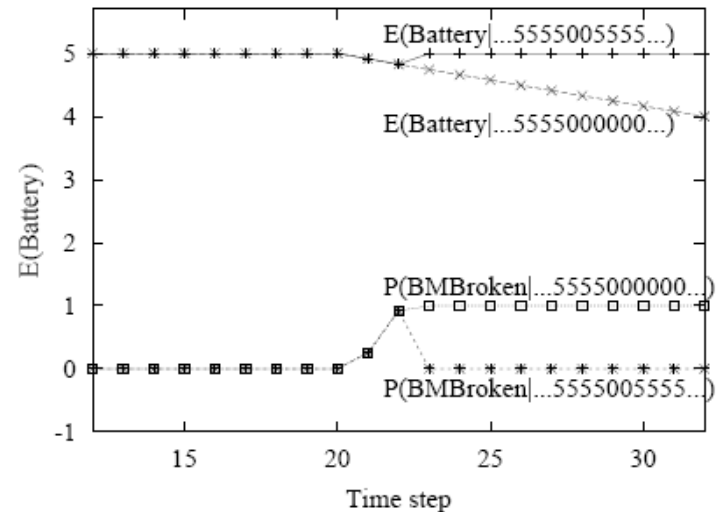
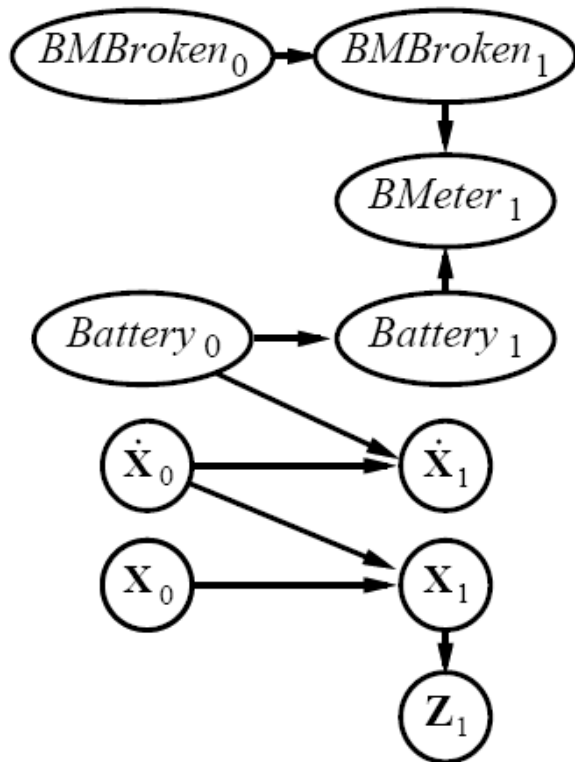
- Every HMM is a single-variable DBN; every discrete DBN is an HMM



- Sparse dependencies \Rightarrow exponentially fewer parameters; e.g., 20 state variables, three parents each DBN has $20 \times 2^3 = 160$ parameters, HMM has $2^{20} \times 2^{20} \approx 10^{12}$

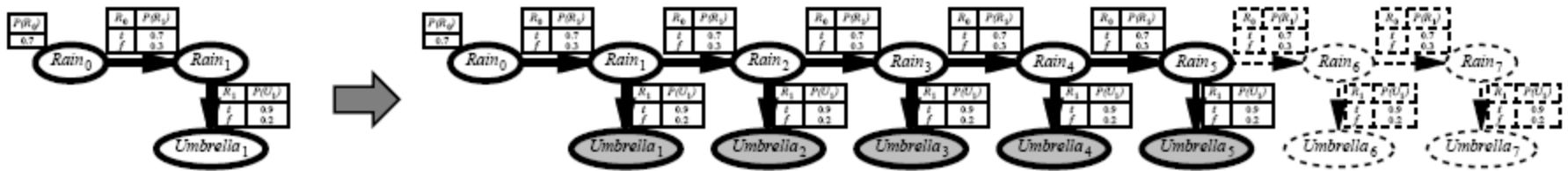
DBNs vs. Kalman filters

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

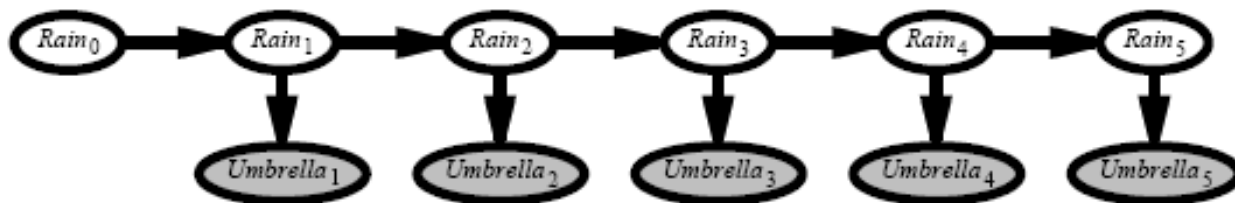
Rollup filtering: add slice $t + 1$, “sum out” slice t using variable elimination

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

(cf. HMM update cost $O(d^{2n})$)

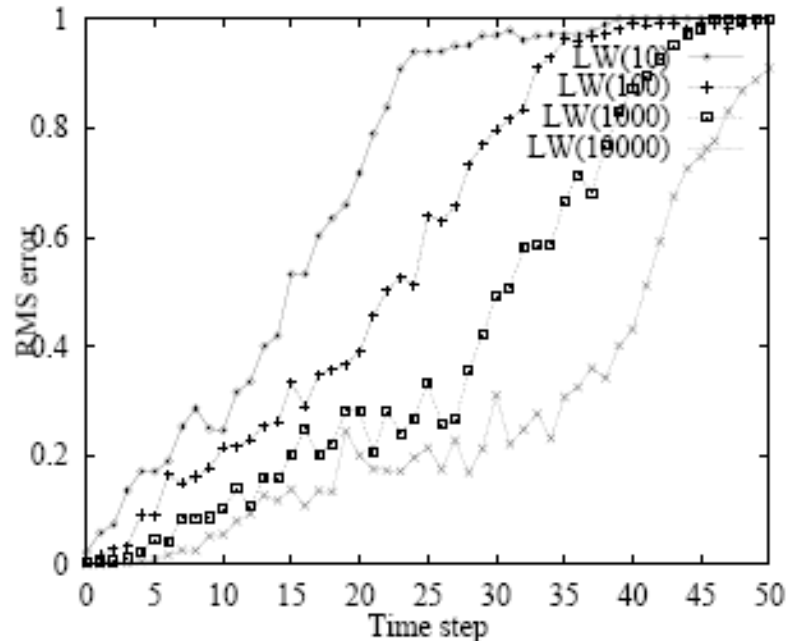
Likelihood weighting for DBNs

Set of weighted samples approximates the belief state



LW samples pay no attention to the evidence!

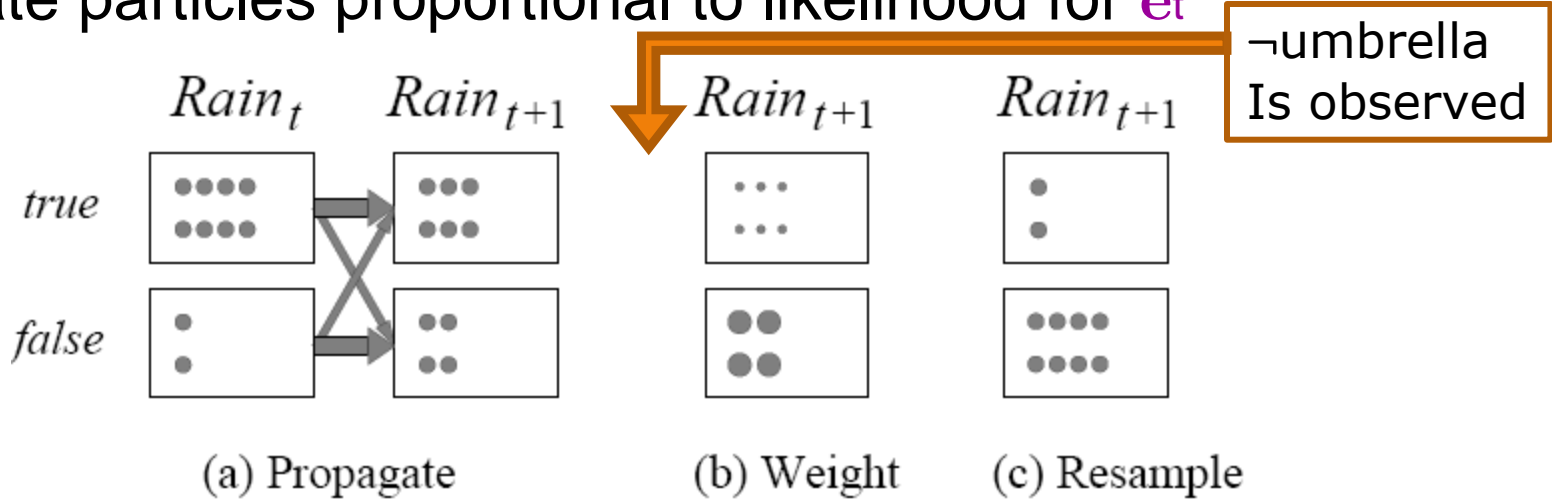
- ⇒ fraction “agreeing” falls exponentially with t
- ⇒ num. samples required grows exponentially with t



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 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

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

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

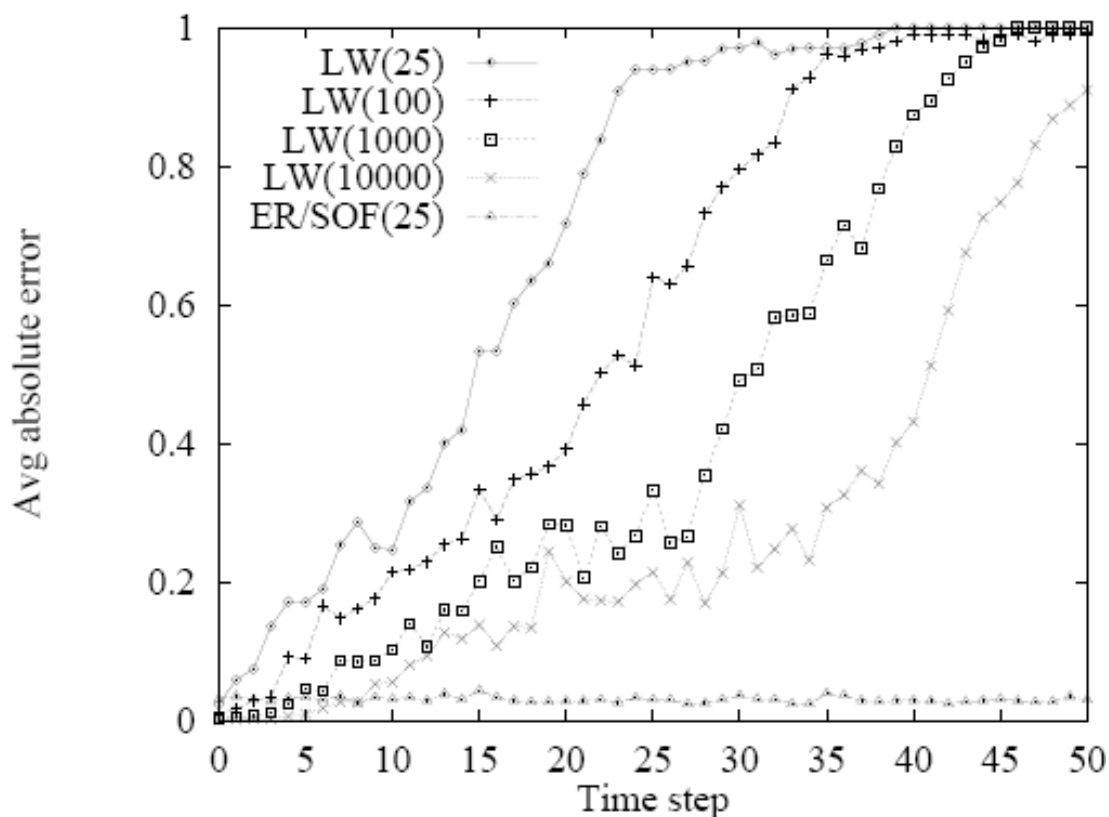
$$W(\mathbf{x}_{t+1} | \mathbf{e}_{1:t+1}) = P(\mathbf{e}_{t+1} | \mathbf{x}_{t+1}) N(\mathbf{x}_{t+1} | \mathbf{e}_{1:t})$$

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, at least empirically—theoretical analysis is difficult



Summary

- Temporal models use state & sensor variables replicated over time
- Markov assumptions and stationarity assumption, so we need
 - transition model $P(\mathbf{X}_t | \mathbf{X}_{t-1})$
 - sensor model $P(\mathbf{E}_t | \mathbf{X}_t)$
- 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
- 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