PROBABILISTIC REASONING OVER TIME

In which we try to interpret the present, understand the past, and perhaps predict the future, even when very little is crystal clear.

Outline

◇ 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

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

Example and notation

Time slices containing a set of random variables, some observable and some not.

X_t: denote the set of state variables at time t, unobservable
E_t: denote the set of observable evidence variables.

Example: Security guard with umbrella. For each day t, the set E_t contains a single evidence variable U_t (umbrella) and the set X_t contains a single state variable R_t (rain).

The interval between time slices fixed.

Evidence starts arriving at $t = 1$ Hence, our umbrella world is represented by state variables $R_0, R_1, R_2, \ldots$ and evidence variables $U_1, U_2, \ldots$.

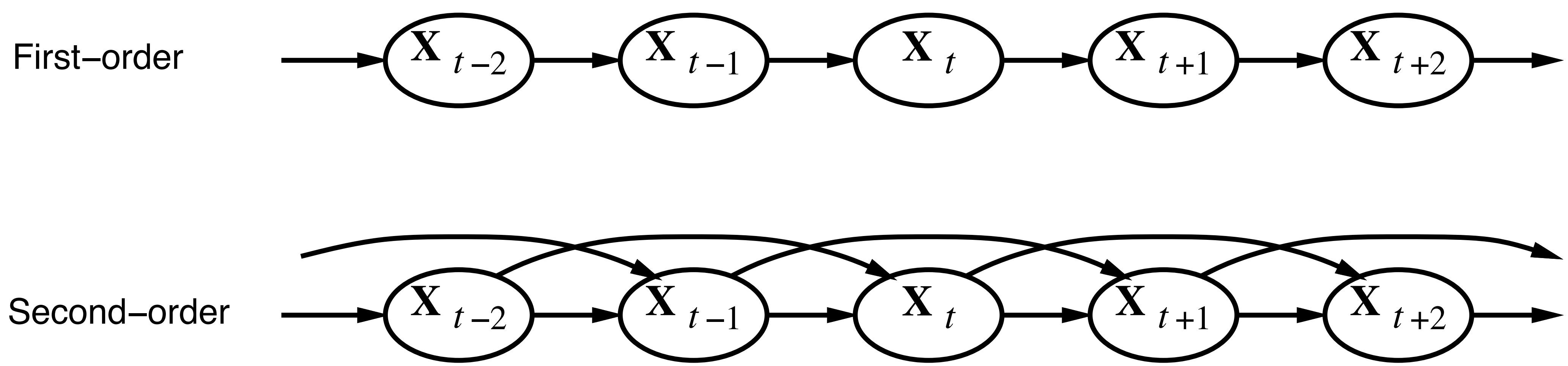
$a : b$ denotes the sequence of integers from a to b (inclusive), and $X_{a:b}$ denotes the set of variables from X_a to X_b.

Markov processes (Markov chains)

Construct a Bayes net from these variables: parents?

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

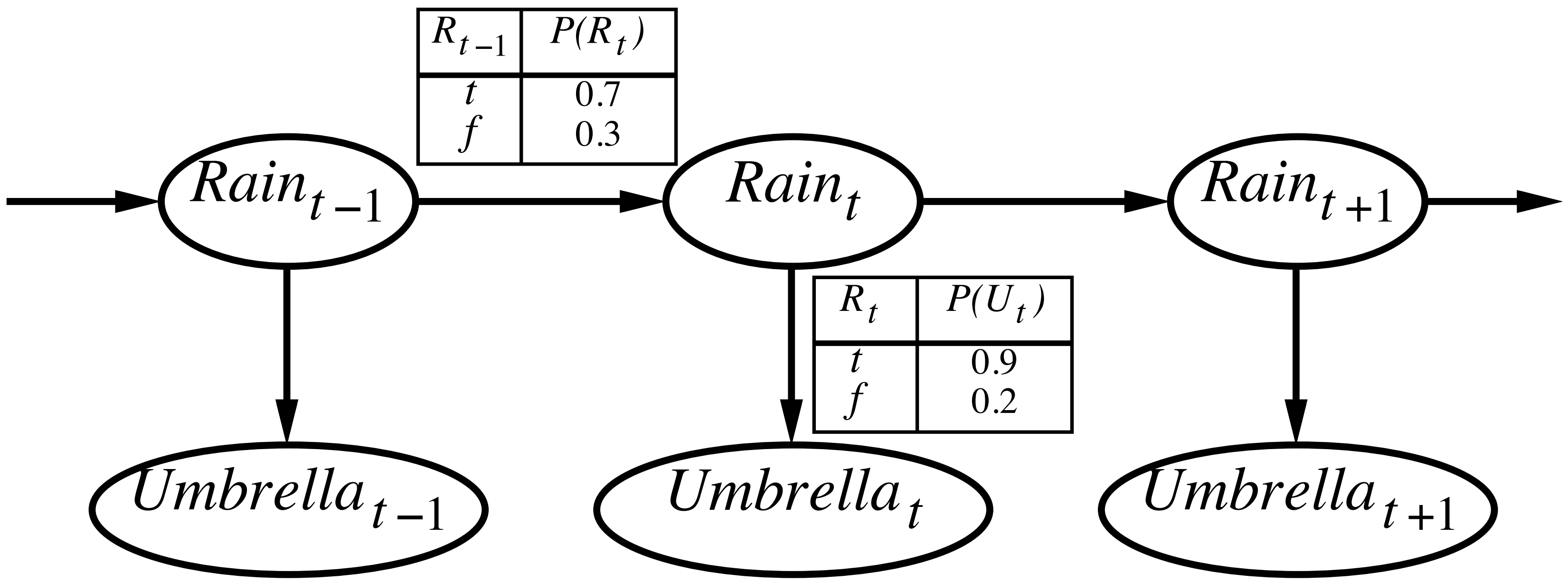
First-order Markov process: $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-1})$
Second-order Markov process: $\mathbf{P}(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \mathbf{P}(\mathbf{X}_t|\mathbf{X}_{t-2}, \mathbf{X}_{t-1})$



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 $Battery_t$

Get started

How to get started?

The prior probability distribution at time 0, $\mathbf{P}(\mathbf{X}_0)$

With that, we have a specification of the complete joint distribution over all the variables. For any t,

$$\mathbf{P}(\mathbf{X}_{0:t}, \mathbf{E}_{1:t}) = \mathbf{P}(\mathbf{X}_0) \prod_{i=1}^{t} \mathbf{P}(\mathbf{X}_i | \mathbf{X}_{i-1}) \mathbf{P}(\mathbf{E}_i | \mathbf{X}_i) \ .$$

with

Initial state model: $\mathbf{P}(\mathbf{X}_0)$
Transition model: $\mathbf{P}(\mathbf{X}_i | \mathbf{X}_{i-1})$
Sensor model: $\mathbf{P}(\mathbf{E}_i | \mathbf{X}_\mathbf{i})$.

Get started

How to get started?

The prior probability distribution at time 0, $\mathbf{P}(\mathbf{X}_0)$

With that, we have a specification of the complete joint distribution over all the variables. For any t,

$$\mathbf{P}(\mathbf{X}_{0:t}, \mathbf{E}_{1:t}) = \mathbf{P}(\mathbf{X}_0) \prod_{i=1}^{t} \mathbf{P}(\mathbf{X}_i | \mathbf{X}_{i-1}) \mathbf{P}(\mathbf{E}_i | \mathbf{X}_i) \; .$$

with

Initial state model: $\mathbf{P}(\mathbf{X}_0)$
Transition model: $\mathbf{P}(\mathbf{X}_i | \mathbf{X}_{i-1})$
Sensor model: $\mathbf{P}(\mathbf{E}_i | \mathbf{X_i})$.

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:

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

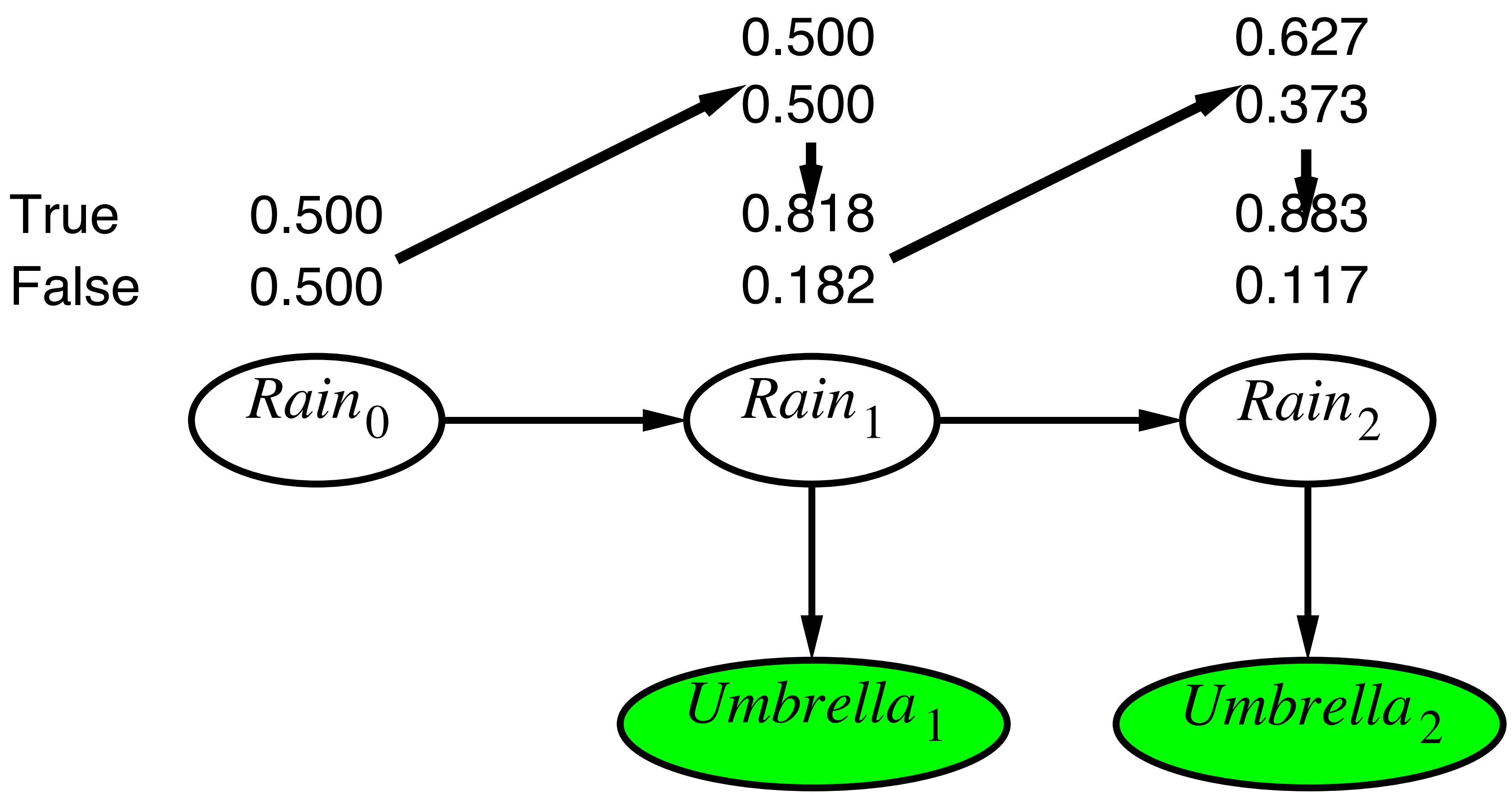
$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) &= \mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}, \mathbf{e}_{t+1})\\ &= \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}, \mathbf{e}_{1:t})\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})\\ &= \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})\end{aligne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

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

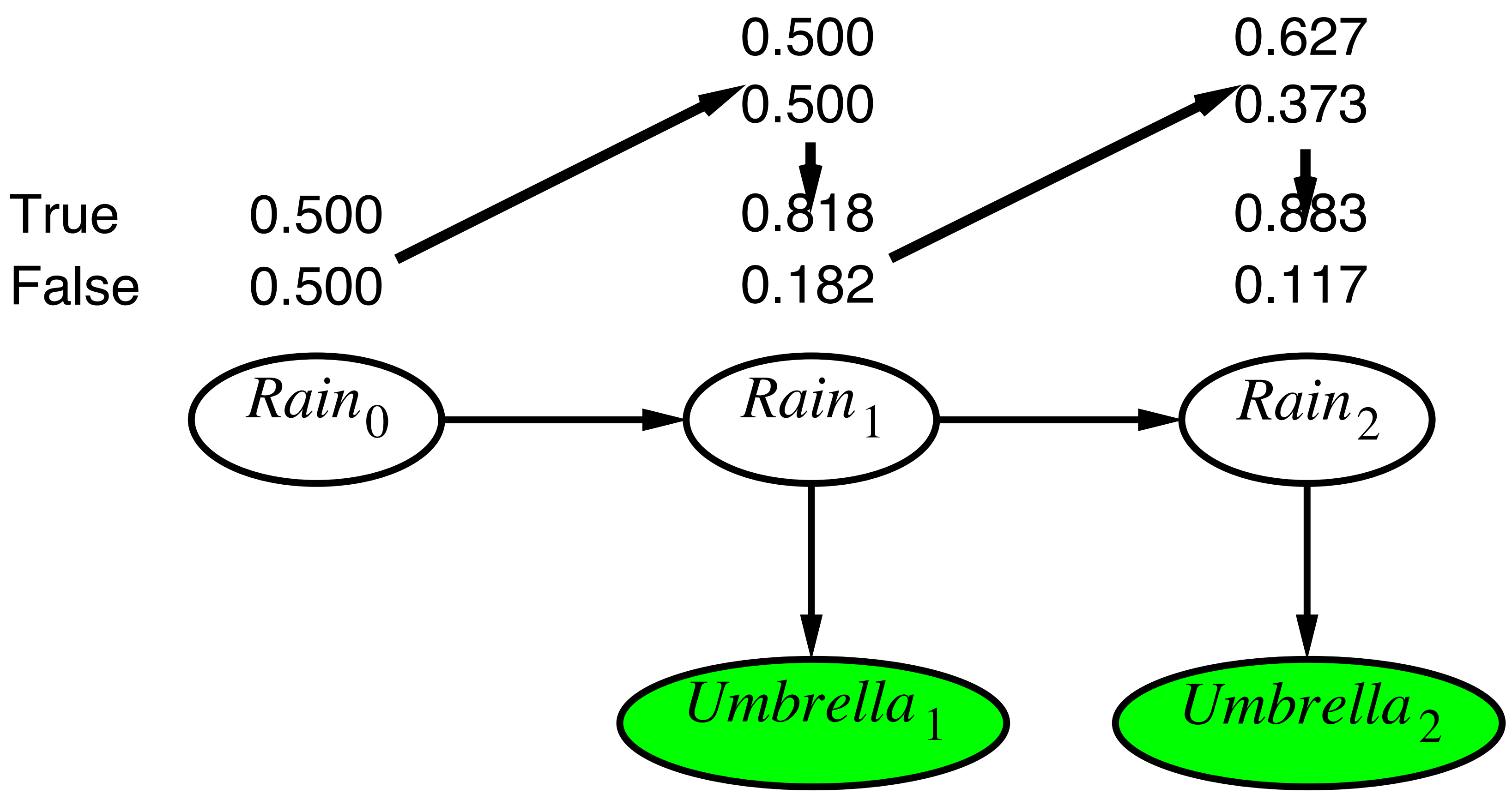
$$\begin{aligned}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) &= \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})\Sigma_{\mathbf{x}_t}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t, \mathbf{e}_{1:t})P(\mathbf{x}_t|\mathbf{e}_{1:t})\\ &= \alpha\mathbf{P}(\mathbf{e}_{t+1}|\mathbf{X}_{t+1})\Sigma_{\mathbf{x}_t}\mathbf{P}(\mathbf{X}_{t+1}|\mathbf{x}_t)P(\mathbf{x}_t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

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

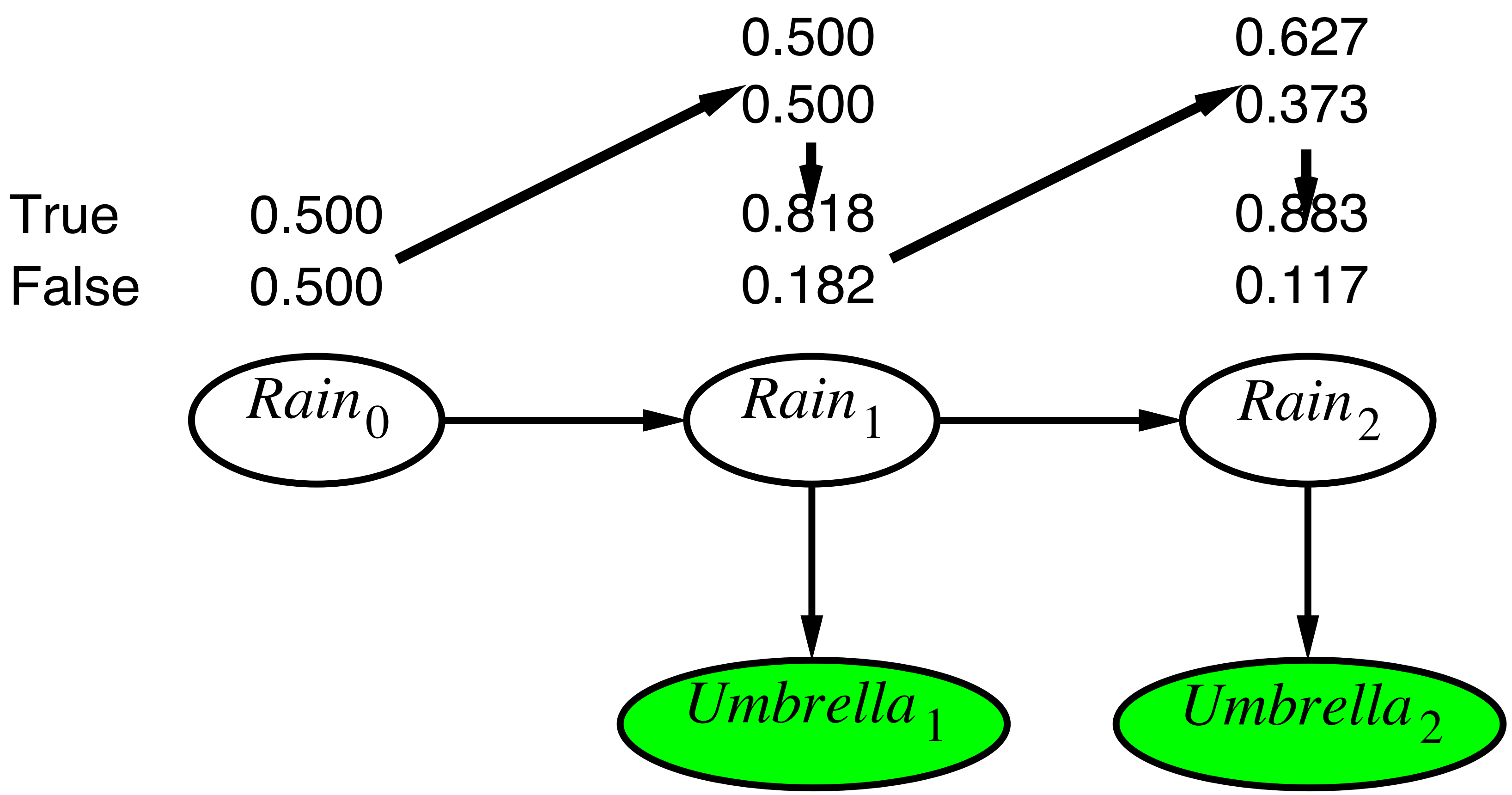
Filtering example



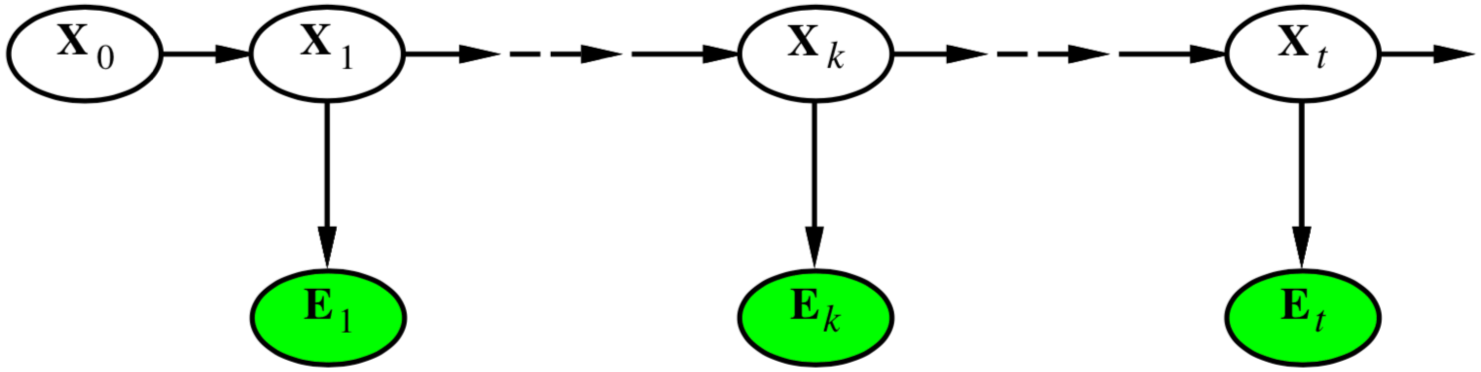
Filtering example



Filtering example



Smoothing



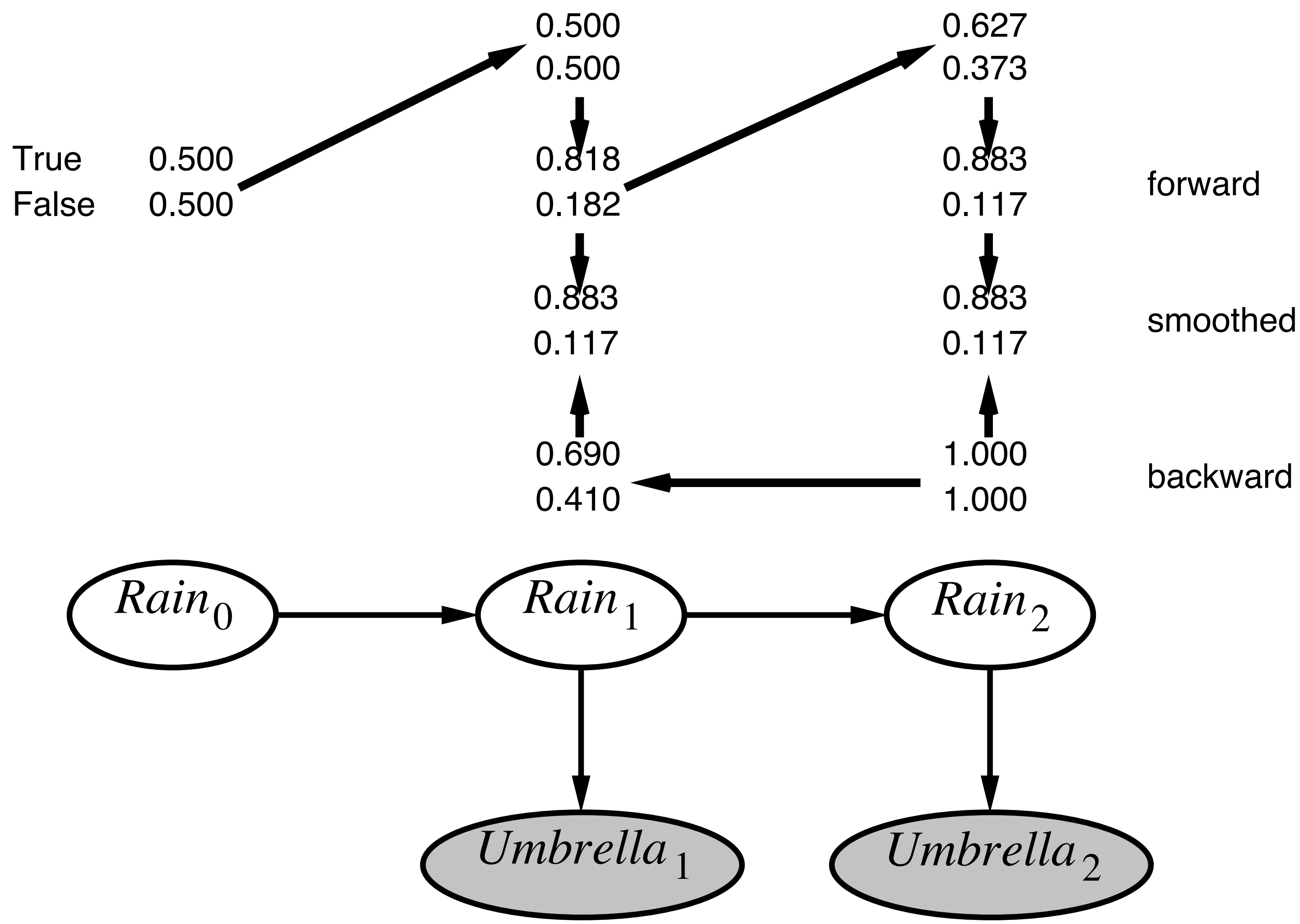
Divide evidence $\mathbf{e}_{1:t}$ into $\mathbf{e}_{1:k}$, $\mathbf{e}_{k+1:t}$:

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

Backward message computed by a backwards recursion:

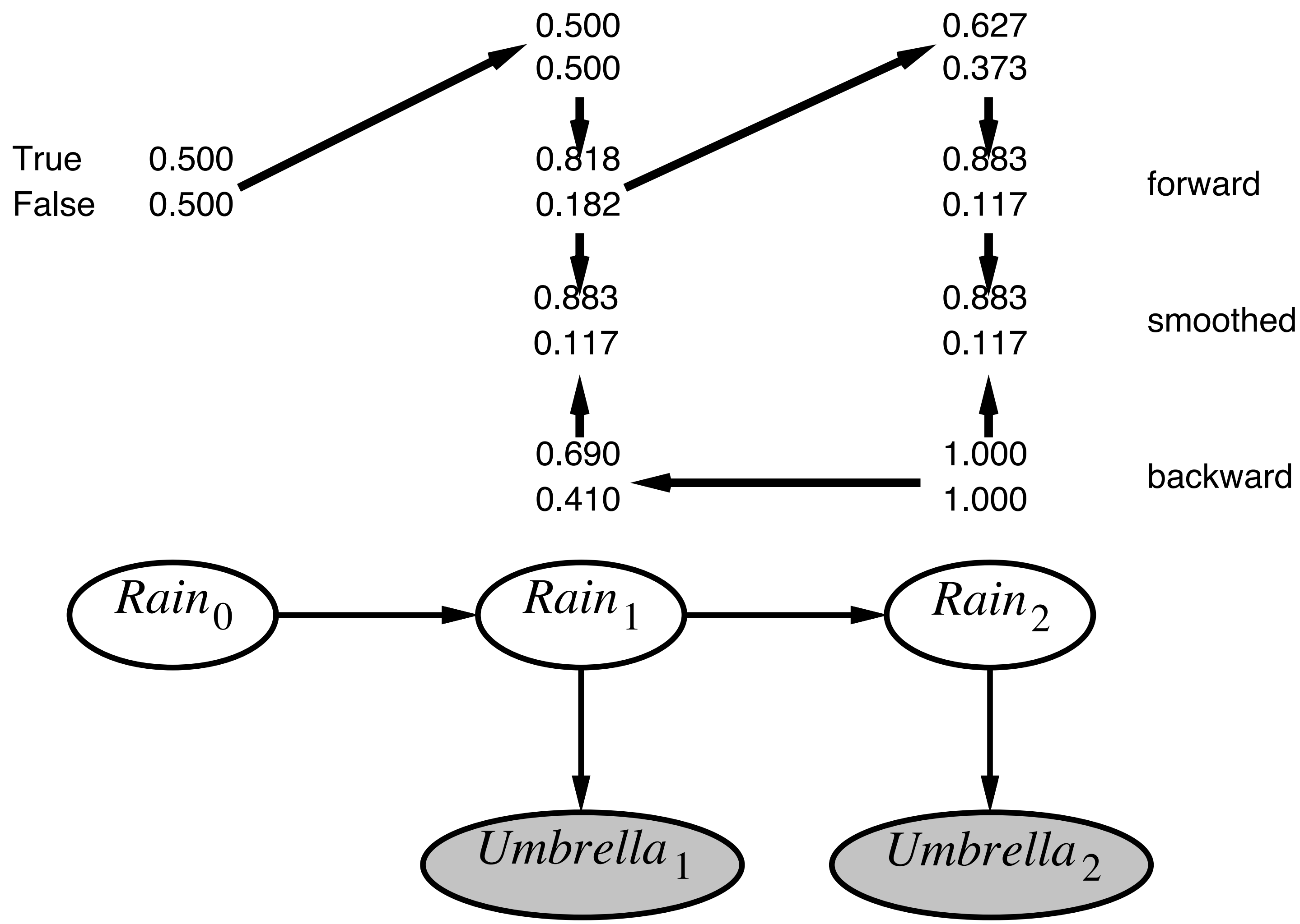
$$\begin{aligned}
\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k) &= \Sigma_{\mathbf{x}_{k+1}}\mathbf{P}(\mathbf{e}_{k+1:t}|\mathbf{X}_k, \mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k) \\
&= \Sigma_{\mathbf{x}_{k+1}}P(\mathbf{e}_{k+1:t}|\mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1}|\mathbf{X}_k)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

Smoothing example



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

Smoothing example



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

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

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

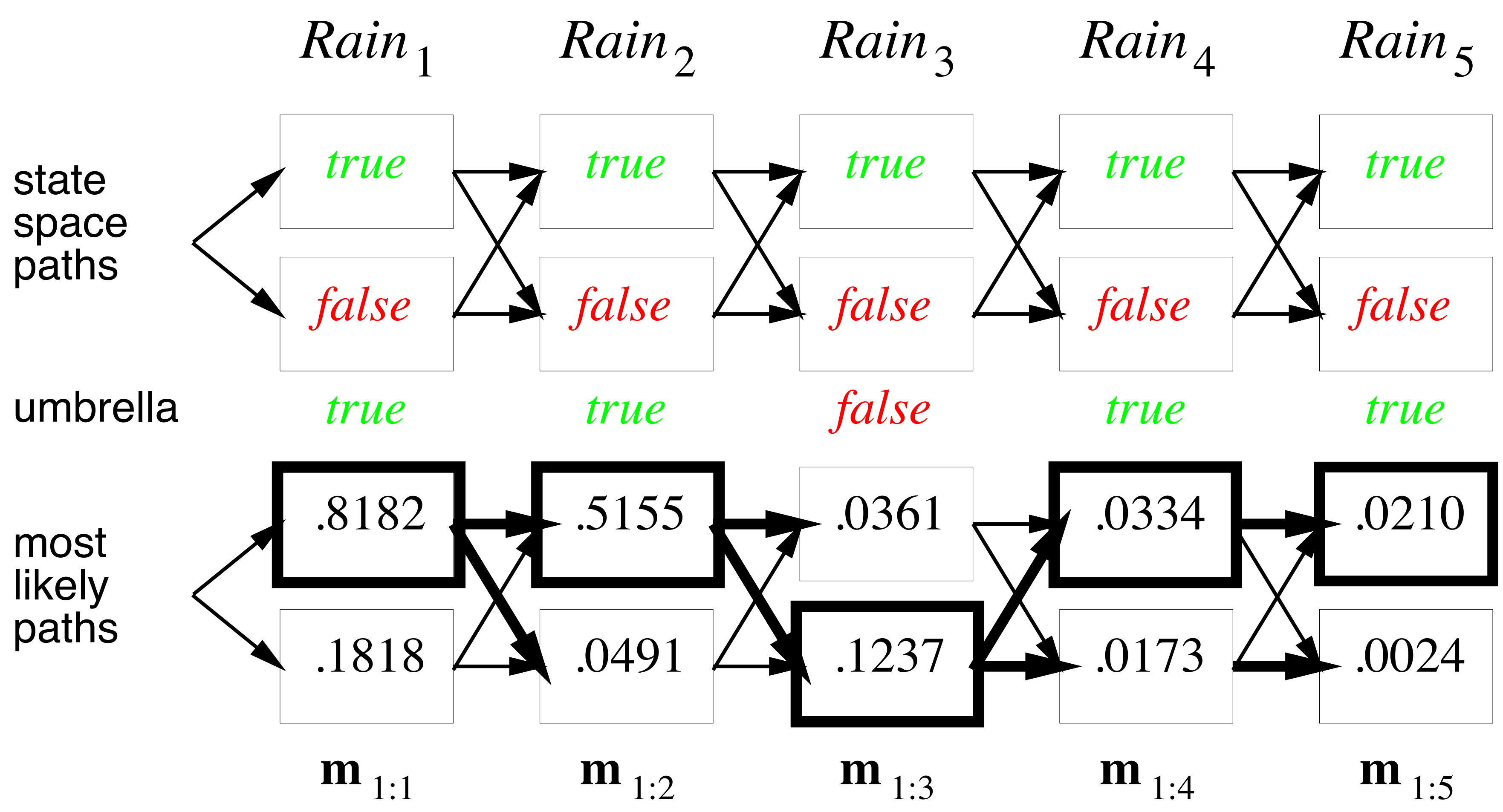
Identical to filtering, except $\mathbf{f}_{1:t}$ replaced by

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

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



function FORWARD-BACKWARD(**ev**, *prior*) **returns** a vector of probability distributions
  inputs: **ev**, a vector of evidence values for steps $1, \ldots, t$
        prior, the prior distribution on the initial state, $\mathbf{P}(\mathbf{X}_0)$
  local variables: **fv**, a vector of forward messages for steps $0, \ldots, t$
        b, a representation of the backward message, initially all 1s
        sv, a vector of smoothed estimates for steps $1, \ldots, t$

  $\mathbf{fv}[0] \leftarrow prior$
  for $i = 1$ **to** t **do**
    $\mathbf{fv}[i] \leftarrow$ FORWARD($\mathbf{fv}[i-1], \mathbf{ev}[i]$)
  for $i = t$ **downto** 1 **do**
    $\mathbf{sv}[i] \leftarrow$ NORMALIZE($\mathbf{fv}[i] \times \mathbf{b}$)
    $\mathbf{b} \leftarrow$ BACKWARD($\mathbf{b}, \mathbf{ev}[i]$)
  return sv

Figure 15.4 The forward–backward algorithm for smoothing: computing posterior probabilities of a sequence of states given a sequence of observations. The FORWARD and BACKWARD operators are defined by Equations (15.5) and (15.9), respectively.

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 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 $O(St)$

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 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 $O(St)$

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 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 $O(St)$

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 $\mathbf{P}(e_{t-d}|X_{t-d})$
    $\mathbf{B} \leftarrow \mathbf{O}_{t-d}^{-1}\mathbf{T}^{-1}\mathbf{B}\mathbf{T}\mathbf{O}_t$
  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

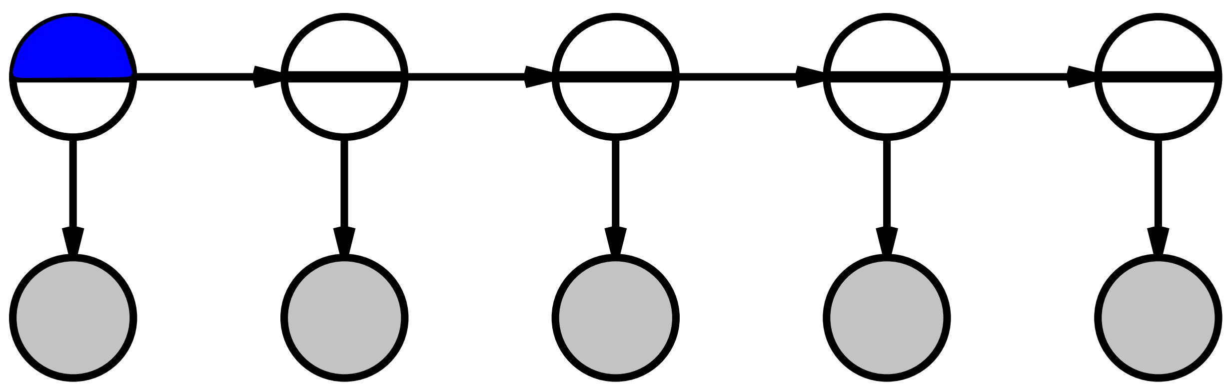
Figure 15.6 An algorithm for smoothing with a fixed time lag of d steps, implemented as an online algorithm that outputs the new smoothed estimate given the observation for a new time step. Notice that the final output NORMALIZE($\mathbf{f} \times \mathbf{B1}$) is just $\alpha\,\mathbf{f} \times \mathbf{b}$,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

Country dance algorithm

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

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

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

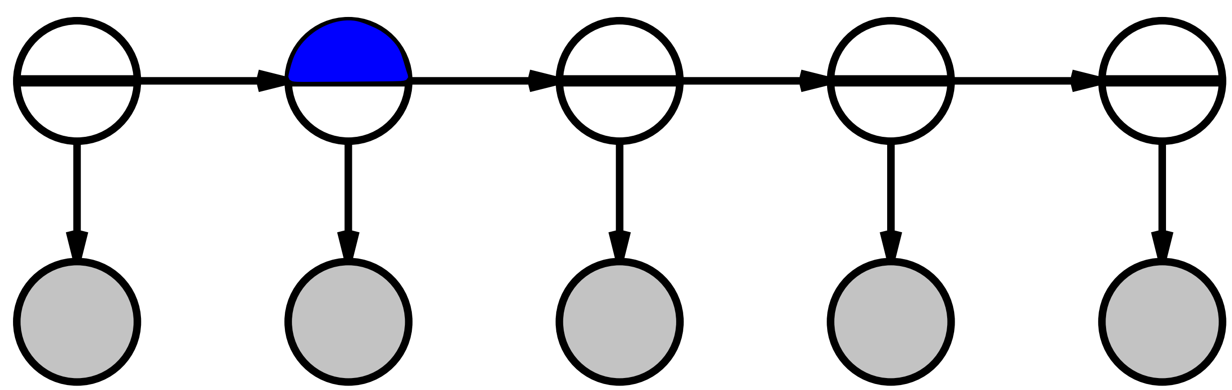


Country dance algorithm

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

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

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

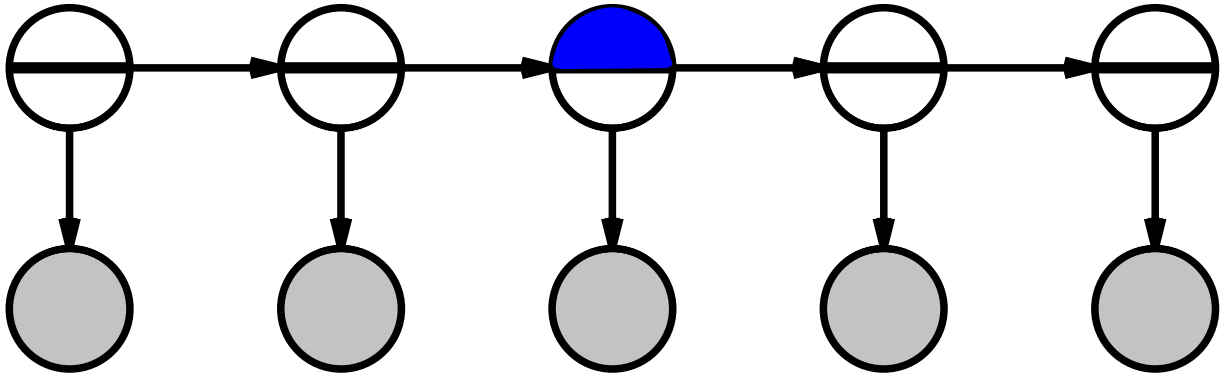


Country dance algorithm

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

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

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

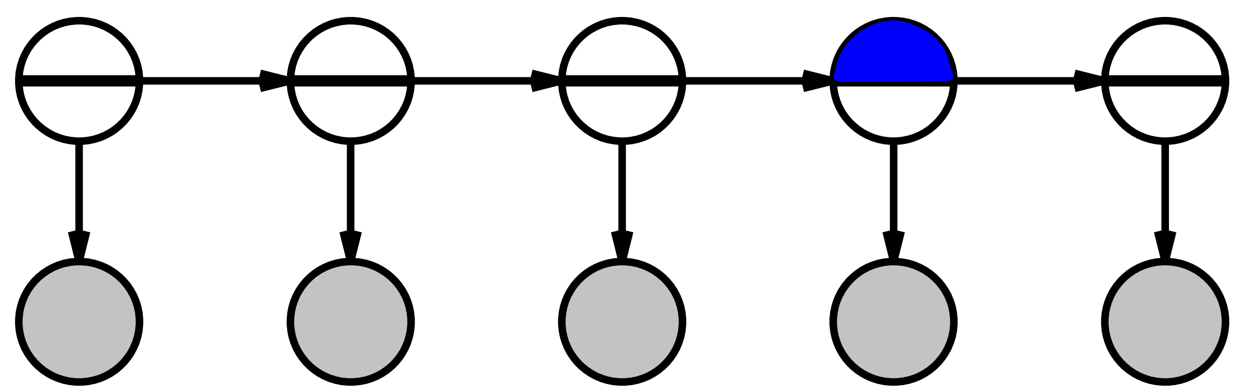


Country dance algorithm

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

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

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

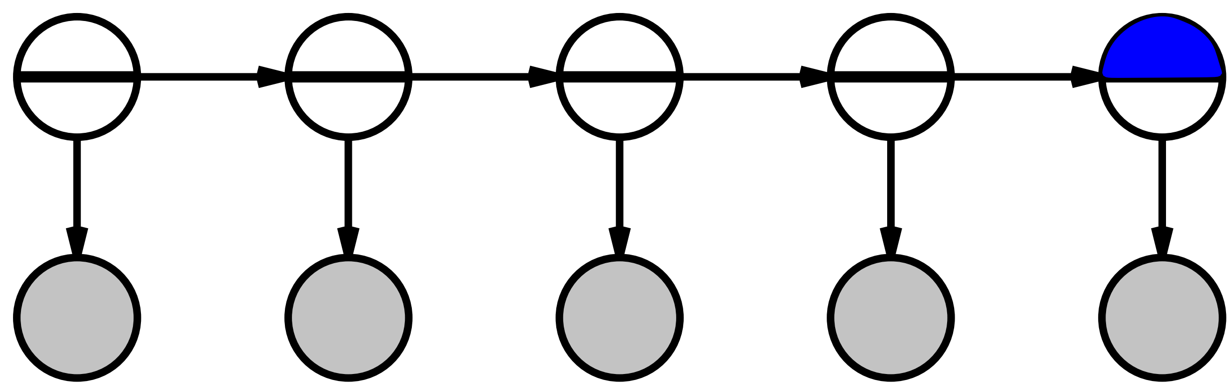


Country dance algorithm

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

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

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

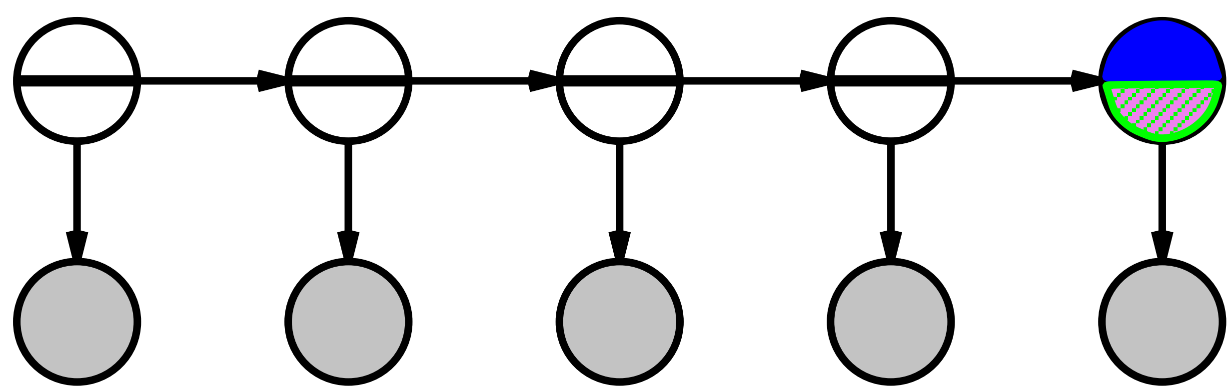


Country dance algorithm

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

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

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

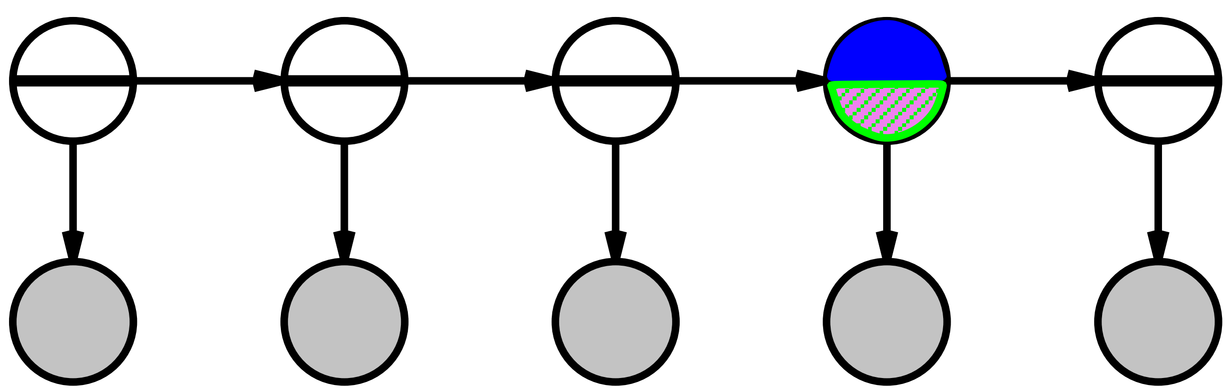


Country dance algorithm

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

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

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

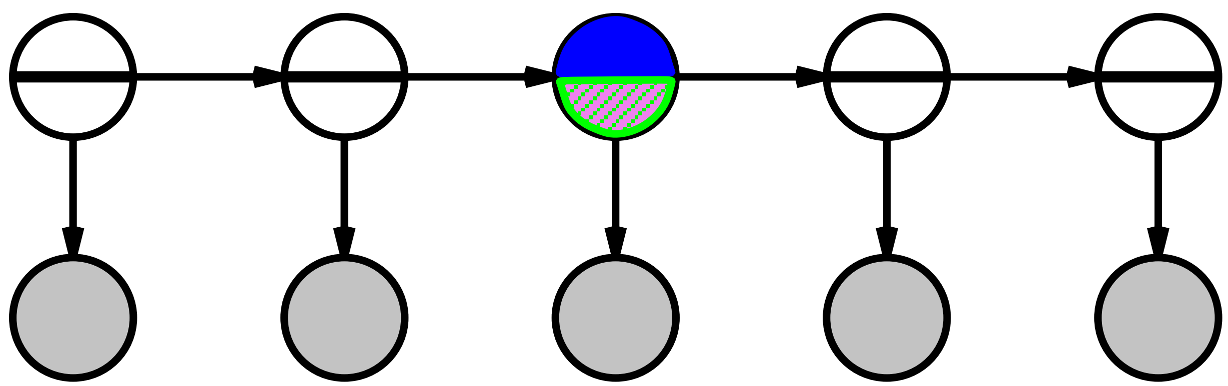


Country dance algorithm

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

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

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

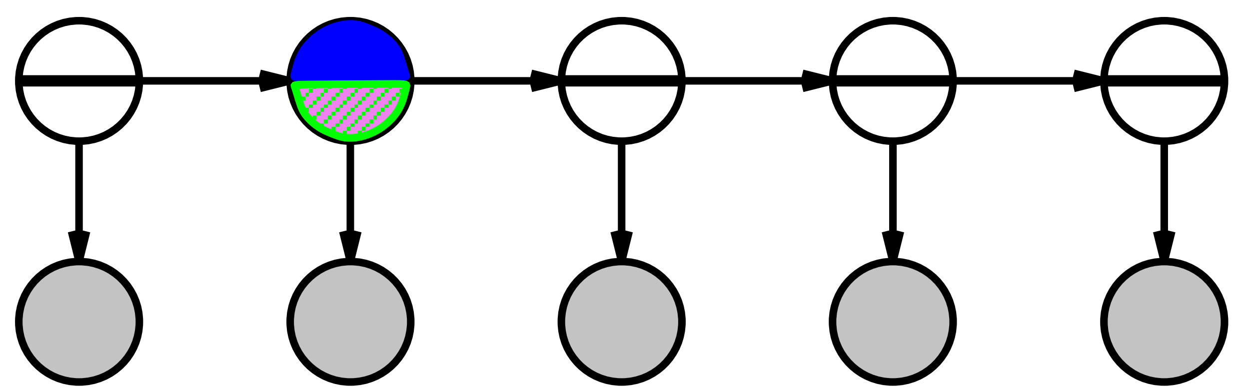


Country dance algorithm

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

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

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

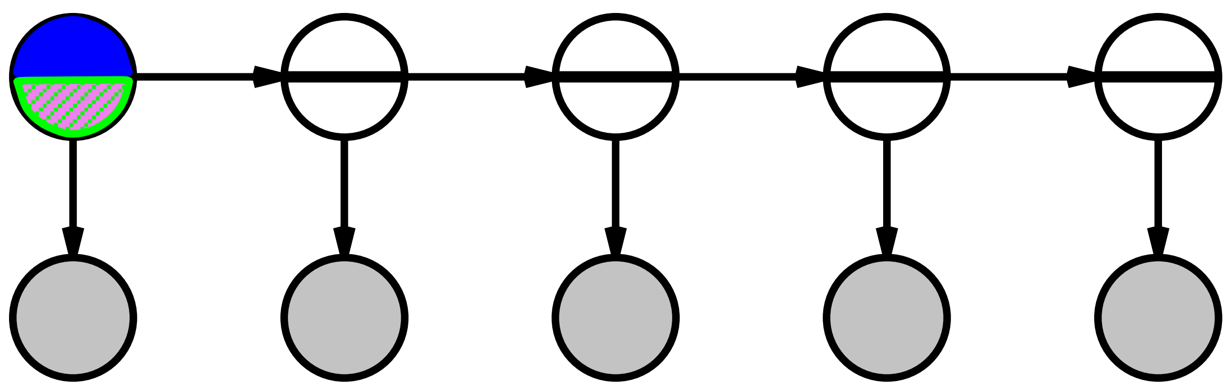


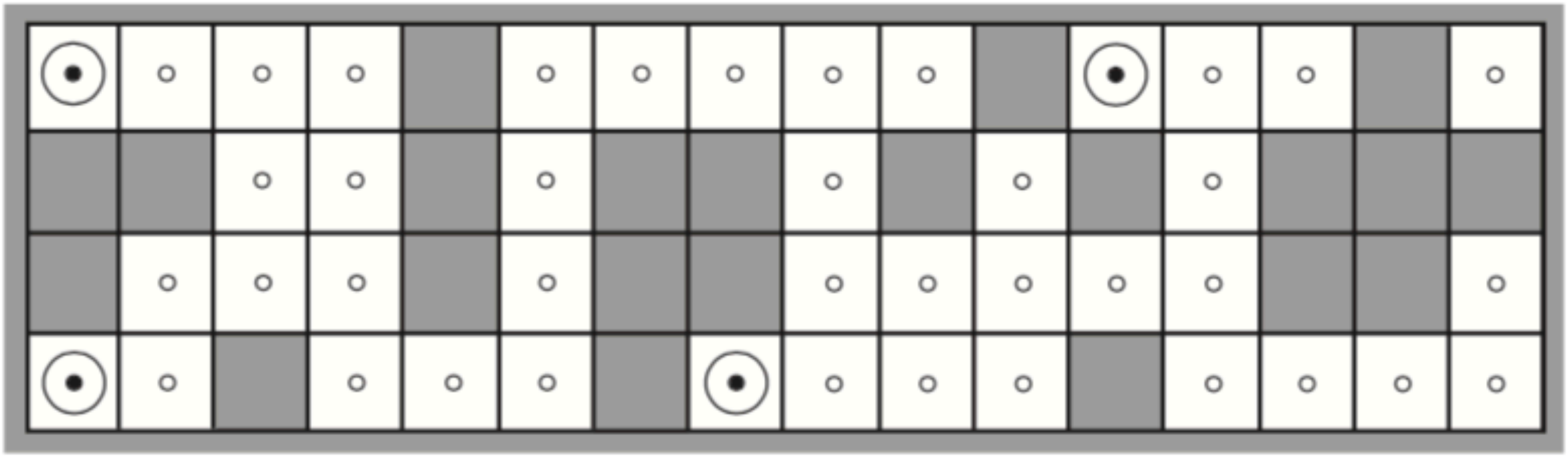
Country dance algorithm

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

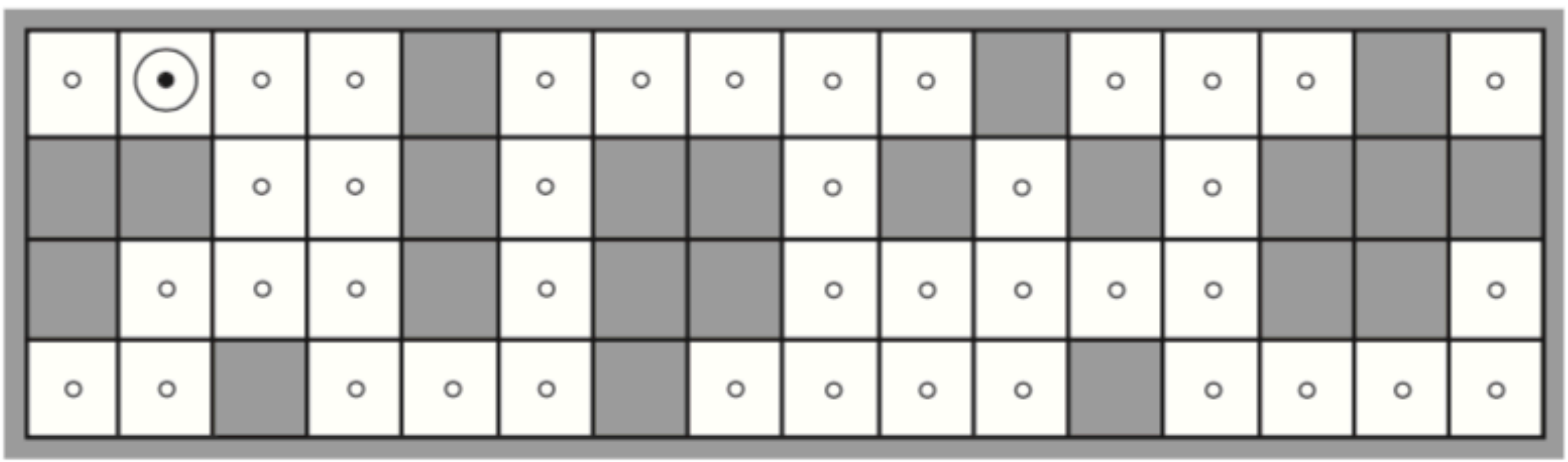
$$\begin{aligned}
\mathbf{f}_{1:t+1} &= \alpha \mathbf{O}_{t+1} \mathbf{T}^{\top} \mathbf{f}_{1:t} \\
\mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \alpha \mathbf{T}^{\top} \mathbf{f}_{1:t} \\
\alpha' (\mathbf{T}^{\top})^{-1} \mathbf{O}_{t+1}^{-1} \mathbf{f}_{1:t+1} &= \mathbf{f}_{1:t}
\end{aligned}$$

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

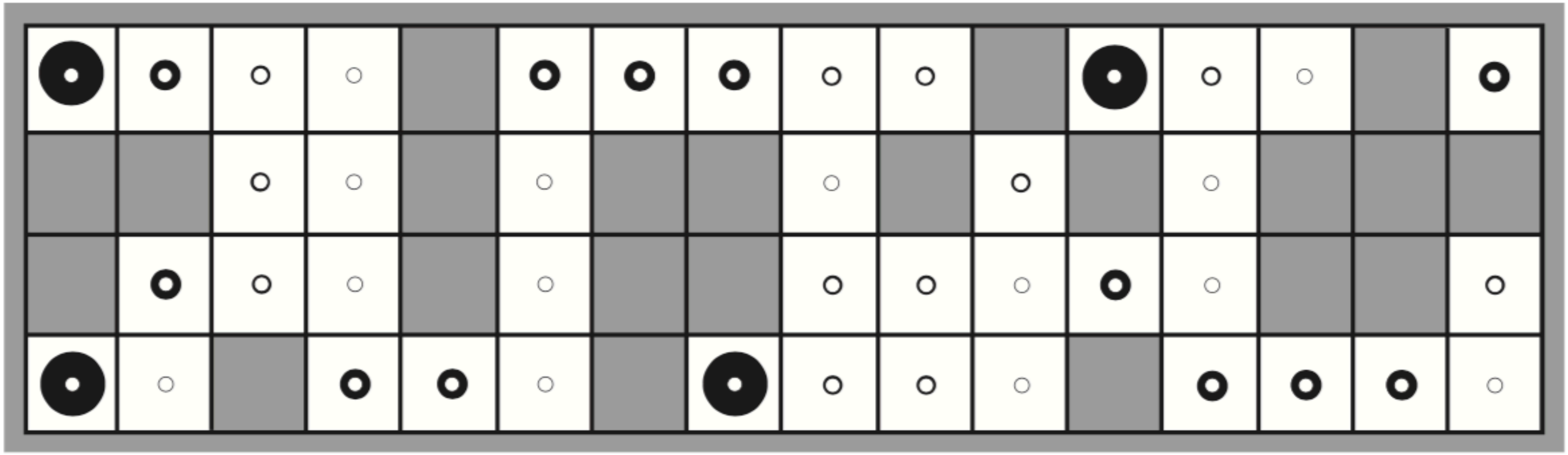




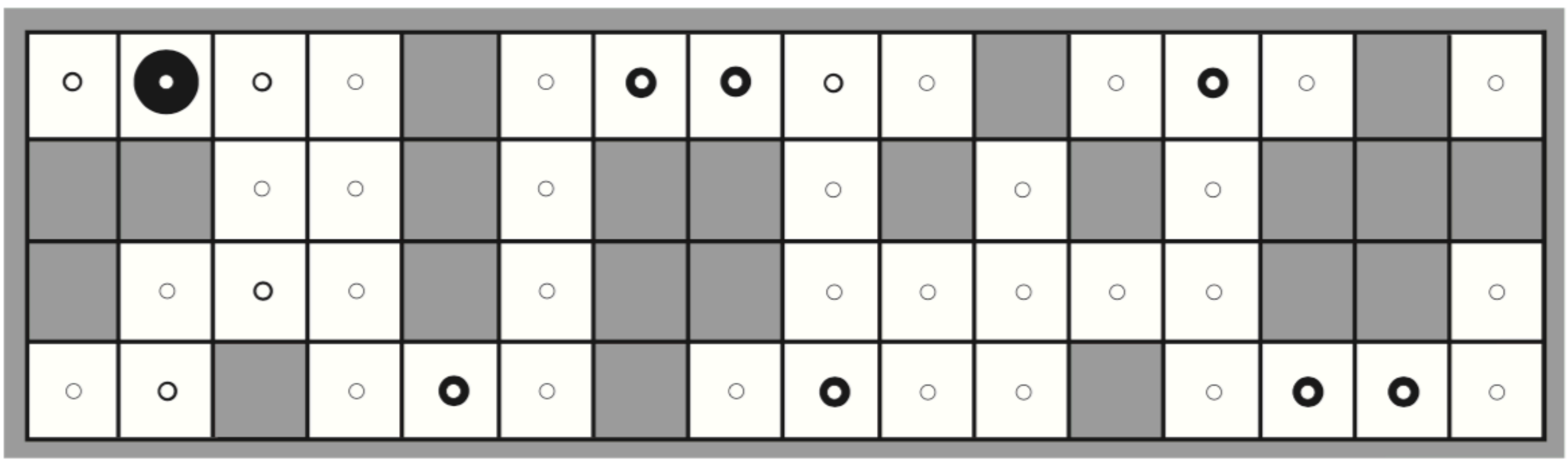
(a) Possible locations of robot after $E_1 = NSW$



(b) Possible locations of robot After $E_1 = NSW, E_2 = NS$

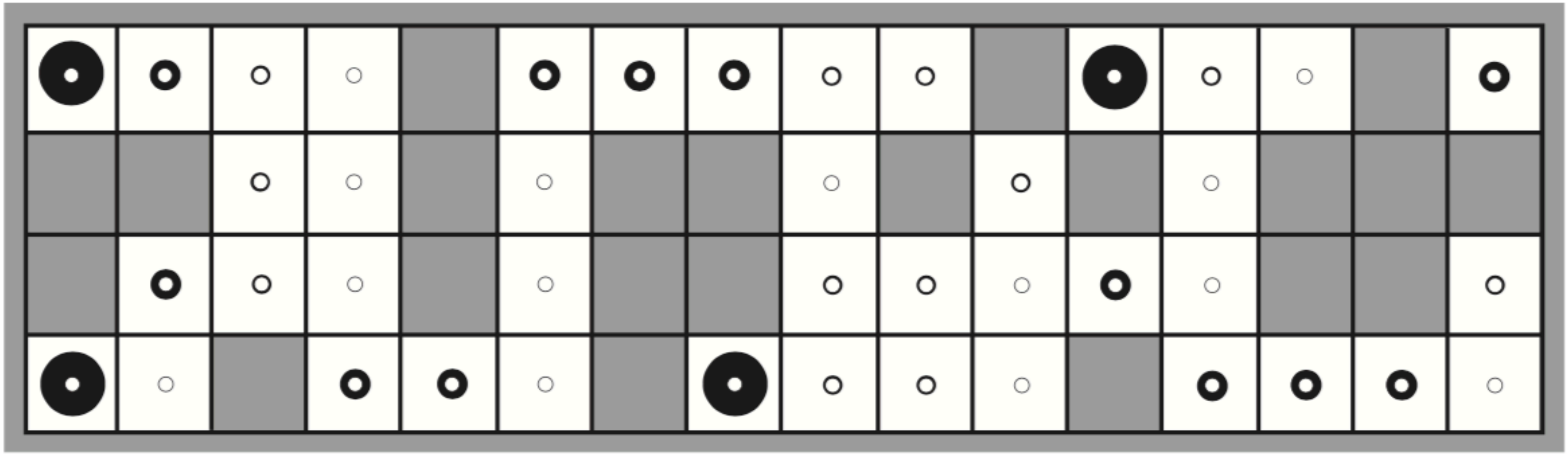


(a) Posterior distribution over robot location after $\mathsf{E}_1 = \mathsf{NSW}$

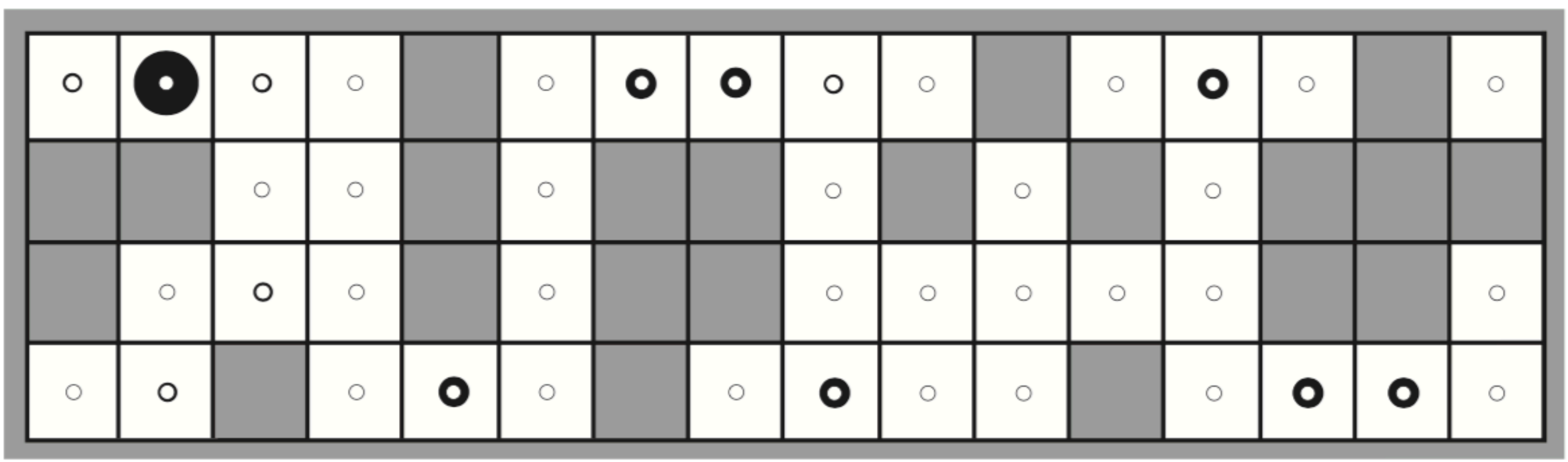


(b) Posterior distribution over robot location after $\mathsf{E}_1 = \mathsf{NSW}, \mathsf{E}_2 = \mathsf{NS}$

Figure 15.7 Posterior distribution over robot location: (a) one observation $E_1 = NSW$; (b) after a second observation $E_2 = NS$. The size of each disk corresponds to the probability that the robot is at that location. The sensor error rate is $\epsilon = 0.2$.



(a) Posterior distribution over robot location after $\mathsf{E}_1 = \mathsf{NSW}$



(b) Posterior distribution over robot location after $\mathsf{E}_1 = \mathsf{NSW}, \mathsf{E}_2 = \mathsf{NS}$

Figure 15.7 Posterior distribution over robot location: (a) one observation $E_1 = NSW$; (b) after a second observation $E_2 = NS$. The size of each disk corresponds to the probability that the robot is at that location. The sensor error rate is $\epsilon = 0.2$.

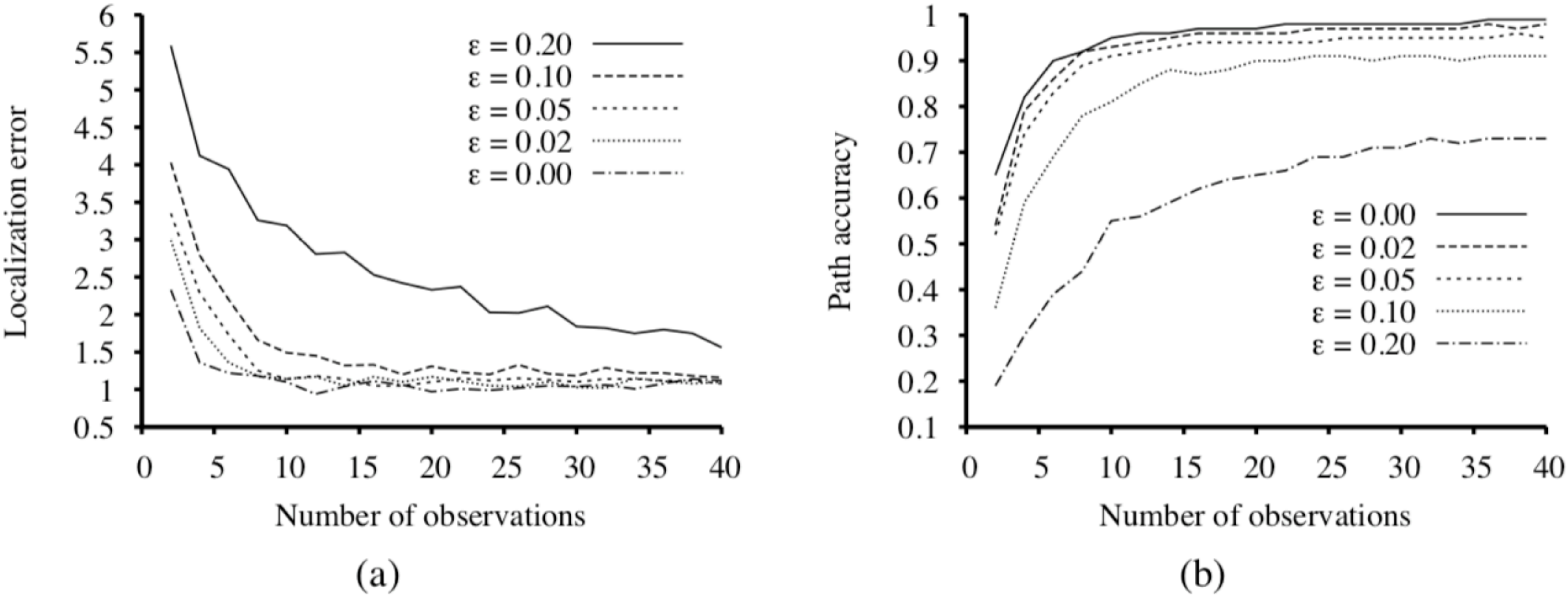


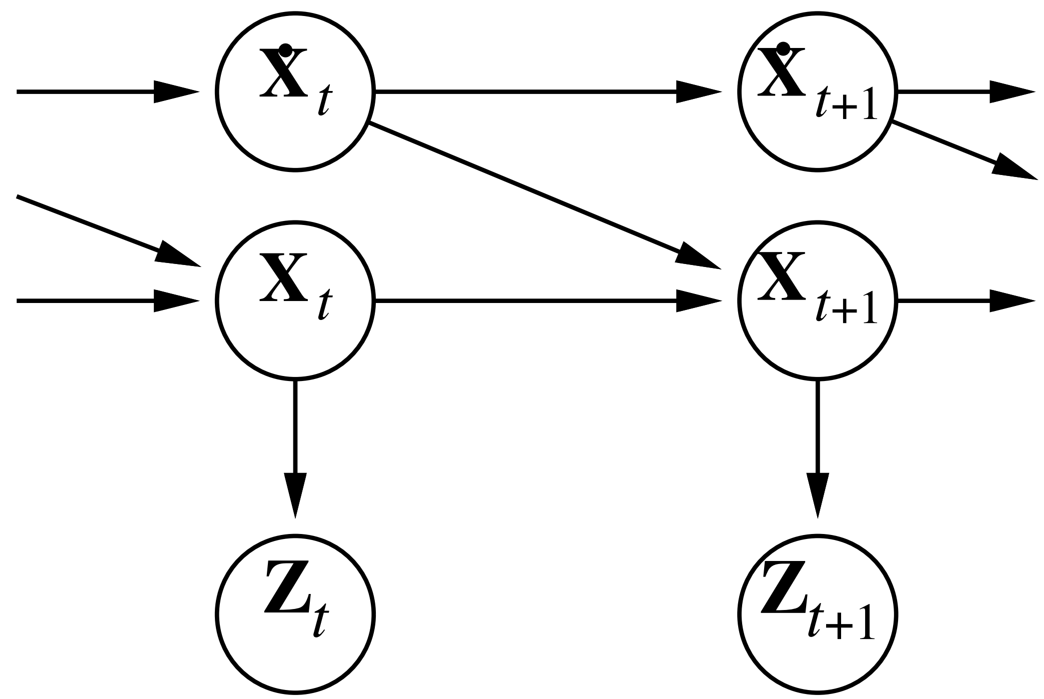
Figure 15.8 Performance of HMM localization as a function of the length of the observation sequence for various different values of the sensor error probability ϵ; data averaged over 400 runs. (a) The localization error, defined as the Manhattan distance from the true location. (b) The Viterbi path accuracy, defined as the fraction of correct states on the Viterbi path.

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

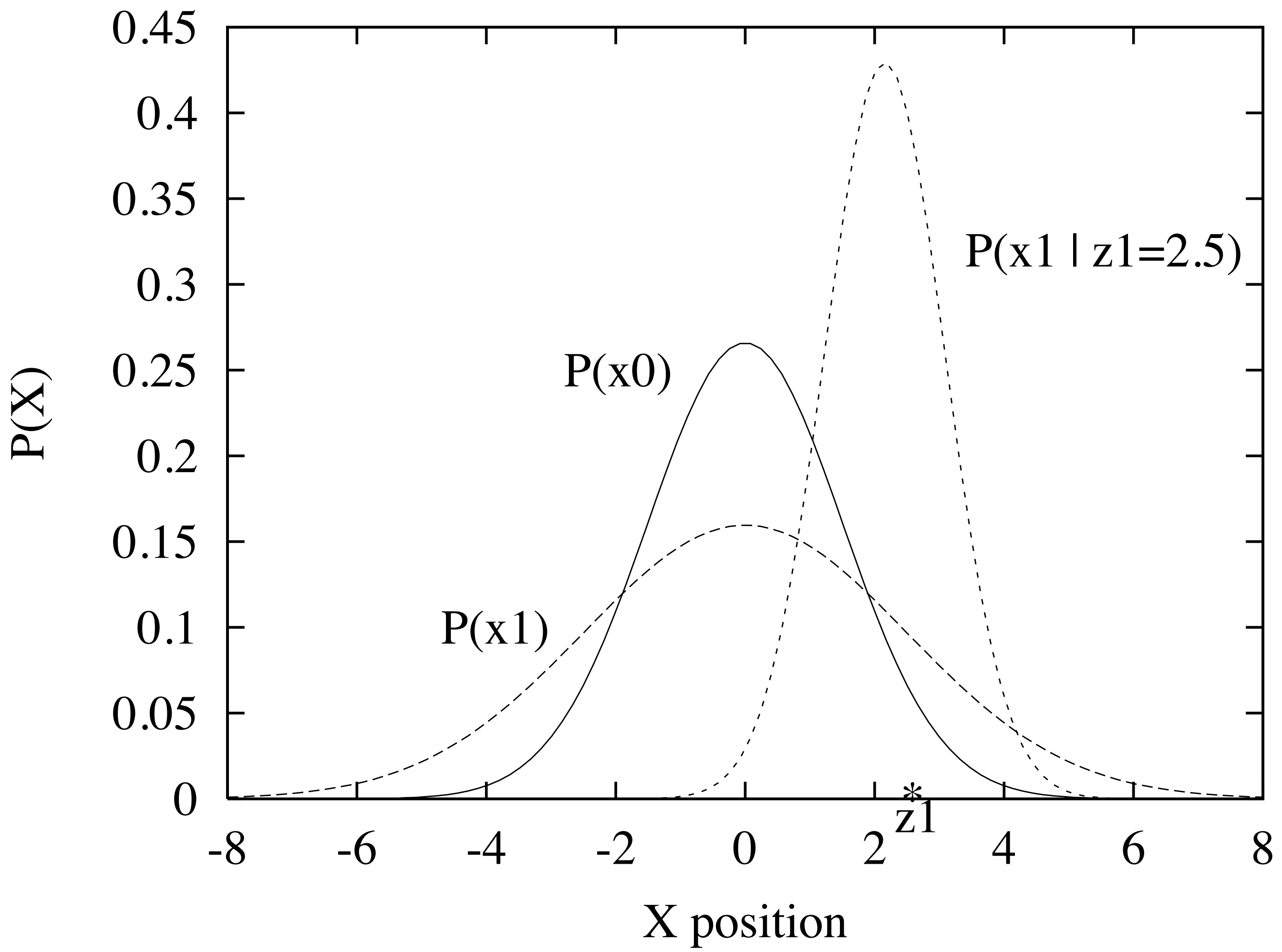
Hence $\mathbf{P}(\mathbf{X}_t|\mathbf{e}_{1:t})$ is multivariate Gaussian $N(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ for all t

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

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

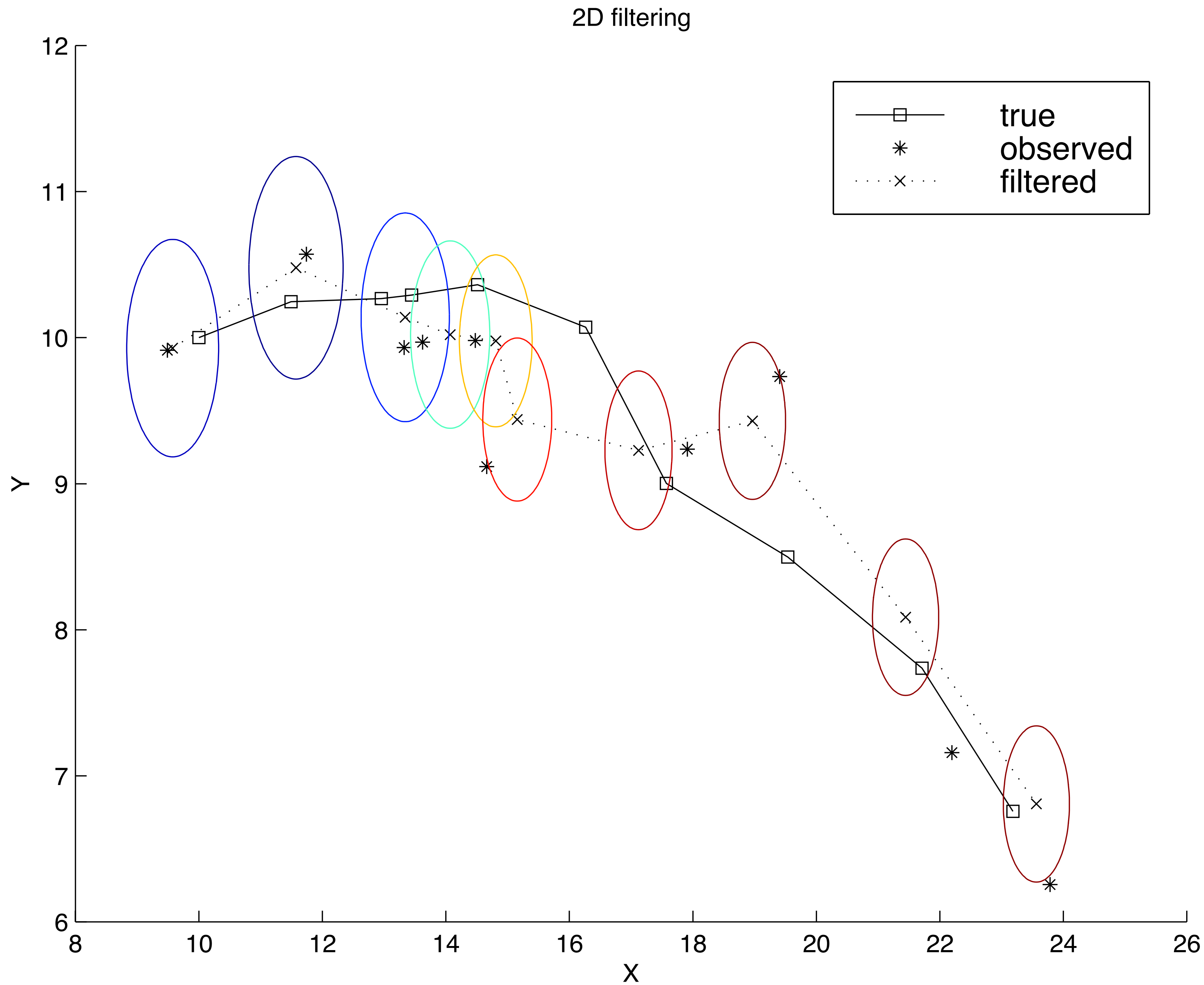
Filter computes the following update:

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

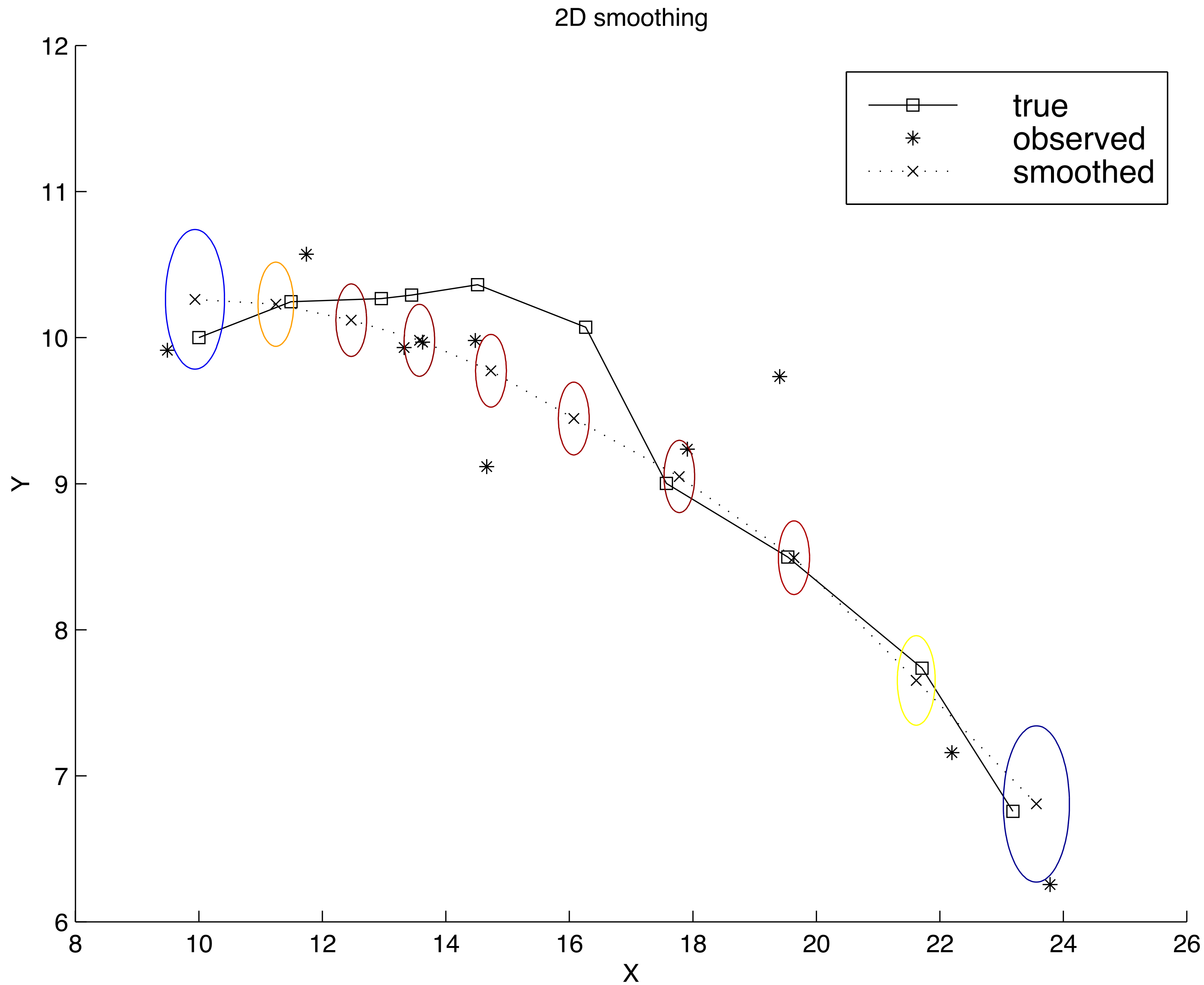
where $\mathbf{K}_{t+1} = (\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^\top + \mathbf{\Sigma}_x)\mathbf{H}^\top(\mathbf{H}(\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^\top + \mathbf{\Sigma}_x)\mathbf{H}^\top + \mathbf{\Sigma}_z)^{-1}$
is the Kalman gain matrix

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

2-D tracking example: filtering



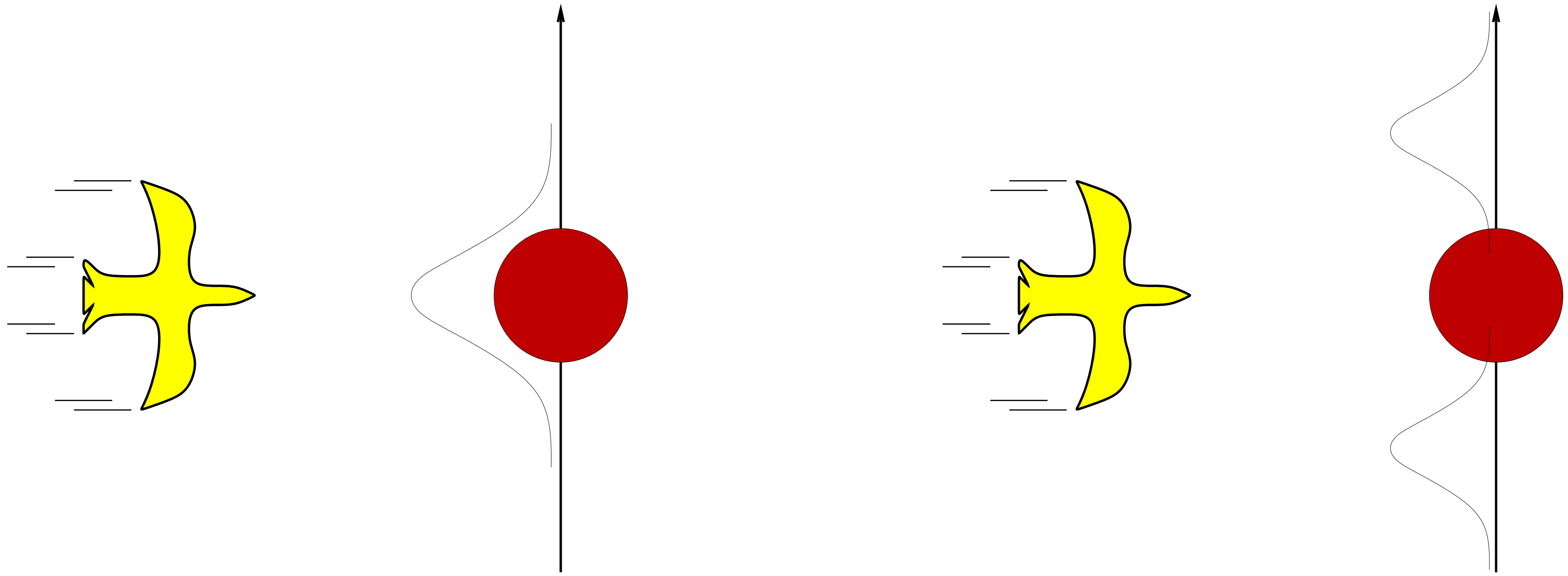
2-D tracking example: smoothing



Where it breaks

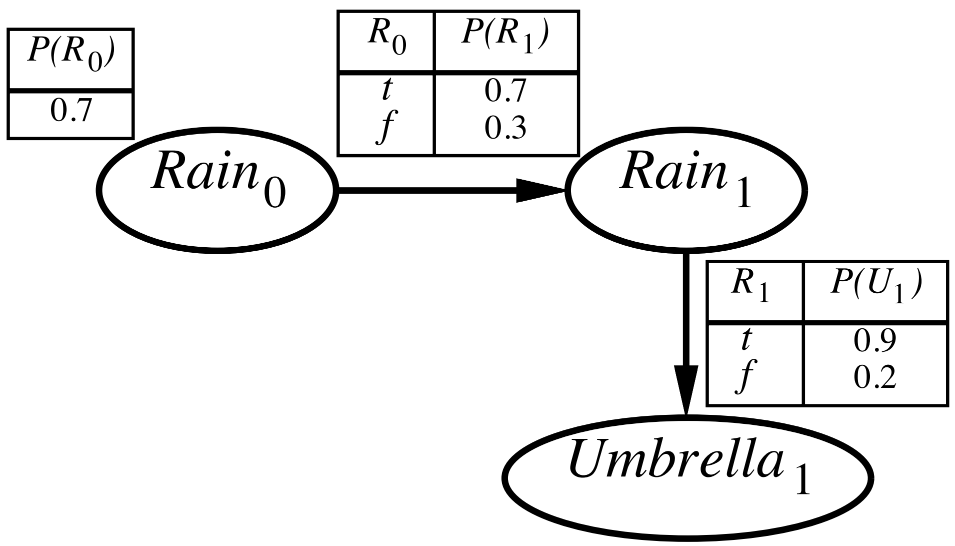
Cannot be applied if the transition model is nonlinear

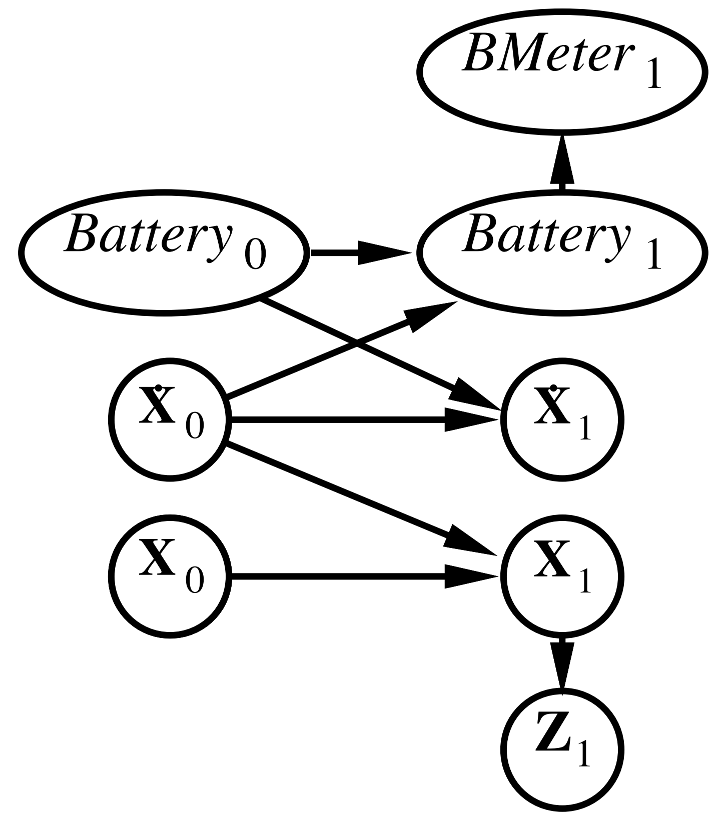
Extended Kalman Filter models transition as **locally linear** around $\mathbf{x}_t = \boldsymbol{\mu}_t$
Fails if systems is locally unsmooth



Dynamic Bayesian networks

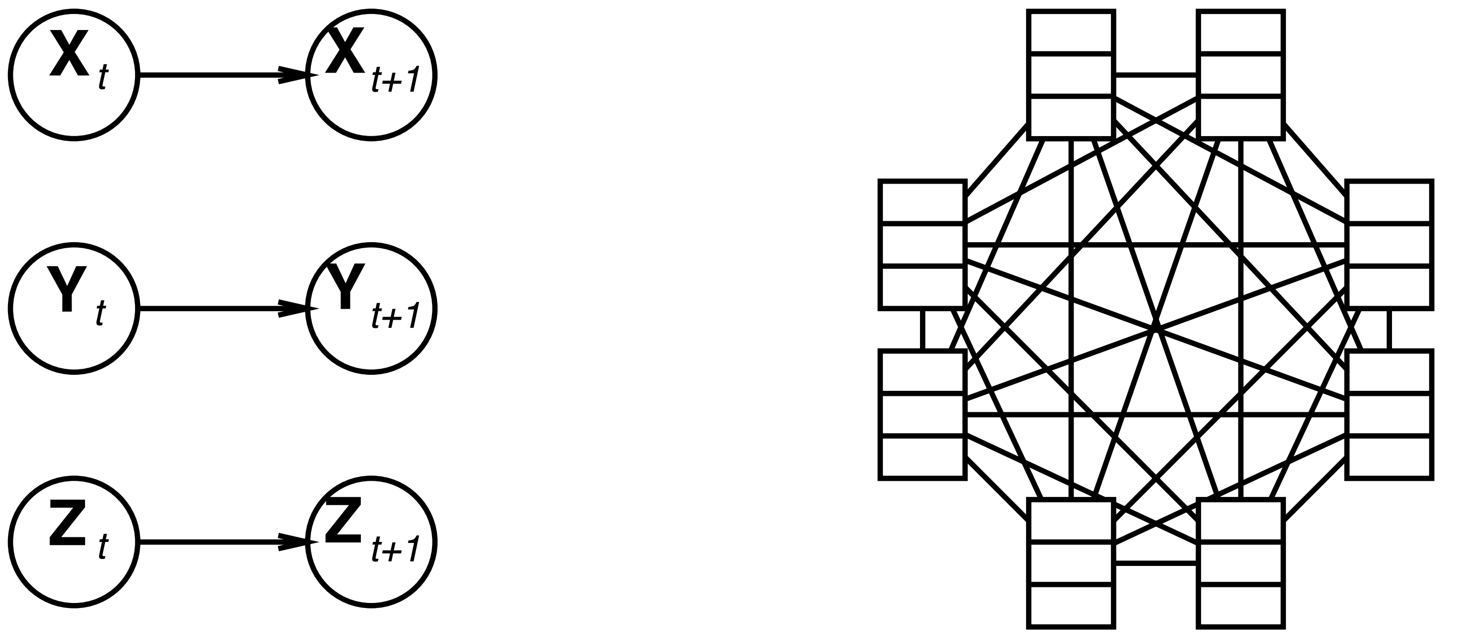
$\mathbf{X}_t$, $\mathbf{E}_t$ contain arbitrarily many variables in a replicated Bayes net





DBNs vs. HMMs

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

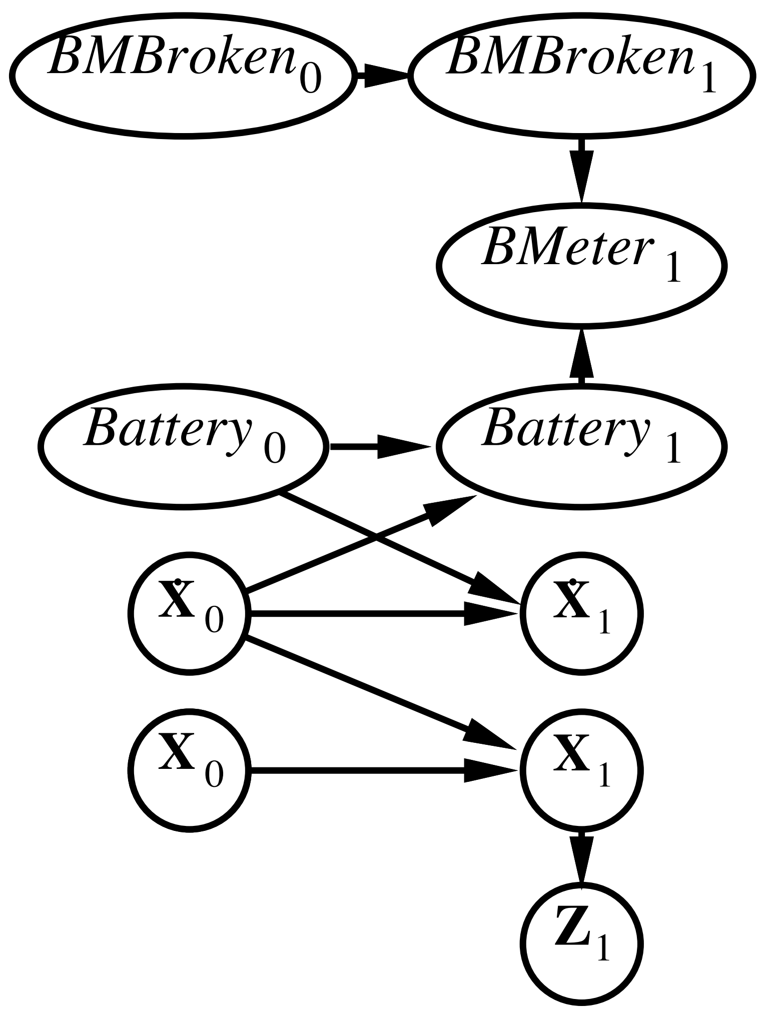


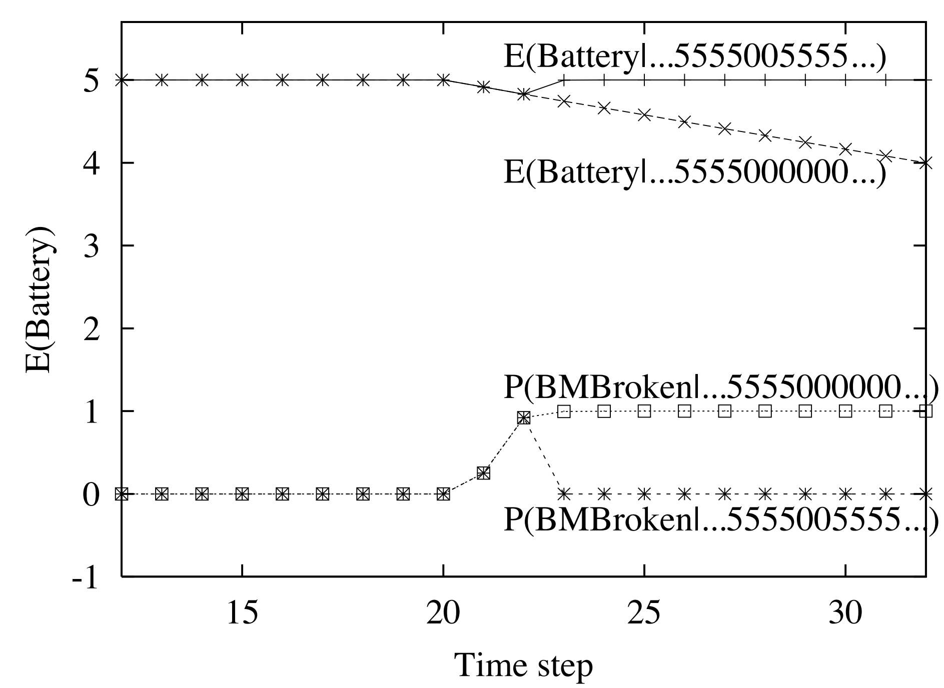
Sparse dependencies $\Rightarrow$ exponentially fewer parameters;
e.g., 20 state variables, three parents each
DBN has $20 \times 2^3 = 160$ parameters, HMM has $2^{20} \times 2^{20} \approx 10^{12}$

DBNs vs Kalman filters

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

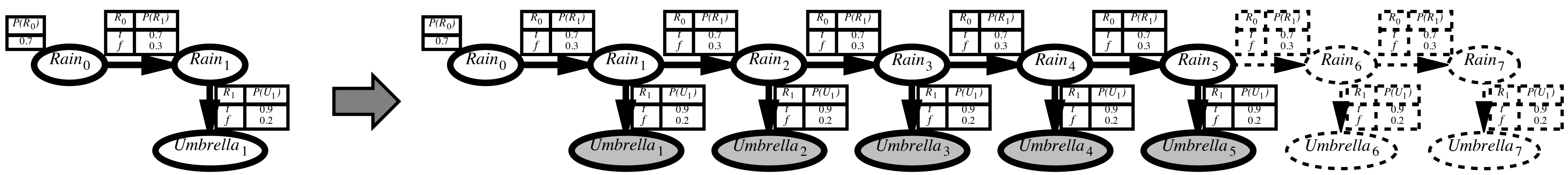
E.g., where are bin Laden and my keys? What's the battery charge?





Exact inference in DBNs

Naive method: unroll the network and run any exact algorithm



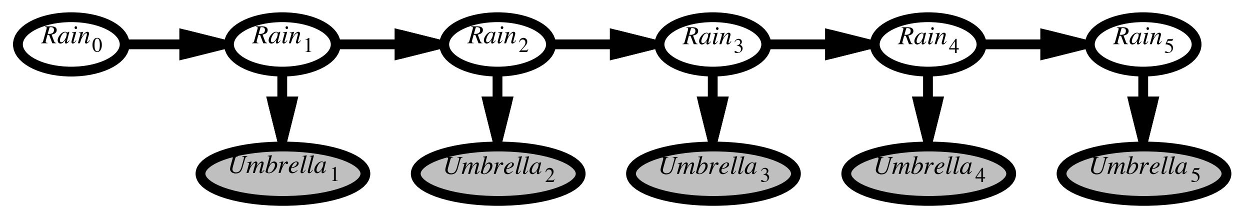
Problem: inference cost for each update grows with t

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

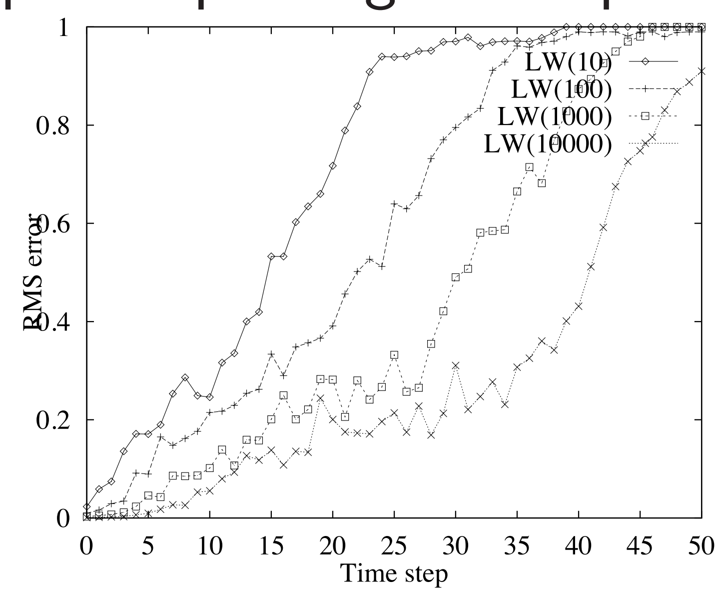
Set of weighted samples approximates the belief state



LW samples pay no attention to the evidence!

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

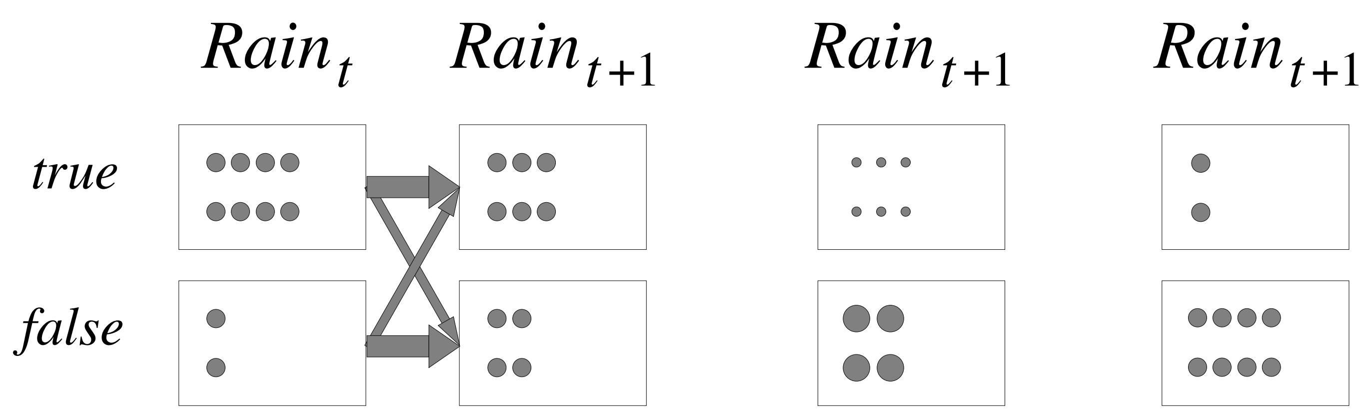
$\Rightarrow$ number of samples required grows exponentially with t



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 $\mathbf{e}_t$



(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
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 $\mathbf{e}_{t+1}$:

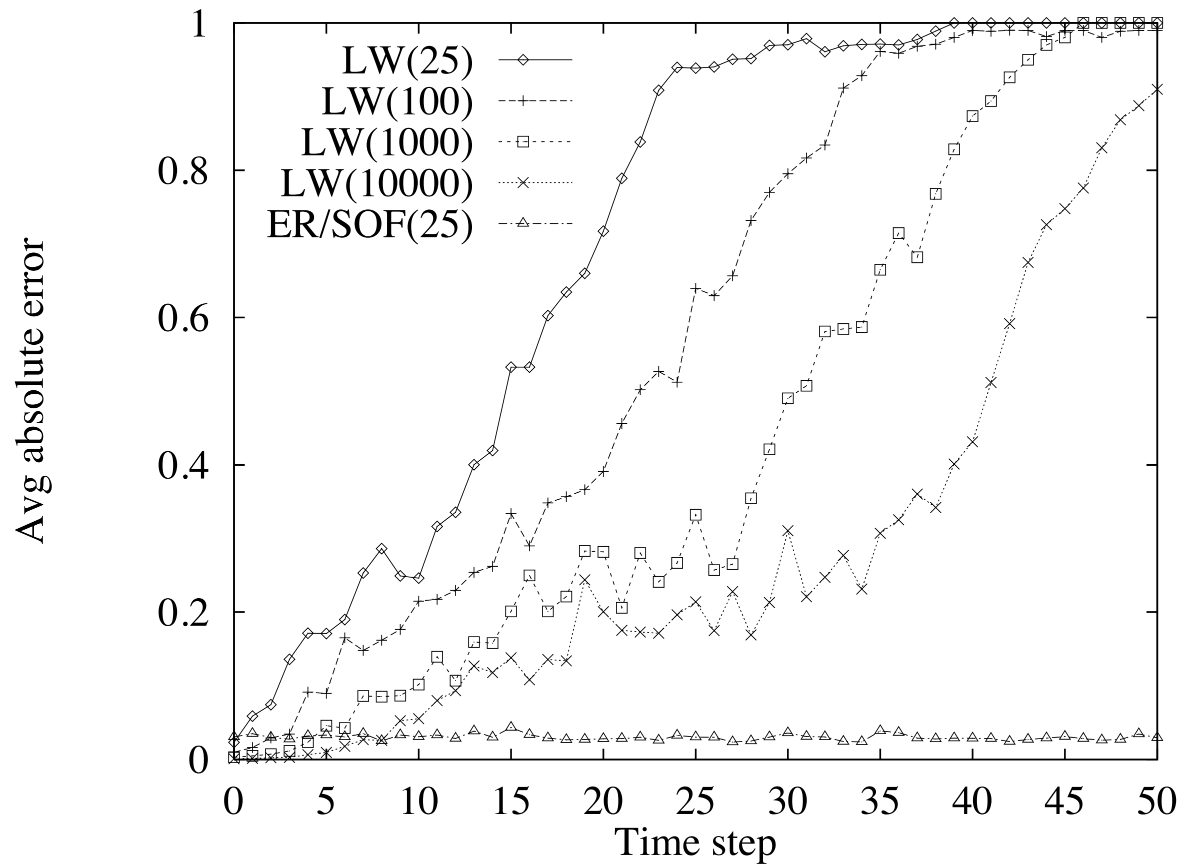
$$W(\mathbf{x}_{t+1}|\mathbf{e}_{1:t+1}) = P(\mathbf{e}_{t+1}|\mathbf{x}_{t+1}) N(\mathbf{x}_{t+1}|\mathbf{e}_{1:t})$$

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 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

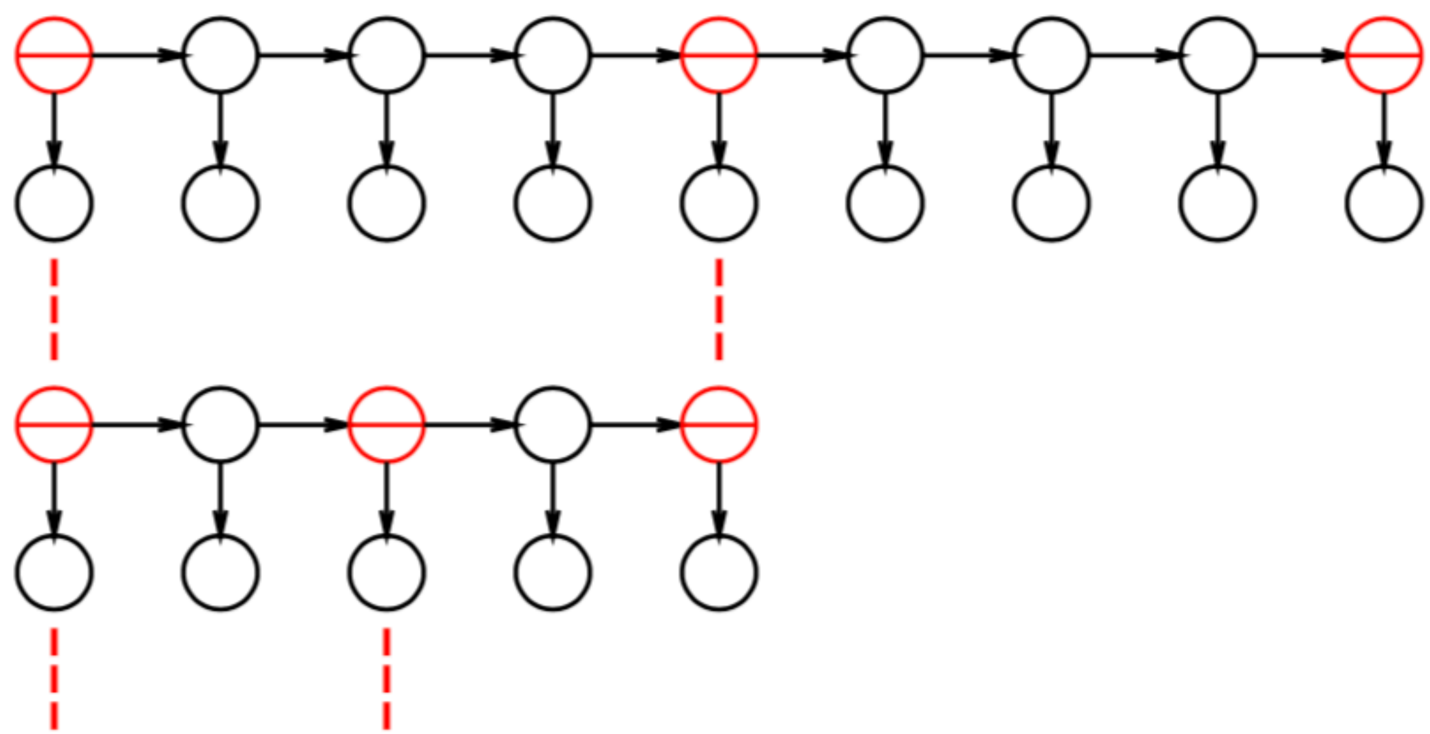
Kalman filters allow n state variables, linear Gaussian, $O(n^3)$ update

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

Particle filtering is a good approximate filtering algorithm for DBNs

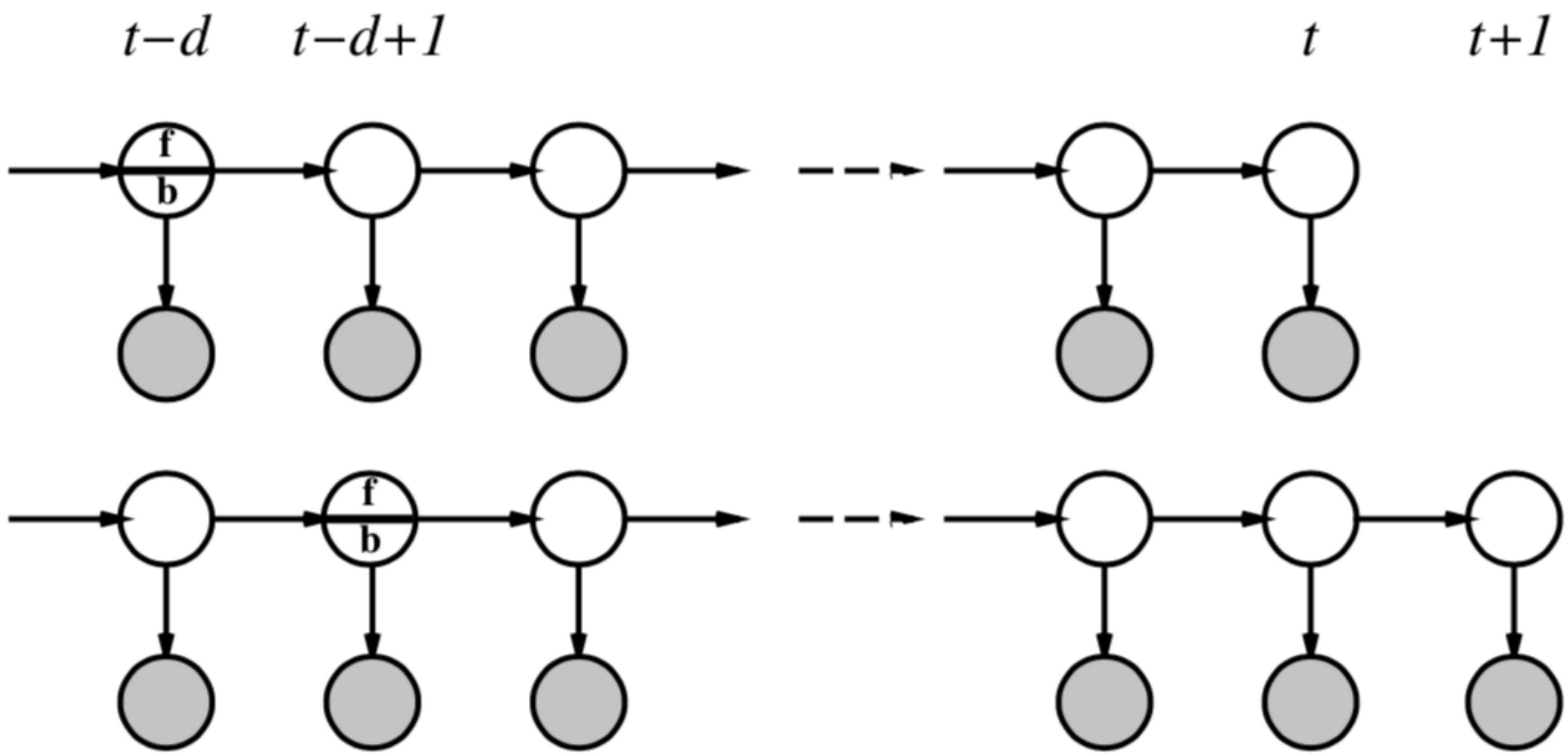
Island algorithm

Idea: run forward-backward storing $\mathbf{f}_t$, $\mathbf{b}_t$ at only $k-1$ points
Call recursively (depth-first) on k subtasks



$O(k|\mathbf{f}|\log_k t)$ space, $O(k \log_k t)$ more time

Online fixed-lag smoothing



Obvious method runs forward–backward for d steps each time

Recursively compute $\mathbf{f}_{1:t-d+1},\ \mathbf{b}_{t-d+2:t+1}$ from $\mathbf{f}_{1:t-d},\ \mathbf{b}_{t-d+1:t}$?

Forward message OK, 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

Approximate inference in DBNs

Particle filtering (Gordon, 1994; Kanazawa, Koller, and Russell, 1995; Blake and Isard, 1996)

Factored approximation (Boyen and Koller, 1999)

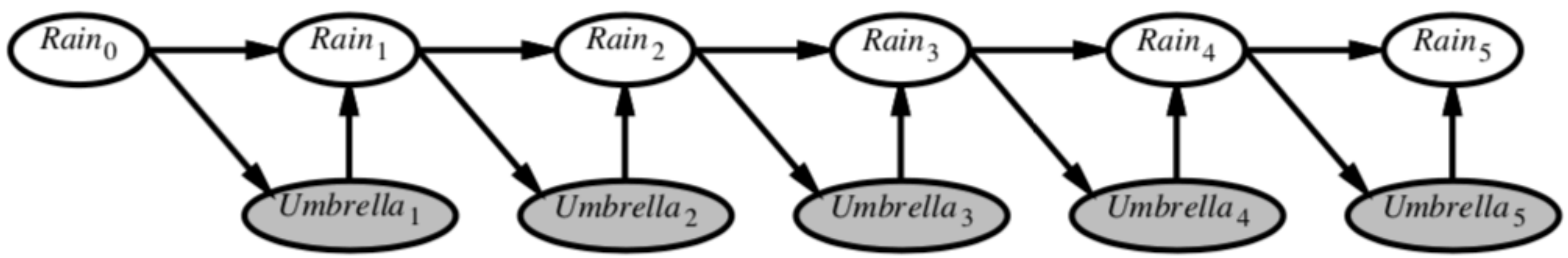
Loopy propagation (Pearl, 1988; Yedidia, Freeman, and Weiss, 2000)

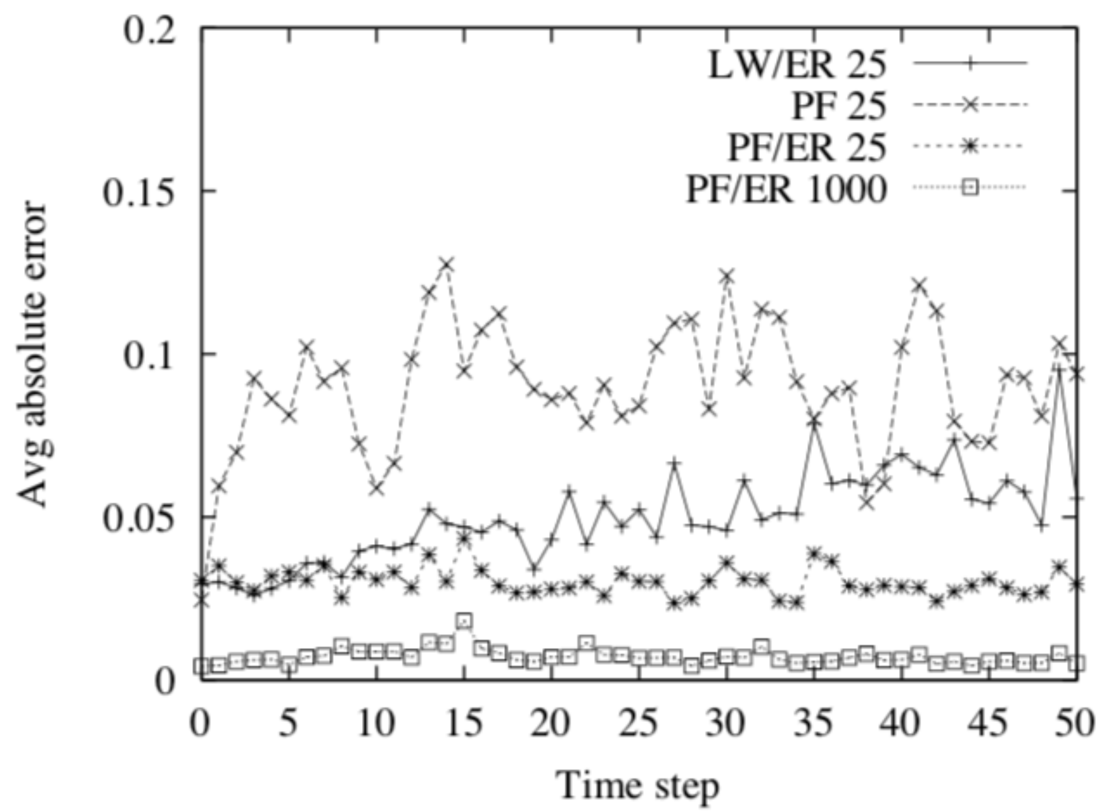
Variational approximation (Ghahramani and Jordan, 1997)

Decayed MCMC (unpublished)

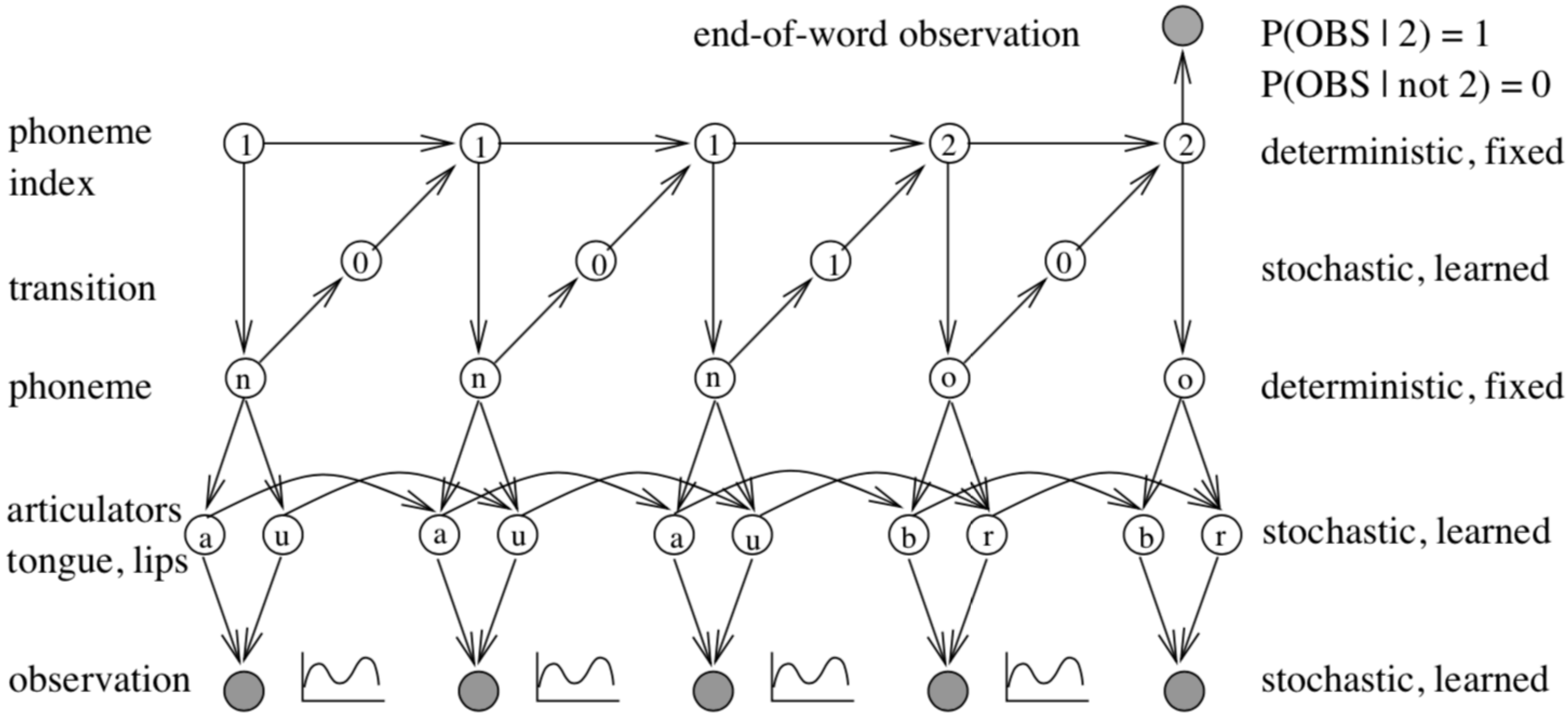
Evidence reversal

Better to propose new samples conditioned on the new evidence
Minimizes the variance of the posterior estimates (Kong & Liu, 1996)





Example: DBN for speech recognition



Also easy to add variables for, e.g., gender, accent, speed.

Zweig and Russell (1998) show up to 40% error reduction over HMMs

Vectors, Matrices, and Linear Algebra

Vector as ordered sequence of values, e.g. $\mathbf{x} = \langle 3, 4 \rangle$, $\mathbf{y} = \langle 0, 2 \rangle$

Fundamental operations:
– Vector Addition: $\mathbf{x} + \mathbf{y}$ is elementwise sum: $\mathbf{x} + \mathbf{y} = \langle 3+0, 4+2 \rangle = \langle 3, 6 \rangle$
– Scalar multiplication: $5\mathbf{x} = \langle 5 \times 3, 5 \times 4 \rangle = \langle 15, 20 \rangle$

Length of $\mathbf{x}$: $|\mathbf{x}| = \sqrt{(3^2 + 4^2)}$

Dot product $\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i = 3 \times 0 + 4 \times 2 = 8$

Matrix, e.g. 3x4 $\begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} \\ A_{3,1} & A_{3,2} & A_{3,3} & A_{3,4} \end{pmatrix}$

Sum $(\mathbf{A} + \mathbf{B})_{i,j} = \mathbf{A}_{i,j} + \mathbf{B}_{i,j}$, undefined if sizes are different.

Vectors, Matrices, and Linear Algebra

Multiplication by scalar: $(c\mathbf{A})_{i,j} = c\mathbf{A}_{i,j}$.

Matrix multiplication: $\mathbf{AB}$: $\mathbf{A}$ has to be of size $a \times b$ and $\mathbf{B}$ of size $b \times c$, result is matrix of size $a \times c$.

$$(\mathbf{AB})_{i,k} = \sum_j \mathbf{A}_{i,j}\mathbf{B}_{j,k}$$

Dot product can be expressed as a transpose and a matrix multiplication: $\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T\mathbf{y}$.

Identity matrix $\mathbf{I}$ has elements $\mathbf{I}_{i,j} = 1$ when $i = j$ and 0 otherwise.

Transpose matrix: turning rows into columns and vice versa.

Inverse matrix: $\mathbf{A}^{-1}$, square matrix such that $\mathbf{AA}^{-1} = \mathbf{I}$.

Vectors, Matrices, and Linear Algebra

Matrices to solve linear equations in $O(n^3)$ time. Example:

$$\begin{aligned} 2x + y - z &= 8 \\ -3x - y + 2z &= -11 \\ -2x + y + 2z &= -3 \end{aligned}$$

We can represent this system as a matrix equation $\mathbf{A}\ \mathbf{x} = \mathbf{b}$, where

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{pmatrix}, \ \mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \ \mathbf{b} = \begin{pmatrix} 8 \\ -11 \\ -3 \end{pmatrix}$$

To solve, multiply both sides by $\mathbf{A}^{-1}$:
$\mathbf{A}^{-1}\mathbf{A}\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ which simplified is $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$

Then invert $\mathbf{A}$ and multiply by $\mathbf{b}$ we get $\mathbf{x} = \begin{pmatrix} 2 \\ 3 \\ -1 \end{pmatrix}$