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

\[ X_t = \text{set of unobservable state variables at time } t \]
  \[ \text{e.g., } \text{BloodSugar}_t, \text{StomachContents}_t, \text{etc.} \]

\[ E_t = \text{set of observable evidence variables at time } t \]
  \[ \text{e.g., } \text{MeasuredBloodSugar}_t, \text{PulseRate}_t, \text{FoodEaten}_t \]

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

Notation: \[ X_{a:b} = X_a, X_{a+1}, \ldots, X_{b-1}, X_b \]
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:** $X_t$ depends on **bounded** subset of $X_{0:t-1}$

**First-order Markov process:** $P(X_t|X_{0:t-1}) = P(X_t|X_{t-1})$

**Second-order Markov process:** $P(X_t|X_{0:t-1}) = P(X_t|X_{t-2}, X_{t-1})$

**Sensor Markov assumption:** $P(E_t|X_{0:t}, E_{0:t-1}) = P(E_t|X_t)$

**Stationary** process: transition model $P(X_t|X_{t-1})$ and sensor model $P(E_t|X_t)$ fixed for all $t$
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, \( P(X_0) \)

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

\[
P(X_{0:t}, E_{1:t}) = P(X_0) \prod_{i=1}^{t} P(X_i|X_{i-1})P(E_i|X_i).
\]

with

Initial state model: \( P(X_0) \)
Transition model: \( P(X_i|X_{i-1}) \)
Sensor model: \( P(E_i|X_i) \).
Get started

How to get started?

The prior probability distribution at time 0, $P(X_0)$

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

$$P(X_{0:t}, E_{1:t}) = P(X_0) \prod_{i=1}^{t} P(X_i|X_{i-1})P(E_i|X_i).$$

with

Initial state model: $P(X_0)$
Transition model: $P(X_i|X_{i-1})$
Sensor model: $P(E_i|X_i)$. 
Inference tasks

Filtering: $P(X_t|e_{1:t})$
belief state—input to the decision process of a rational agent

Prediction: $P(X_{t+k}|e_{1:t})$ for $k > 0$
evaluation of possible action sequences;
like filtering without the evidence

Smoothing: $P(X_k|e_{1:t})$ for $0 \leq k < t$
better estimate of past states, essential for learning

Most likely explanation: $\arg \max_{x_{1:t}} P(x_{1:t}|e_{1:t})$
speech recognition, decoding with a noisy channel
Filtering

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

\[
P(X_{t+1}|e_{1:t+1}) = f(e_{t+1}, P(X_t|e_{1:t}))
\]

\[
P(X_{t+1}|e_{1:t+1}) = P(X_{t+1}|e_{1:t}, e_{t+1})
= \alpha P(e_{t+1}|X_{t+1}, e_{1:t})P(X_{t+1}|e_{1:t})
= \alpha P(e_{t+1}|X_{t+1})P(X_{t+1}|e_{1:t})
\]

I.e., prediction + estimation. Prediction by summing out \(X_t\):

\[
P(X_{t+1}|e_{1:t+1}) = \alpha P(e_{t+1}|X_{t+1})\sum_{x_t} P(X_{t+1}|x_t, e_{1:t})P(x_t|e_{1:t})
= \alpha P(e_{t+1}|X_{t+1})\sum_{x_t} P(X_{t+1}|x_t)P(x_t|e_{1:t})
\]

\[
f_{1:t+1} = \text{FORWARD}(f_{1:t}, e_{t+1}) \quad \text{where} \quad f_{1:t} = P(X_t|e_{1:t})
\]

Time and space **constant** (independent of \(t\))
Filtering example

\[ \begin{align*}
    \text{True} & : 0.500 \\
    \text{False} & : 0.500 \\
\end{align*} \]

\[ \begin{align*}
    \text{True} & : 0.500 & \text{False} & : 0.500 \\
    \text{True} & : 0.818 & \text{False} & : 0.182 \\
    \text{True} & : 0.500 & \text{False} & : 0.500 \\
    \text{True} & : 0.627 & \text{False} & : 0.373 \\
    \text{True} & : 0.883 & \text{False} & : 0.117 \\
\end{align*} \]
Filtering example

```
True    0.500  0.500  0.627
False   0.500  0.818  0.373

Rain_0  Rain_1  Rain_2

Umbrella_1  Umbrella_2
```

Chapter 15, Sections 1–5
Filtering example

\[
\begin{align*}
\text{True} & : 0.500 \\
\text{False} & : 0.500 \\
\end{align*}
\]

\[
\begin{align*}
0.500 & \quad 0.500 & & 0.627 \\
0.818 & & 0.373 & & \\
0.182 & & 0.883 & & \\
0.117 & & & & \\
\end{align*}
\]

\[
\begin{align*}
\text{Rain}_0 & \rightarrow \text{Rain}_1 & \rightarrow \text{Rain}_2 \\
\text{Umbrella}_1 & & \text{Umbrella}_2 \\
\end{align*}
\]
Divide evidence $e_{1:t}$ into $e_{1:k}$, $e_{k+1:t}$:

$$P(X_k|e_{1:t}) = P(X_k|e_{1:k}, e_{k+1:t})$$
$$= \alpha P(X_k|e_{1:k}) P(e_{k+1:t}|X_k, e_{1:k})$$
$$= \alpha P(X_k|e_{1:k}) P(e_{k+1:t}|X_k)$$
$$= \alpha f_{1:k} \times b_{k+1:t}$$

Backward message computed by a backwards recursion:

$$P(e_{k+1:t}|X_k) = \sum_{x_{k+1}} P(e_{k+1:t}|X_k, x_{k+1}) P(x_{k+1}|X_k)$$
$$= \sum_{x_{k+1}} P(e_{k+1:t}|x_{k+1}) P(x_{k+1}|X_k)$$
$$= \sum_{x_{k+1}} P(e_{k+1}|x_{k+1}) P(e_{k+2:t}|x_{k+1}) P(x_{k+1}|X_k)$$
Forward–backward algorithm: cache forward messages along the way. Time linear in $t$ (polytree inference), space $O(t |f|)$
**Smoothing example**

Forward–backward algorithm: cache forward messages along the way

Time linear in $t$ (polytree inference), space $O(t|f|)$
Most likely explanation

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

Most likely path to each $x_{t+1}$

$=$ most likely path to some $x_t$ plus one more step

$$\max_{x_1 \ldots x_t} P(x_1, \ldots, x_t, X_{t+1} | e_{1:t+1})$$

$=$ $P(e_{t+1} | X_{t+1}) \max_{x_t} \left( P(X_{t+1} | x_t) \max_{x_1 \ldots x_{t-1}} P(x_1, \ldots, x_{t-1}, x_t | e_{1:t}) \right)$

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

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

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:

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

Rain_1  |  Rain_2  |  Rain_3  |  Rain_4  |  Rain_5
---|---|---|---|---
true | true | true | true | true
false | false | false | false | false
true | true | false | true | true
true | true | false | true | true

Most likely paths:

m_1:1: 0.8182
m_1:2: 0.5155
m_1:3: 0.0361
m_1:4: 0.0334
m_1:5: 0.0210

Umbrella:

true true false true true

State space paths:

m_1:1: 0.1818
m_1:2: 0.0491
m_1:3: 0.1237
m_1:4: 0.0173
m_1:5: 0.0024
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, P(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

    fv[0] ← prior
    for i = 1 to t do
        fv[i] ← FORWARD(fv[i - 1], ev[i])
    for i = t down to 1 do
        sv[i] ← NORMALIZE(fv[i] \times b)
        b ← BACKWARD(b, 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**

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

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

Sensor matrix $O_t$ for each time step, diagonal elements $P(e_t | X_t = i)$
e.g., with $U_1 = true$, $O_1 = 
\begin{pmatrix}
0.9 & 0 \\
0 & 0.2
\end{pmatrix}$

Forward and backward messages as column vectors:

$$f_{1:t+1} = \alpha O_{t+1} T^\top f_{1:t}$$
$$b_{k+1:t} = TO_{k+1} b_{k+2:t}$$

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

$X_t$ is a single, discrete variable (usually $E_t$ is too)

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

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

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

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

Forward and backward messages as column vectors:

\[
\begin{align*}
\mathbf{f}_{1:t+1} &= \alpha O_{t+1} \mathbf{T}^\top \mathbf{f}_{1:t} \\
\mathbf{b}_{k+1:t} &= \mathbf{T} O_{k+1} \mathbf{b}_{k+2:t}
\end{align*}
\]

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

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

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

Forward and backward messages as column vectors:

$$f_{1:t+1} = \alpha O_{t+1} T^T f_{1:t}$$
$$b_{k+1:t} = TO_{k+1} b_{k+2:t}$$

Forward-backward algorithm needs time $O(S^2t)$ and space $O(St)$
function **FIXED-LAG-SMOOTHING**($e_t, hmm, d$) returns a distribution over $X_{t-d}$

inputs: $e_t$, the current evidence for time step $t$
$hmm$, a hidden Markov model with $S \times S$ transition matrix $T$
$d$, the length of the lag for smoothing

persistent: $t$, the current time, initially 1
$f$, the forward message $P(X_t|e_{1:t})$, 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: $O_{t-d}, O_t$, diagonal matrices containing the sensor model information

add $e_t$ to the end of $e_{t-d:t}$
$O_t \leftarrow$ diagonal matrix containing $P(e_t|X_t)$

if $t > d$ then
    $f \leftarrow \text{FORWARD}(f, e_t)$
    remove $e_{t-d-1}$ from the beginning of $e_{t-d:t}$
    $O_{t-d} \leftarrow$ diagonal matrix containing $P(e_{t-d}|X_{t-d})$
    $B \leftarrow O_{t-d}^{-1} T^{-1} BTO_t$
else $B \leftarrow BTO_t$
    $t \leftarrow t + 1$
endif $t > d$ then return $\text{NORMALIZE}(f \times B1)$ 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 $\text{NORMALIZE}(f \times B1)$ is just $\alpha f \times b$, by Equation (15.14).
Country dance algorithm

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

\[
\begin{align*}
    f_{1:t+1} &= \alpha O_{t+1} T^\top f_{1:t} \\
    O_{t+1}^{-1} f_{1:t+1} &= \alpha T^\top f_{1:t} \\
    \alpha'(T^\top)^{-1} O_{t+1}^{-1} f_{1:t+1} &= f_{1:t}
\end{align*}
\]

Algorithm: forward pass computes \( f_t \), backward pass does \( f_i, b_i \)
Country dance algorithm

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

\[ f_{1:t+1} = \alpha O_{t+1} T^\top f_{1:t} \]
\[ O_{t+1}^{-1} f_{1:t+1} = \alpha T^\top f_{1:t} \]
\[ \alpha'(T^\top)^{-1} O_{t+1}^{-1} f_{1:t+1} = f_{1:t} \]

Algorithm: forward pass computes \( f_t \), backward pass does \( f_i, b_i \)
Country dance algorithm

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

\[
\begin{align*}
  f_{1:t+1} &= \alpha O_{t+1} T^T f_{1:t} \\
  O_{t+1}^{-1} f_{1:t+1} &= \alpha T^T f_{1:t} \\
  \alpha'(T^T)^{-1} O_{t+1}^{-1} f_{1:t+1} &= f_{1:t}
\end{align*}
\]

Algorithm: forward pass computes \( f_t \), backward pass does \( f_i, b_i \)
Country dance algorithm

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

\[
\begin{align*}
    f_{1:t+1} &= \alpha O_{t+1} T^\top f_{1:t} \\
    O_{t+1}^{-1} f_{1:t+1} &= \alpha T^\top f_{1:t} \\
    \alpha'(T^\top)^{-1} O_{t+1}^{-1} f_{1:t+1} &= f_{1:t}
\end{align*}
\]

Algorithm: forward pass computes \( f_t \), backward pass does \( f_i, b_i \)
Country dance algorithm

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

\[
\begin{align*}
  f_{1:t+1} &= \alpha O_{t+1} T^T f_{1:t} \\
  O_{t+1}^{-1} f_{1:t+1} &= \alpha T^T f_{1:t} \\
  \alpha'(T^T)^{-1} O_{t+1}^{-1} f_{1:t+1} &= f_{1:t}
\end{align*}
\]

Algorithm: forward pass computes \( f_t \), backward pass does \( f_i, b_i \)
Country dance algorithm

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

\[
f_{1:t+1} = \alpha O_{t+1} T^\top f_{1:t} \\
O^{-1}_{t+1} f_{1:t+1} = \alpha T^\top f_{1:t} \\
\alpha'(T^\top)^{-1} O^{-1}_{t+1} f_{1:t+1} = f_{1:t}
\]

Algorithm: forward pass computes \( f_t \), backward pass does \( f_i, b_i \)
Country dance algorithm

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

\[
\begin{align*}
    f_{1:t+1} &= \alpha O_{t+1} T^\top f_{1:t} \\
    O_{t+1}^{-1} f_{1:t+1} &= \alpha T^\top f_{1:t} \\
    \alpha'(T^\top)^{-1} O_{t+1}^{-1} f_{1:t+1} &= f_{1:t}
\end{align*}
\]

Algorithm: forward pass computes \( f_t \), backward pass does \( f_i, b_i \)
Country dance algorithm

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

\[
\begin{align*}
    f_{1:t+1} &= \alpha O_{t+1} T^\top f_{1:t} \\
    O^{-1}_{t+1} f_{1:t+1} &= \alpha T^\top f_{1:t} \\
    \alpha'(T^\top)^{-1} O^{-1}_{t+1} f_{1:t+1} &= f_{1:t}
\end{align*}
\]

Algorithm: forward pass computes \( f_t \), backward pass does \( f_i, b_i \)
Country dance algorithm

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

\[ f_{1:t+1} = \alpha O_{t+1} T^\top f_{1:t} \]
\[ O_{t+1}^{-1} f_{1:t+1} = \alpha T^\top f_{1:t} \]
\[ \alpha' (T^\top)^{-1} O_{t+1}^{-1} f_{1:t+1} = f_{1:t} \]

Algorithm: forward pass computes \( f_t \), backward pass does \( f_i, b_i \)
Country dance algorithm

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

\[
\begin{align*}
    f_{t+1} & = \alpha O_{t+1} T^T f_{1:t} \\
    O_{t+1}^{-1} f_{1:t+1} & = \alpha T^T f_{1:t} \\
    \alpha' (T^T)^{-1} O_{t+1}^{-1} f_{1:t+1} & = f_{1:t}
\end{align*}
\]

Algorithm: forward pass computes \( f_t \), backward pass does \( f_i, b_i \)
(a) Possible locations of robot after $E_1 = \text{NSW}$

(b) Possible locations of robot After $E_1 = \text{NSW}, E_2 = \text{NS}$
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 $E_1 = NSW$

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

**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 $\epsilon$; 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—\(X_t = X, Y, Z, \dot{X}, \dot{Y}, \dot{Z}\).

Airplanes, robots, ecosystems, economies, chemical plants, planets, ...
Updating Gaussian distributions

Prediction step: if $P(X_t|e_{1:t})$ is Gaussian, then prediction

$$P(X_{t+1}|e_{1:t}) = \int_{X_t} P(X_{t+1}|x_t)P(x_t|e_{1:t}) \, dx_t$$

is Gaussian. If $P(X_{t+1}|e_{1:t})$ is Gaussian, then the updated distribution

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

is Gaussian.

Hence $P(X_t|e_{1:t})$ is multivariate Gaussian $N(\mu_t, \Sigma_t)$ 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. $\sigma_x$, sensor s.d. $\sigma_z$

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

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

![Graph showing Gaussian random walk on X-axis with parameters $\mu_t$, $\sigma_t$, $\sigma_x$, and $\sigma_z$.]
General Kalman update

Transition and sensor models:

\[ P(x_{t+1}|x_t) = N(Fx_t, \Sigma_x)(x_{t+1}) \]
\[ P(z_t|x_t) = N(Hx_t, \Sigma_z)(z_t) \]

\( F \) is the matrix for the transition; \( \Sigma_x \) the transition noise covariance
\( H \) is the matrix for the sensors; \( \Sigma_z \) the sensor noise covariance

Filter computes the following update:

\[ \mu_{t+1} = F\mu_t + K_{t+1}(z_{t+1} - HF\mu_t) \]
\[ \Sigma_{t+1} = (I - K_{t+1})(F\Sigma_tF^T + \Sigma_x) \]

where \( K_{t+1} = (F\Sigma_tF^T + \Sigma_x)H^T(H(F\Sigma_tF^T + \Sigma_x)H^T + \Sigma_z)^{-1} \)

is the Kalman gain matrix

\( \Sigma_t \) and \( K_t \) are independent of observation sequence, so compute offline
2-D tracking example: filtering

2D filtering

- true
- observed
- filtered

Chapter 15, Sections 1–5
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
$X_t$, $E_t$ contain arbitrarily many variables in a replicated Bayes net

```
<table>
<thead>
<tr>
<th>$P(R_0)$</th>
<th>$P(R_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>0.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$R_0$</th>
<th>$P(U_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>0.9</td>
</tr>
<tr>
<td>$f$</td>
<td>0.2</td>
</tr>
</tbody>
</table>
```
Every HMM is a single-variable DBN; every discrete DBN is an HMM

\[ X_t \rightarrow X_{t+1} \]

\[ Y_t \rightarrow Y_{t+1} \]

\[ Z_t \rightarrow Z_{t+1} \]

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 \( O(d^{m+2}) \)

(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 $t$
⇒ 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 $e_t$

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 $t$: $N(x_t|e_{1:t})/N = P(x_t|e_{1:t})$

Propagate forward: populations of $x_{t+1}$ are

$$N(x_{t+1}|e_{1:t}) = \sum_{x_t} P(x_{t+1}|x_t)N(x_t|e_{1:t})$$

Weight samples by their likelihood for $e_{t+1}$:

$$W(x_{t+1}|e_{1:t+1}) = P(e_{t+1}|x_{t+1})N(x_{t+1}|e_{1:t})$$

Resample to obtain populations proportional to $W$:

$$N(x_{t+1}|e_{1:t+1})/N = \alpha W(x_{t+1}|e_{1:t+1}) = \alpha P(e_{t+1}|x_{t+1})N(x_{t+1}|e_{1:t})$$

$$= \alpha P(e_{t+1}|x_{t+1}) \sum_{x_t} P(x_{t+1}|x_t)N(x_t|e_{1:t})$$

$$= \alpha' P(e_{t+1}|x_{t+1}) \sum_{x_t} P(x_{t+1}|x_t)P(x_t|e_{1:t})$$

$$= P(x_{t+1}|e_{1:t+1})$$
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 $P(X_t|X_{t-1})$
  – sensor model $P(E_t|X_t)$

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 $f_t$, $b_t$ at only $k - 1$ points
Call recursively (depth-first) on $k$ subtasks

$O(k|f| \log_k t)$ space, $O(k \log_k t)$ more time
Online fixed-lag smoothing

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

Recursively compute $f_{1:t-d+1}$, $b_{t-d+2:t+1}$ from $f_{1:t-d}$, $b_{t-d+1:t}$?

Forward message OK, backward message not directly obtainable
Online fixed-lag smoothing contd.

Define $B_{j:k} = \prod_{i=j}^{k} TO_i$, so

$$b_{t-d+1:t} = B_{t-d+1:t} 1$$
$$b_{t-d+2:t+1} = B_{t-d+2:t+1} 1$$

Now we can get a recursive update for $B$:

$$B_{t-d+2:t+1} = O_{t-d+1}^{-1} T^{-1} B_{t-d+1:t} TO_{t+1}$$

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. \( 3 \times 4 \)

\[
\begin{pmatrix}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\
A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} \\
A_{3,1} & A_{3,2} & A_{3,3} & A_{3,4}
\end{pmatrix}
\]

Sum \( (\mathbf{A} + \mathbf{B})_{i,j} = A_{i,j} + B_{i,j} \), undefined if sizes are different.
Vectors, Matrices, and Linear Algebra

Multiplication by scalar: \((cA)_{i,j} = cA_{i,j}\).

Matrix multiplication: \(AB\): \(A\) has to be of size \(a \times b\) and \(B\) of size \(b \times c\), result is matrix of size \(a \times c\).

\[(AB)_{i,k} = \sum_j A_{i,j}B_{j,k}\]

Dot product can be expressed as a transpose and a matrix multiplication: \(x \cdot y = x^Ty\).

Identity matrix \(I\) has elements \(I_{i,j} = 1\) when \(i = j\) and 0 otherwise.

Transpose matrix: turning rows into columns and vice versa.

Inverse matrix: \(A^{-1}\), square matrix such that \(AA^{-1} = I\).
Matrices to solve linear equations in $O(n^3)$ time. Example:

\[
\begin{align*}
2x + y - z &= 8 \\
-3x - y + 2z &= -11 \\
-2x + y + 2z &= -3
\end{align*}
\]

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

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

To solve, multiply both sides by $A^{-1}$:

$A^{-1}Ax = A^{-1}b$ which simplified is $x = A^{-1}b$

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