Instructor: Sofus A. Macskassy, macskass@usc.edu
TAs: Nadeesha Ranasinghe (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:
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., \( \text{BloodSugar}_t, \text{StomachContents}_t \), etc.

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

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

Notation: \( X_{a:b} = X_a, X_{a+1}, \ldots, X_{b-1}, X_b \)
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: $P(X_t|X_{0:t-1}) = P(X_t|X_{t-2},X_{t-1})$

- First-order:

- Second-order:

- 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 $t$
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 $\text{Temp}_t$, $\text{Pressure}_t$

Example: robot motion.
  - Augment position and velocity with $\text{Battery}_t$
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 $0 < k < t$
  - 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 \textit{recursive} state estimation algorithm:
  \[ P(X_{t+1} | e_{1:t+1}) = f(e_{1:t+1}, P(X_t | e_{1:t})) \]

  \[
  P(X_{t+1} | e_{1:t+1}) = P(X_{t+1} | e_{1:t}, e_{t+1}) \\
  = \alpha P(e_{t+1} | X_{t+1}, e_{1:t}) P(X_{t+1} | e_{1:t}) \quad \text{Divide up evidence} \\
  = \alpha P(e_{t+1} | X_{t+1}) P(X_{t+1} | e_{1:t}) \quad \text{Using Bayes rule} \\
  \]

- I.e., \textit{prediction + estimation}. Prediction by summing out $X_t$:
  \[
  P(X_{t+1} | e_{1:t+1}) = \alpha P(e_{t+1} | X_{t+1}) \sum_{x_t} P(X_{t+1} | x_t, e_{1:t}) P(x_t | e_{1:t}) \\
  = \alpha P(e_{t+1} | X_{t+1}) \sum_{x_t} P(X_{t+1} | x_t) P(x_t | e_{1:t}) \\
  \]

- $f_{1:t+1} = \alpha \text{Forward}(f_{1:t}, e_{t+1})$ where $f_{1:t} = P(X_t | e_{1:t})$

  Time and space \textit{constant} (independent of $t$)
Filtering example

Day 0:
\[ P(R_0) = <0.5,0.5> \]

Day 1 (Umbrella appears \( \Rightarrow U_1 = \text{true} \))
\[
P(R_1) = \sum_{r_0} P(R_1|r_0) P(r_0) \\
= <0.7,0.3> \times 0.5 + <0.3,0.7> \times 0.5 \\
= <0.5,0.5>
\]

updating with evidence for \( t=1 \) gives:
\[
P(R_1|u_1) = \alpha P(u_1|R_1) P(R_1) = \alpha <0.9,0.2> \times <0.5,0.5> \\
= \alpha <0.45,0.1> \approx <0.818,0.182>
\]

Day 2 (Umbrella appears \( \Rightarrow U_2 = \text{true} \))
\[
P(R_2|u_1) = \sum_{r_1} P(R_2|r_1) P(r_1|u_1) \\
= <0.7,0.3> \times 0.818 + <0.3,0.7> \times 0.182 \approx <0.627,0.373>
\]

updating with evidence for \( t=2 \) gives:
\[
P(R_2|u_1,u_2) = \alpha P(u_2|R_2) P(R_2|u_1) = \alpha <0.9,0.2> \times <0.627,0.373> \\
= \alpha <0.565,0.075> \approx <0.883,0.117>
\]
Smoothing

- Divide evidence $e_{1:t}$ into $e_{1:k}$, $e_{k+1:t}$:

  $$P(X_k | e_{1:t}) = P(X_k | e_{1:k}, e_{k+1:t})$$

  $$= \alpha P(X_k | e_{1:k}) P(e_{k+1:t} | X_k, e_{1:k}) \quad \text{Using Bayes rule}$$

  $$= \alpha P(X_k | e_{1:k}) P(e_{k+1:t} | X_k) \quad \text{Using conditional independence}$$

  $$= \alpha f_{1:k} b_{k+1:t}$$

- Backward message computed by a backwards recursion:

  $$P(e_{k+1:t} | X_k) = \sum_{x_{k+1}} P(e_{k+1:t} | x_{k+1}) P(x_{k+1} | X_k) \quad \text{Conditioning on } X_{k+1}$$

  $$= \sum_{x_{k+1}} P(e_{k+1:t} | x_{k+1}) P(x_{k+1} | X_k) \quad \text{Using conditional independence}$$

  $$= \sum_{x_{k+1}} P(e_{k+1} | x_{k+1}) P(e_{k+2:t} | x_{k+1}) P(x_{k+1} | X_k) = \text{BACKWARD}(b_{k+2:t}, e_{k+1:t})$$
Smoothing example

Compute estimate for rain at $t=1$

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

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

$$P(u_2|R_1) = \Sigma_r P(u_2|r) P(|r_2) P(r_2|R_1)$$

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

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

Smoothed estimate:

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

- **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 ≠ sequence of most likely states!!!!
- Most likely path to each $x_{t+1}$
  
  = most likely path to some $x_t$ plus one more step

\[
\max_{x_1 \ldots x_t} P(x_1, \ldots, x_t, X_{t+1} \mid e_{1:t+1})
\]

\[
= P(e_{t+1} \mid X_{t+1}) \max_{x_t} (P(X_{t+1} \mid x_t) \max_{x_1 \ldots x_{t-1}} P(x_1, \ldots, x_{t-1}, x_t \mid e_{1:t}))
\]

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

\[
m_{1:t} = \max_{x_1 \ldots x_{t-1}} P(x_1, \ldots, x_{t-1}, X_t \mid e_{1:t})
\]

- 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:

\[
m_{1:t+1} = P(e_{1:t+1} \mid X_{t+1}) \max_{x_t} \Phi(X_{t+1} \mid x_t) m_{1:t}
\]
**Viterbi Example**

The diagram illustrates a Viterbi example with states representing the weather conditions (Rain for each time step), and paths representing the sequence of states. The most likely paths are highlighted, with the probabilities associated with each transition.

- **State Space Paths**:
  - Rain\(_t\) states: true, false
  - Umbrella\(_t\) states: true, false

- **Most Likely Paths**:
  - Probabilities for transitions between states:
    - $P(R_{t-1} | R_t) = 0.7$ for true to true, 0.3 for false to true
    - $P(U_{t-1} | R_t) = 0.9$ for true to true, 0.2 for false to true

The diagram shows the most likely path through the states, with the highest probability highlighted. For example, the path $\text{Rain}_1, \text{Rain}_2, \text{Rain}_3, \text{Rain}_4, \text{Rain}_5$ with transitions $\text{true} \rightarrow \text{true} \rightarrow \text{true} \rightarrow \text{true} \rightarrow \text{true}$ is the most likely sequence, with probabilities $0.8182, 0.5155, 0.0361, 0.0334, 0.0210$.
Viterbi Example

\[
\begin{array}{cccccc}
\text{Rain}_1 & \text{Rain}_2 & \text{Rain}_3 & \text{Rain}_4 & \text{Rain}_5 \\
\text{true} & \text{true} & \text{true} & \text{true} & \text{true} \\
0.8182 \times 0.7 \times 0.9 & 0.5155 \times 0.7 \times 0.1 & 0.1237 \times 0.3 \times 0.9 & 0.0334 \times 0.7 \times 0.9 \\
\end{array}
\]
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 $U_1 = \text{true}$, $O_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}$

- Forward and backward messages as column vectors:
  \[f_{1:t+1} = O_{t+1}^T f_{1:t}\]
  \[b_{k+1:t} = T O_{k+1} b_{k+2:t}\]

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

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

- 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:

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

- 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:

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

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

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

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

- 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:

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

- Algorithm: forward pass computes \( f_t \), backward pass does \( f_i, b_i \)

![Diagram of country dance algorithm]
Country dance algorithm

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

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

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:

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

- Algorithm: forward pass computes \( f_t \), backward pass does \( f_i, b_i \)
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 $P(X_t|e_{1:t})$ 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

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

is Gaussian.

Hence $P(X_t|e_{1:t})$ is multivariate Gaussian $N(\mu_t, \Sigma_t)$ for all $t$.

**FORWARD** operator for Kalman filters takes a Gaussian forward message $f_{1:t}$ specified by $N(\mu_t, \Sigma_t)$ and produces $N(\mu_{t+1}, \Sigma_{t+1})$. 
Simple 1-D example

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

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

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

First observation
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
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

\( X_t, E_t \) contain arbitrarily many variables in a replicated Bayes net.

\[
\begin{array}{|c|c|}
\hline
R_0 & P(R_1) \\
\hline
0.7 & 0.7 \\
0.3 & \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
R_1 & P(U_1) \\
\hline
0.9 & \\
0.2 & \\
\hline
\end{array}
\]
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 $t$: $N(x_t|e_{1:t})/N = P(x_t|e_{1:t})$

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

$$N(x_{t+1}|e_{1:t}) = x_tP(x_{t+1}|x_t)N(x_t|e_{1:t})$$

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 & 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
- 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