Lecture 8

Graphical Models for Sequential Data

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Slides by Stuart Russell and Peter Norvig

Course Overview

- ✔ Introduction
  - ✔ Artificial Intelligence
  - ✔ Intelligent Agents
- ✔ Search
  - ✔ Uninformed Search
  - ✔ Heuristic Search
- Uncertain knowledge and Reasoning
  - ✔ Probability and Bayesian approach
  - ✔ Bayesian Networks
  - Hidden Markov Chains
  - Kalman Filters
- Learning
  - Supervised
    Learning Bayesian Networks, Neural Networks
  - Unsupervised
    EM Algorithm
- Reinforcement Learning
- Games and Adversarial Search
  - Minimax search and Alpha-beta pruning
  - Multiagent search
- Knowledge representation and Reasoning
  - Propositional logic
  - First order logic
  - Inference
  - Plannning

Outline

Outline

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

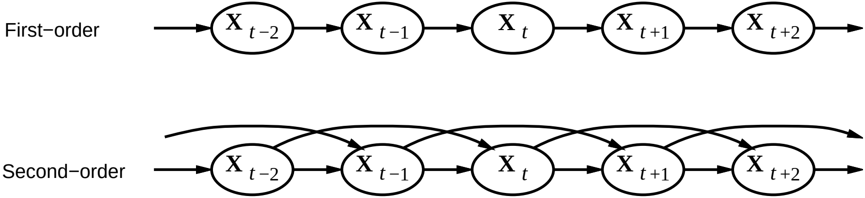
Time and uncertainty

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

Markov processes (Markov chains)

Construct a Bayes net from these variables:
- unbounded number of conditional probability table
- unbounded number of parents

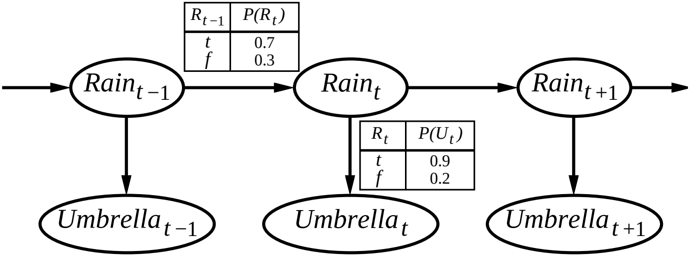
Markov assumption: $\mathbf{X}_t$ depends on **bounded** subset of $\mathbf{X}_{0:t-1}$
First-order Markov process: $\Pr(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \Pr(\mathbf{X}_t|\mathbf{X}_{t-1})$
Second-order Markov process: $\Pr(\mathbf{X}_t|\mathbf{X}_{0:t-1}) = \Pr(\mathbf{X}_t|\mathbf{X}_{t-2}, \mathbf{X}_{t-1})$



Sensor Markov assumption: $\Pr(\mathbf{E}_t|\mathbf{X}_{0:t}, \mathbf{E}_{0:t-1}) = \Pr(\mathbf{E}_t|\mathbf{X}_t)$
$\rightsquigarrow$Stationary process:

- transition model $\Pr(\mathbf{X}_t|\mathbf{X}_{t-1})$ and
- sensor model $\Pr(\mathbf{E}_t|\mathbf{X}_t)$ 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

1. Filtering: $\Pr(\mathbf{X}_t|\mathbf{e}_{1:t})$
   belief state—input to the decision process of a rational agent

2. Prediction: $\Pr(\mathbf{X}_{t+k}|\mathbf{e}_{1:t})$ for $k > 0$
   evaluation of possible action sequences;
   like filtering without the evidence

3. Smoothing: $\Pr(\mathbf{X}_k|\mathbf{e}_{1:t})$ for $0 \leq k < t$
   better estimate of past states, essential for learning

4. 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:

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

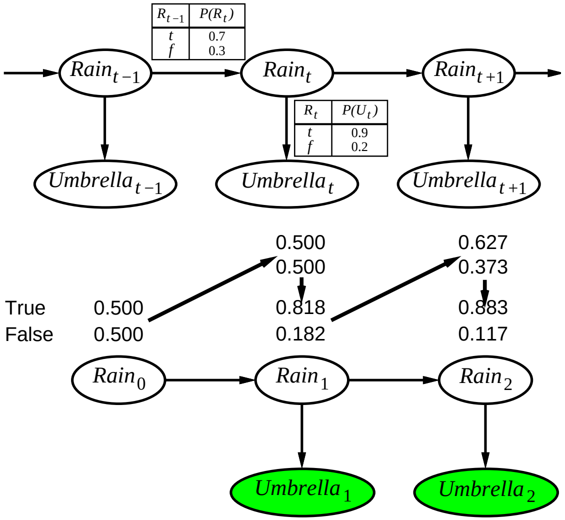
$$\begin{aligned}\Pr(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) &= \Pr(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}, \mathbf{e}_{t+1}) \\ &= \alpha \Pr(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}, \mathbf{e}_{1:t}) \Pr(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) \\ &= \alpha \Pr(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \Pr(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})\end{aligned}$$

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

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

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

Filtering example



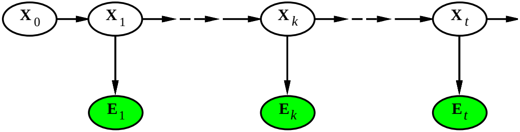
Prediction

$\Pr(\mathbf{X}_{t+k+1}|\mathbf{e}_{1:t}) = \sum_{\mathbf{x}_{t+k}} \Pr(\mathbf{X}_{t+k+1}|\mathbf{x}_{t+k})P(\mathbf{x}_{t+k}|\mathbf{e}_{1:t})$

As $k \to \infty$, $P(\mathbf{x}_{t+k}|\mathbf{e}_{1:t})$ tends to the **stationary distribution** of the Markov chain

Mixing time depends on how **stochastic** the chain is

Smoothing



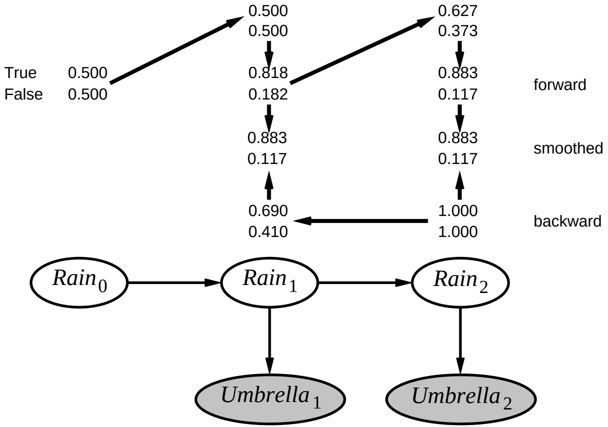
Divide evidence $\mathbf{e}_{1:t}$ into $\mathbf{e}_{1:k}$, $\mathbf{e}_{k+1:t}$:

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

Backward message computed by a backwards recursion:

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

Smoothing example



If we want to smooth the whole sequence:

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 (joint distr.)!
Most likely path to each $\mathbf{x}_{t+1}$
= most likely path to **some** $\mathbf{x}_t$ plus one more step

$$\max_{\mathbf{x_1} \ldots \mathbf{x_t}} \Pr(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathbf{X}_{t+1} | \mathbf{e}_{1:t+1})$$

$$= \Pr(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \max_{\mathbf{x_t}} \left(\Pr(\mathbf{X}_{t+1} | \mathbf{x}_t) \max_{\mathbf{x_1} \ldots \mathbf{x_{t-1}}} P(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \mathbf{x}_t | \mathbf{e}_{1:t}) \right)$$

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

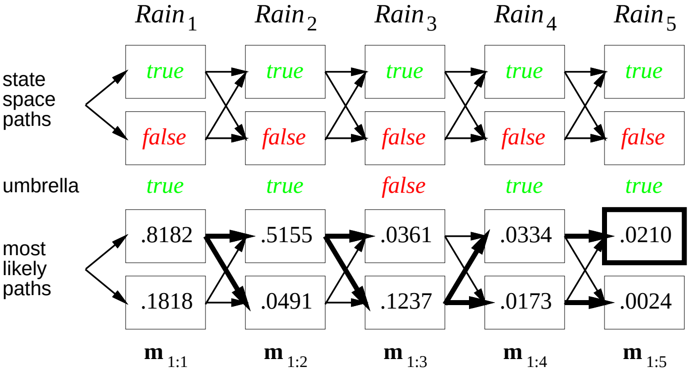
$$\mathbf{m}_{1:t} = \max_{\mathbf{x_1} \ldots \mathbf{x_{t-1}}} \Pr(\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \mathbf{X}_t | \mathbf{e}_{1:t}),$$

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:

$$\mathbf{m}_{1:t+1} = \Pr(\mathbf{e}_{t+1} | \mathbf{X}_{t+1}) \max_{\mathbf{x_t}} (\Pr(\mathbf{X}_{t+1} | \mathbf{x}_t) \mathbf{m}_{1:t})$$

Viterbi example



Hidden Markov models

$\mathbf{X}_t$ is a single, discrete variable (usually $\mathbf{E}_t$ is too)
Domain of X_t is $\{1, \ldots, S\}$ – can be a macro variable representing several state vars.

HMMs allow for an elegant matrix representation

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 convenience) for each time step, diagonal elements $P(e_t | X_t = i)$
e.g., for $U_1 = true$, $\mathbf{O}_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix}$

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 $O(St)$

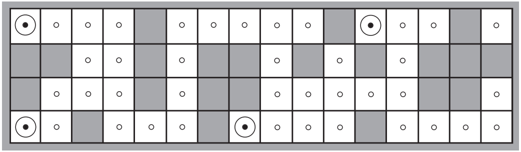
Real HMM examples

- Speech recognition HMMs:
  Observations are acoustic signals (continuous valued) States are specific positions in specific words (so, tens of thousands)

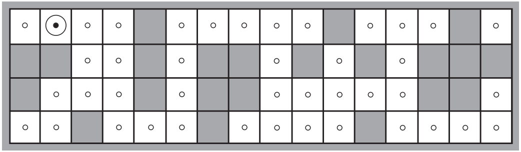
- Machine translation HMMs:
  Observations are words (tens of thousands) States are translation options

- Robot tracking:
  Observations are features of environment (discrete) or range readings (continuous) States are cells (discrete) or positions on a map (continuous)

Localization

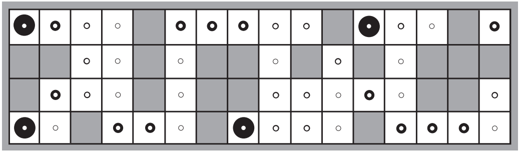


(a) Possible locations of robot after $E_1 = NSW$

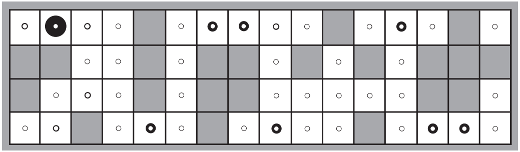


(b) Possible locations of robot After $E_1 = NSW, E_2 = NS$

Localization



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



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

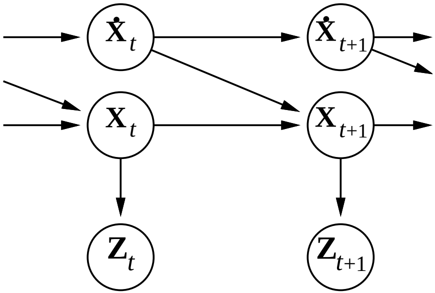
- $\Pr(X_0 = i) = 1/n$
- $\Pr(X_{t+1} = j \mid X_t = i) = \mathbf{T}_{ij} = \begin{cases} 1/N(i) & \text{if } i \text{ is adjacent to } j \\ 0 & \text{otherwise} \end{cases}$
- $\Pr(E_t = e_t \mid X_t = i) = \mathbf{O}_{ti} = (1 - \epsilon)^{4 - d_{it}} \epsilon^{d_{it}}$

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

$$\Pr(\mathbf{X}_{t+1}|\mathbf{e}_{1:t}) = \int_{\mathbf{x_t}} \Pr(\mathbf{X}_{t+1}|\mathbf{x}_t)P(\mathbf{x}_t|\mathbf{e}_{1:t})\, d\mathbf{x}_t$$

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

$$\Pr(\mathbf{X}_{t+1}|\mathbf{e}_{1:t+1}) = \alpha \Pr(\mathbf{e}_{t+1}|\mathbf{X}_{t+1}) \Pr(\mathbf{X}_{t+1}|\mathbf{e}_{1:t})$$

is Gaussian

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

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

General Kalman update

Transition and sensor models:

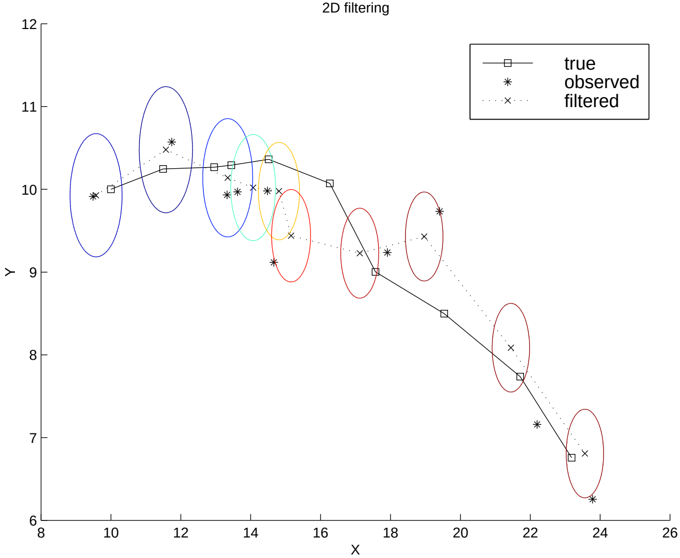
$$\begin{aligned} P(\mathbf{x}_{t+1}|\mathbf{x}_t) &= N(\mathbf{F}\mathbf{x}_t, \mathbf{\Sigma}_x)(\mathbf{x}_{t+1}) \\ P(\mathbf{z}_t|\mathbf{x}_t) &= N(\mathbf{H}\mathbf{x}_t, \mathbf{\Sigma}_z)(\mathbf{z}_t) \end{aligned}$$

$\mathbf{F}$ 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:

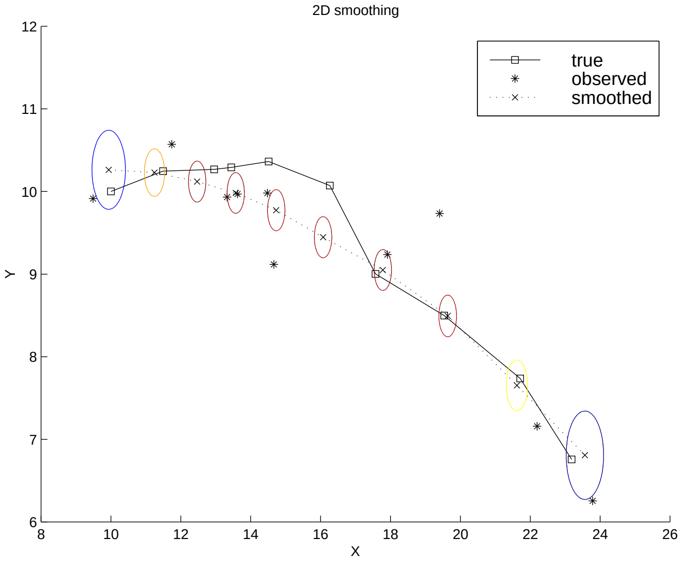
$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \mathbf{F}\boldsymbol{\mu}_t + \mathbf{K}_{t+1}(\mathbf{z}_{t+1} - \mathbf{H}\mathbf{F}\boldsymbol{\mu}_t) \\ \mathbf{\Sigma}_{t+1} &= (\mathbf{I} - \mathbf{K}_{t+1})(\mathbf{F}\mathbf{\Sigma}_t\mathbf{F}^\top + \mathbf{\Sigma}_x) \end{aligned}$$

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

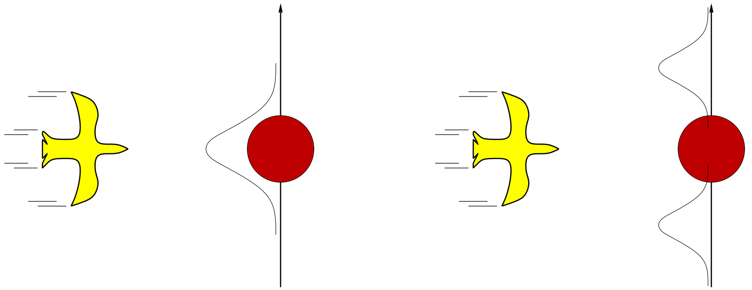


2-D tracking example: smoothing



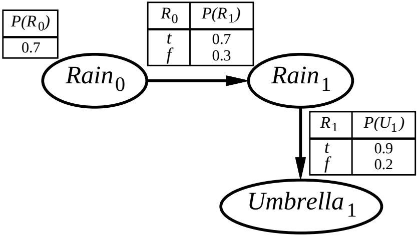
Where it breaks

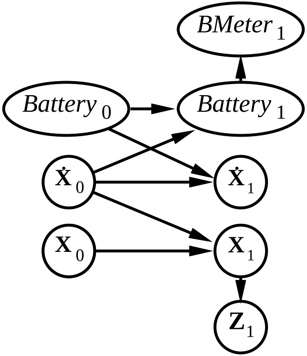
Cannot be applied if the transition model is nonlinear
Extended Kalman Filter models transition as **locally linear** around $\mathbf{x}_t = \boldsymbol{\mu}_t$
Fails if systems is locally unsmooth



Dynamic Bayesian networks

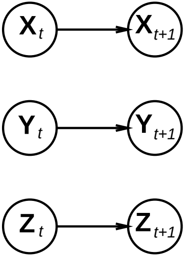
$\mathbf{X}_t$, $\mathbf{E}_t$ contain arbitrarily many variables in a replicated Bayes net

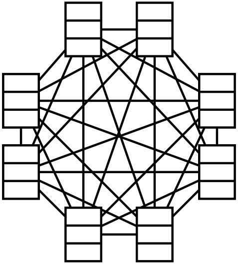




DBNs vs. HMMs

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





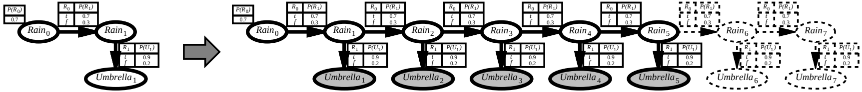
Sparse dependencies $\Rightarrow$ exponentially fewer parameters;
e.g., 20 state variables, three parents each
DBN has $20 \times 2^3 = 160$ parameters, HMM has $2^{20} \times 2^{20} \approx 10^{12}$

DBNs vs Kalman filters

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

Exact inference in DBNs

Naive method: unroll the network and run any exact algorithm



Problem: inference cost for each update grows with t

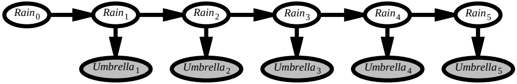
Rollup filtering: add slice $t+1$, "sum out" slice t using variable elimination

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



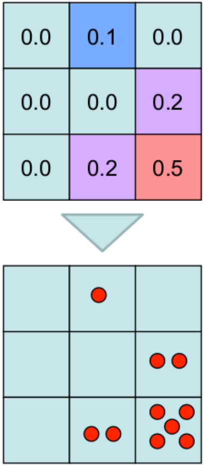
LW samples pay no attention to the evidence!

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

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

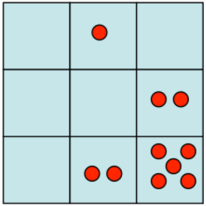
Particle filtering

- Sometimes |X| is too big to use exact inference
  - |X| may be too big to even store B(X)
  - E.g. X is continuous
  - $|X|^2$ may be too big to do updates
- Solution: approximate inference
  - Track samples of X, not all values
  - Samples are called particles
  - Time per step is linear in the number of samples
  - But: number needed may be large
- This is how robot localization works in practice



Particle filtering

- Our representation of P(X) is now a list of N particles (samples)
  - Generally, N << |X|
  - Storing map from X to counts would defeat the point
- P(x) approximated by number of particles with value x
  - So, many x will have P(x) = 0
  - More particles, more accuracy
- Initially, all particles have a weight of 1



Particles:
(3,3)
(2,3)
(3,3)
(3,2)
(3,3)
(3,2)
(2,1)
(3,3)
(3,3)
(2,1)

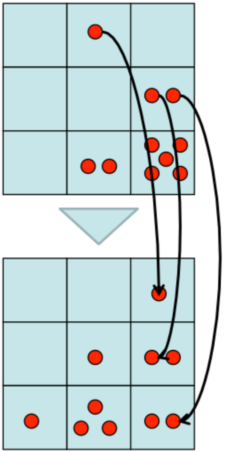
Particle filtering

- Each particle is moved by sampling its next position from the transition model

$$x' = \text{sample}(P(X'|x))$$

  - This is like prior sampling – samples' frequencies reflect the transition probs
  - Here, most samples move clockwise, but some move in another direction or stay in place

- This captures the passage of time
  - If we have enough samples, close to the exact values before and after (consistent)



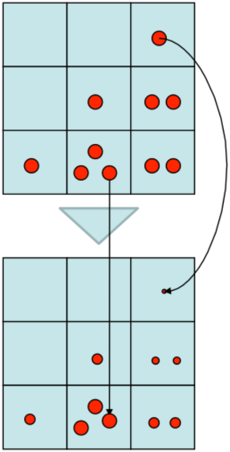
Particle filtering

- Slightly trickier:
  - Don't do rejection sampling (why not?)
  - We don't sample the observation, we fix it
  - As in likelihood weighting, downweight samples based on the evidence:

$$w(x) = P(e|x)$$

$$B(X) \propto P(e|X)B'(X)$$

  - Note that, as before, the probabilities don't sum to one, since most have been downweighted (in fact they sum to an approximation of P(e))

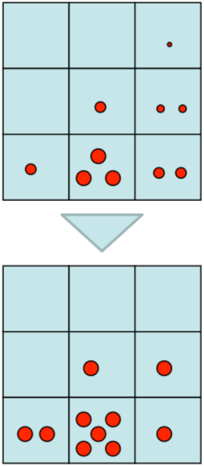


Particle filtering

- Rather than tracking weighted samples, we resample
- N times, we choose from our weighted sample distribution (i.e. draw with replacement)
- This is analogous to renormalizing the distribution
- Now the update is complete for this time step, continue with the next one

Old Particles:
(3,3) w=0.1
(2,1) w=0.9
(2,1) w=0.9
(3,1) w=0.4
(3,2) w=0.3
(2,2) w=0.4
(1,1) w=0.4
(3,1) w=0.4
(2,1) w=0.9
(3,2) w=0.3

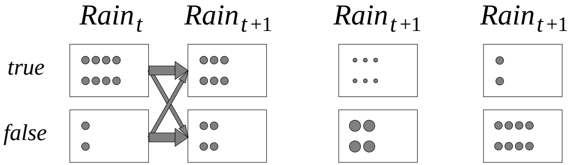
New Particles:
(2,1) w=1
(2,1) w=1
(2,1) w=1
(3,2) w=1
(2,2) w=1
(2,1) w=1
(1,1) w=1
(3,1) w=1
(2,1) w=1
(1,1) w=1



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

Summary

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