BELIEF NETWORKS II

LECTURE 24
(CHapter 15.3 4 + new)
ARTIFICIAL INTELLIGENCE I
AUTUMN 2001
HENRY KAUTZ

Inference tasks

Causal:
Given burglary, what is probability John calls?

Diagnostic:
Given John calls, what is probability of earthquake?

Mixed:
Given John calls and there is an earthquake, what is the probability of burglary?

Most Probable Explanation:
Given earthquake, what is the most likely simultaneous setting of all of the other variables?

Inference by enumeration

Slightly intelligent way to sum out variables from the joint without actually constructing its explicit representation

Simple query on the burglary network:

\[ P(B | J, M) \]

\[ = \frac{P(B, J, M)}{P(J, M)} \]

\[ = \alpha P(B, J, M) \]

\[ = \alpha \sum_a \sum_e P(B, e, a, J, M) \]

Rewrite full joint entries using product of CPT entries:

\[ P(B | J, M) \]

\[ = \alpha \sum_a \sum_e P(B) P(e) P(a | B, e) P(J | a) P(M | a) \]

\[ = \alpha P(B) \sum_e P(e) \sum_a P(a | B, e) P(J | a) P(M | a) \]
Enumeration algorithm

Exhaustive depth-first enumeration: \( O(n) \) space, \( O(d^n) \) time

```
ENUMERATIONASK(X,e,bs) returns a distribution over X
inputs: X, the query variable
        e, evidence specified as an event
        bs, a belief network specifying joint distribution \( P(X_1,\ldots,X_n) \)
Q(X) --> a distribution over X
for each value \( x_i \) of X do
    extend e with value \( x_i \) for X
Q(x) = ENUMERATEALL(VARS[bs],e)
return NORMALIZE(Q(X))
```

ENUMERATEALL(vare,e) returns a real number
if EMPTY(vare) then return 1.0
else do
    \( Y = \text{FIRST}(vare) \)
    if \( Y \) has value \( y \) in \( e \)
        then return \( P(y \mid Pa(Y)) \times \text{ENUMERATEALL(REST}(vare),e) \)
    else return \( \sum_{y} P(y \mid Pa(Y)) \times \text{ENUMERATEALL(REST}(vare),e) \)
    where \( e_y \) is \( e \) extended with \( Y = y \)
```

Inference by variable elimination

Enumeration is inefficient: repeated computation
e.g., computes \( P(J \mid a) P(M \mid a) \) for each value of \( e \)

Variable elimination: carry out summations right-to-left,
storing intermediate results (factors) to avoid recomputation

\[
\begin{align*}
P(B \mid J, M) &= \alpha \sum_B P(B) \sum_e P(e) \sum_a P(a \mid B, e) \sum_a P(J \mid a) P(M \mid a) \\
&= \alpha \sum_B P(B) \sum_e P(e) \sum_a P(a \mid B, e) \sum_a P(J = \text{true} \mid a) f_M(a) \\
&= \alpha \sum_B P(B) \sum_e P(e) \sum_a f_A(a, b, e) f_j(a) f_M(a) \\
&= \alpha \sum_B P(B) f_{EAM}(b, e) (\text{sum out } A) \\
&= \alpha \sum_B P(B) f_{EAM}(b) (\text{sum out } E) \\
&= \alpha f_B(b) \times f_{EAM}(b)
\end{align*}
\]

A form of dynamic programming. Can also be implemented using message passing
of intermediate results.

Variable elimination algorithm

```
function ELIMINATIONASK(X,e,bs) returns a distribution over X
inputs: X, the query variable
        e, evidence specified as an event
        bs, a belief network specifying joint distribution \( P(X_1,\ldots,X_n) \)
if X \in e then return observed point distribution \( \delta_X \)
factors = []; vars = REVERSE VARS[bs]
for each \( \text{var} \in \text{vars} \) do
    factors = [MAKEFACTOR(var, e)[factors]
    if var is a hidden variable then factors = SUMOUT(var, factors)
return NORMALIZE(POINTWISEPRODUCT(factors))
```

Variable elimination: Basic operations

Pointwise product of factors \( f_1 \) and \( f_2 \):
\[
f_1(x_1,\ldots,x_j, y_1,\ldots,y_k, z_1,\ldots,z_l) \\
= f(x_1,\ldots,x_j, y_1,\ldots,y_k, z_1,\ldots,z_l)
\]

E.g., \( f_1(a, b) \times f_2(b, c) = f(a, b, c) \)

Summing out a variable from a product of factors: move any constant factors outside the summation:
\[
\sum_x f_1 \times \cdots \times f_k = f_1 \times \cdots \times f_i \sum_x f_{i+1} \times \cdots \times f_k = f_1 \times \cdots \times f_i \times f_X
\]

assuming \( f_1,\ldots,f_i \) do not depend on \( X \)
Complexity of Bayes net inference

Singly connected networks (or polytrees):
- any two nodes are connected by at most one (undirected) path
- time and space cost of variable elimination are $O(d^n)$

Multiply connected networks:
- can reduce 3SAT to exact inference $\Rightarrow$ NP-hard
- equivalent to counting 3SAT models $\Rightarrow$ #P-complete

1. $A \lor B \lor C$
2. $C \lor D \lor \neg A$
3. $B \lor C \lor \neg D$

Inference by stochastic simulation

Basic idea:
1) Draw $N$ samples from a sampling distribution $S$
2) Compute an approximate posterior probability $\hat{P}$
3) Show this converges to the true probability $P$

Outline:
- Sampling from an empty network
- Rejection sampling: reject samples disagreeing with evidence
- Likelihood weighting: use evidence to weight samples
- MCMC: sample from a stochastic process whose stationary distribution is the true posterior

Sampling from an empty network

```plaintext
function PriorSample(ns) returns an event sampled from $P(X_1, \ldots, X_n)$ specified by ns
for i = 1 to n do
    $x_i$ = a random sample from $P(X_i | Parents(X_i))$
return $x$
```

$P(Cloudy) = (0.5, 0.5)$
sample → true

$P(Sprinkler|Cloudy) = (0.1, 0.9)$
sample → false

$P(Rain|Cloudy) = (0.8, 0.2)$
sample → true

$P(WetGrass|\neg Sprinkler, Rain) = (0.9, 0.1)$
sample → true

Sampling from an empty network contd.

Probability that PriorSample generates a particular event

$S_{PS}(x_1 \ldots x_n) = \prod_{i=1}^{n} P(x_i | Parents(X_i)) = P(x_1 \ldots x_n)$
i.e., the true prior probability

Let $N_{PS}(Y = y)$ be the number of samples generated for which $Y = y$, for any set of variables $Y$.

Then $\hat{P}(Y = y) = \frac{N_{PS}(Y = y)}{N}$ and

$$\lim_{N \to \infty} \hat{P}(Y = y) = \sum_h S_{PS}(Y = y, H = h) = \sum_h P(Y = y, H = h) = P(Y = y)$$

That is, estimates derived from PriorSample are consistent
Rejection sampling

\[ \hat{P}(X|e) \] estimated from samples agreeing with \( e \)

\begin{algorithm}
\textbf{function} REJECTION-SAMPLING \((X, e, hN)\) \textbf{returns} an approximation to \( P(X|e) \)
\begin{algorithmic}
\State \( N(X) \) \text{ a vector of counts over } X, \text{ initially zero}
\For{\( j = 1 \) to \( N \)}
\State \( x \leftarrow \text{PRIOR}\text{SAMPLE}(h) \)
\If{\( x \) is consistent with \( e \)}
\State \( N[x] \leftarrow N[x] + 1 \) \text{ where } \( x \) \text{ is the value of } X \text{ in } x 
\EndIf
\EndFor
\State return \text{NORMALIZE}(N[X])
\end{algorithmic}
\end{algorithm}

E.g., estimate \( \hat{P}(\text{Rain}|\text{Sprinkler} = \text{true}) \) using 100 samples
27 samples have \( \text{Sprinkler} = \text{true} \)
Of these, 8 have \( \text{Rain} = \text{true} \) and 19 have \( \text{Rain} = \text{false} \).
\( \hat{P}(\text{Rain}|\text{Sprinkler} = \text{true}) = \text{NORMALIZE}((8, 19)) = (0.296, 0.704) \)
Similar to a basic real-world empirical estimation procedure

Analysis of rejection sampling

\[ \hat{P}(X|e) = \alpha N_{PS}(X, e) \quad \text{(algorithm defn.)} \]
\[ = N_{PS}(X, e)/N_{PS}(e) \quad \text{(normalized by } N_{PS}(e)) \]
\[ \approx \hat{P}(X, e)/P(e) \quad \text{(property of PRIORSAMPLE)} \]
\[ = P(X|e) \quad \text{(defn. of conditional probability)} \]

Hence rejection sampling returns consistent posterior estimates
Problem: hopelessly expensive if \( P(e) \) is small

Likelihood weighting

Idea: fix evidence variables, sample only nonevidence variables, and weight each sample by the likelihood it accords the evidence

\begin{algorithm}
\textbf{function} WEIGHTEDSAMPLE \((h, e)\) \textbf{returns} an event and a weight
\begin{algorithmic}
\State \( x \leftarrow \text{an event with } n \text{ elements}; w \leftarrow 1 \)
\For{\( i = 1 \) to \( n \)}
\If{\( X_i \) has a value \( x_i \) in \( e \)}
\State \( w \leftarrow w \times P(X_i = x_i | \text{Parents}(X_i)) \)
\EndIf
\EndFor
\State return \( x, w \)
\end{algorithmic}
\end{algorithm}

\begin{algorithm}
\textbf{function} LIKELIHOODWEIGHTING \((X, h, N)\) \textbf{returns} an approximation to \( P(X|e) \)
\begin{algorithmic}
\State \( W[X] \) \text{ a vector of weighted counts over } X, \text{ initially zero}
\For{\( j = 1 \) to \( N \)}
\State \( x \leftarrow \text{WEIGHTEDSAMPLE}(h) \)
\State \( W[x] \leftarrow W[x] + w \) \text{ where } \( x \) \text{ is the value of } X \text{ in } x 
\EndFor
\State return \text{NORMALIZE}(W[X])
\end{algorithmic}
\end{algorithm}

Likelihood weighting example

Estimate \( \hat{P}(\text{Rain}|\text{Sprinkler} = \text{true}, \text{WetGrass} = \text{true}) \)
Sample generation process:
1. \( w \leftarrow 1.0 \)
2. Sample \( P(C\text{loudy}) = (0.5, 0.5) \); say true
3. \( \text{Sprinkler has value true, so} \)
   \( w \leftarrow w \times P(\text{Sprinkler = true} | C\text{loudy = true}) = 0.1 \)
4. Sample \( P(\text{Rain} | C\text{loudy = true}) = (0.8, 0.2) \); say true
5. \( \text{WetGrass has value true, so} \)
   \( w \leftarrow w \times P(\text{WetGrass = true} | \text{Sprinkler = true, Rain = true}) = 0.099 \)

**Likelihood weighting analysis**

Sampling probability for \texttt{WeightedSample} is
\[
S_{WS}(y, e) = \prod_{i=1}^{n} P(y_i | \text{Parents}(e_i))
\]
Note: pays attention to evidence in ancestors only
\[ \Rightarrow \text{somewhere "in between" prior and posterior distribution} \]
Weight for a given sample \( y, e \) is
\[
w(y, e) = \prod_{i=1}^{n} P(e_i | \text{Parents}(e_i))
\]
Weighted sampling probability is
\[
S_{WS}(y, e)w(y, e) = \prod_{i=1}^{n} P(y_i | \text{Parents}(e_i)) \prod_{i=1}^{n} P(e_i | \text{Parents}(e_i)) = P(y, e) \text{ (by standard global semantics of network)}
\]
Hence likelihood weighting returns consistent estimates
but performance still degrades with many evidence variables

**Approximate inference using MCMC**

"State" of network = current assignment to all variables
Generate next state by sampling one variable given Markov blanket
Sample each variable in turn, keeping evidence fixed

```function MCMC-ASK(X, e, bs, N) returns an approximation to P(X|e)
local variables: N[X], a vector of counts over X, initially zero
Y, the nonevidence variables in bs
x, the current state of the network, initially copied from e
initialize x with random values for the variables in Y
for j = 1 to N do
    N[x] ← N[x] + 1 where x is the value of X in x
    for each y_j in Y do
        sample the value of y_j in x from P(y_j|MB(y_j)) given the values of MB(y_j) in x
    end
return NORMALIZE(N[X])
```

Approaches stationary distribution: long-run fraction of time spent in each state is exactly proportional to its posterior probability

**MCMC Example**

Estimate \( P(\text{Rain} | \text{Sprinkler = true, WetGrass = true}) \)
Sample \texttt{Cloudy} then \texttt{Rain}, repeat.
Count number of times \texttt{Rain} is true and false in the samples.
Markov blanket of \texttt{Cloudy} is \texttt{Sprinkler} and \texttt{Rain}
Markov blanket of *Rain* is *Cloudy*, *Sprinkler*, and *WetGrass*

![Diagram showing the Markov blanket of Rain]

**MCMC example cont'd.**

Random initial state: *Cloudy = true* and *Rain = false*

1. \( P(\text{Cloudy}|MB(\text{Cloudy})) = P(\text{Cloudy}|\neg\text{Sprinkler}, \neg\text{Rain}) \)
   sample \( \to \) false

2. \( P(\text{Rain}|MB(\text{Rain})) = P(\text{Rain}|\neg\text{Cloudy}, \text{Sprinkler}, \text{WetGrass}) \)
   sample \( \to \) true

Visit 100 states
   - 31 have *Rain = true*, 69 have *Rain = false*

\[ \hat{P}(\text{Rain}|\text{Sprinkler} = \text{true}, \text{WetGrass} = \text{true}) = \text{NORMALIZE}(31, 69) = (0.31, 0.69) \]

**MCMC analysis: Outline**

- Transition probability \( q(y \to y') \)
- Occupancy probability \( \pi_t(y) \) at time \( t \)
- Equilibrium condition on \( \pi_t \) defines stationary distribution \( \pi(y) \)
  
  Note: stationary distribution depends on choice of \( q(y \to y') \)

- Pairwise detailed balance on states guarantees equilibrium

- **Gibbs sampling** transition probability:
  - sample each variable given current values of all others
  - \( \Rightarrow \) detailed balance with the true posterior

  For Bayesian networks, Gibbs sampling reduces to sampling conditioned on each variable’s Markov blanket

**Stationary distribution**

- \( \pi_t(y) = \text{probability in state } y \text{ at time } t \)
- \( \pi_{t+1}(y') = \text{probability in state } y' \text{ at time } t + 1 \)

- \( \pi_{t+1} \) in terms of \( \pi_t \) and \( q(y \to y') \)
  \[ \pi_{t+1}(y') = \sum_y \pi_t(y) q(y \to y') \]

- Stationary distribution: \( \pi = \pi_{t+1} = \pi \)
  \[ \pi(y') = \sum_y \pi(y) q(y \to y') \quad \text{for all } y' \]

  If \( \pi \) exists, it is unique (specific to \( q(y \to y') \))

  In equilibrium, expected "outflow" = expected "inflow"
Detailed balance

"Outflow" = "inflow" for each pair of states:

\[ \pi(y) q(y \rightarrow y') = \pi(y') q(y' \rightarrow y) \quad \text{for all } y, y' \]

Detailed balance \( \Rightarrow \) stationarity:

\[ \sum_y \pi(y) q(y \rightarrow y') = \sum_{y'} \pi(y') q(y' \rightarrow y) \]
\[ = \pi(y') \sum_y q(y' \rightarrow y) \]
\[ = \pi(y') \]

MCMC algorithms typically constructed by designing a transition probability \( q \) that is in detailed balance with desired \( \pi \)

Gibbs sampling

Sample each variable in turn, given all other variables

Sampling \( Y_i \), let \( Y_j \) be all other nonevidence variables

Current values are \( y_i \) and \( y_j \); \( e \) is fixed

Transition probability is given by

\[ q(y \rightarrow y') = q(y_i, y_j \rightarrow y_i', y_j) = P(y'_i | y_j, e) \]

This gives detailed balance with true posterior \( P(y|e) \):

\[ \pi(y) q(y \rightarrow y') = P(y|e) P(y'_i | y_j, e) = P(y_i | y_j, e) P(y'_i | y_j, e) \]
\[ = P(y_i | y_j, e) P(y_i | e) P(y'_i | y_j, e) \quad \text{(chain rule)} \]
\[ = P(y_i | y_j, e) P(y'_i | y_j, e) \quad \text{(chain rule backwards)} \]
\[ = q(y' \rightarrow y) \pi(y') = \pi(y') q(y' \rightarrow y) \]

Markov blanket sampling

A variable is independent of all others given its Markov blanket:

\[ P(y'_i | y_j, e) = P(y'_i | MB(Y_i)) \]

Probability given the Markov blanket is calculated as follows:

\[ P(y'_i | MB(Y_i)) = P(y'_i | \text{Parents}(Y_i)) \prod_{Z_j \in \text{Children}(Y_i)} P(z_j | \text{Parents}(Z_j)) \]

Hence computing the sampling distribution over \( Y_i \) for each flip requires just \( cd \) multiplications if \( Y_i \) has \( c \) children and \( d \) values; can cache it if \( c \) not too large.

Main computational problems:

1) Difficult to tell if convergence has been achieved
2) Can be wasteful if Markov blanket is large:

\[ P(Y_i | MB(Y_i)) \] won’t change much (law of large numbers)

Performance of approximation algorithms

Absolute approximation: \[ |P(X|e) - \hat{P}(X|e)| \leq \epsilon \]

Relative approximation: \[ \frac{|P(X|e) - \hat{P}(X|e)|}{P(X|e)} \leq \epsilon \]

Relative \( \Rightarrow \) absolute since \( 0 \leq P \leq 1 \) (may be \( O(2^n) \))

Randomized algorithms may fail with probability at most \( \delta \)

Polynomial approximation: \( \text{poly}(n, \epsilon^{-1}, \log \delta^{-1}) \)

Theorem (Dagum and Luby, 1993): both absolute and relative approximation for either deterministic or randomized algorithms are NP-hard for any \( \epsilon, \delta < 0.5 \)

(Absolute approximation polynomial with no evidence—Chernoff bounds)