Lecture 16: March 21, 2015
cs 573: Probabilistic Reasoning

Professor Nevatia

Spring 2015

Review

- HW #5 due Mar 30;
- Exam 1, please return with or without comments
- Exam 2, April 29, class period; NOT cumulative
- Previous Lecture
  - Variational approximation
- Today's objective
  - Review and complete Variational Approximation discussion
  - Sampling approach

Variational Approach

- Treats inference problem as an optimization problem
- Approximate the actual distribution, say P, with a simpler distribution, say Q
  - e.g. fit a 1-D Gaussian to a 1-D arbitrary distribution
  - Fit a multi-variate Gaussian but assume co-variances are nil
  - Assume Q is a product of individual variable distributions
  - Need to select a tractable Q
- How to measure difference between P and Q?
- How to choose parameters of Q to minimize the difference between P and Q?

Finding the Minimum

- We choose to minimize $D(Q||P_\Phi) = -H_Q(X) - E_Q[\ln P_\Phi(X)]$

  Equivalent to maximizing "free energy" functional

  $F[\tilde{}P_\Phi, Q] = H_Q(X) + \sum_{\phi\varepsilon\Phi} E_Q[\ln \phi(U_\phi)]$ Find Q that minimizes
- Choose $Q(\boldsymbol{X}) = \prod_i Q(X_i)$; also $\sum_i Q(X_i) = 1$
- Take derivatives of the Lagrangian with respect to $Q(x_i)$ and set = 0
- Leads to a "Mean-Field" approximation

$$Q(x_i) = \frac{1}{Z_i} \exp\left\{ \boldsymbol{E}_{\boldsymbol{X}_{-i}\sim Q}[\ln P_\Phi(x_i \mid \boldsymbol{X}_{-i})] \right\}$$

- After further simplification:

$$Q(x_i) = \frac{1}{Z_i} \exp\left\{ \sum_{\phi : X_i \in Scope[\phi]} \boldsymbol{E}_{(\boldsymbol{U}_\phi - \{X_i\})\sim Q}[\ln \phi(\boldsymbol{U}_\phi, x_i)] \right\}.$$

where Z_i is a normalizing constant.

Explaining the Notation

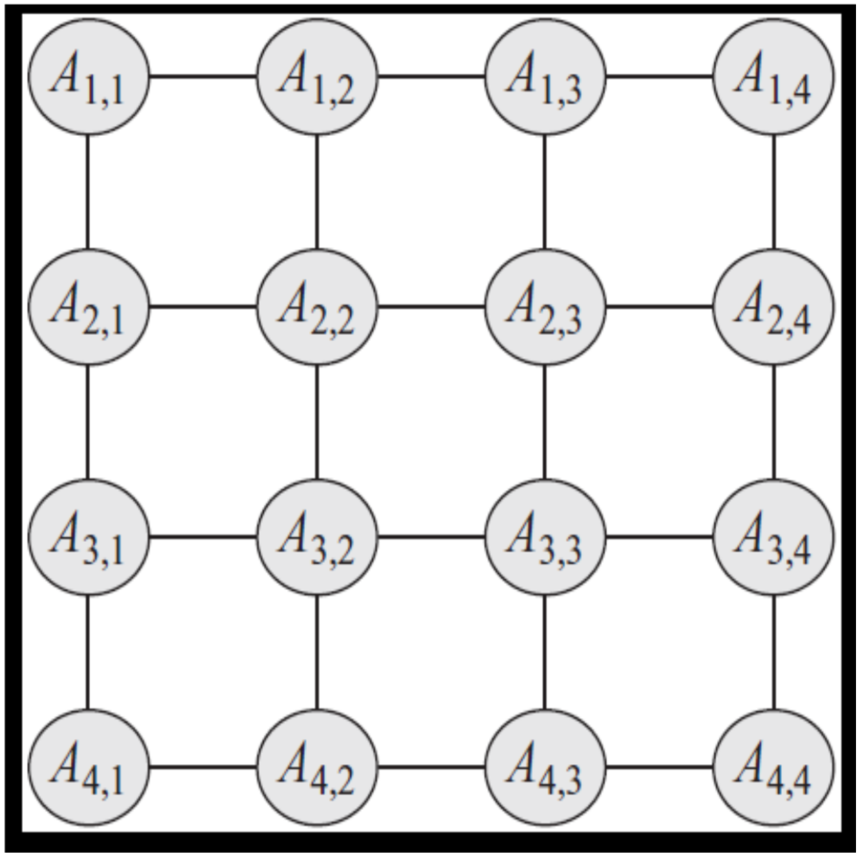
- Start with a clique of just two variables (X_1,X_2), given $\phi(X_1,X_2)$
- For X_1 : $Q(X_1=x_1^i) = 1/z_1 \exp (\sum_{x2} Q(x_2) \log \phi(x_1^i,x_2))$
  - Sum is over all possible values of X_2
  - Note above updates the distribution $Q(X_1)$
  - Similar expression for $Q(X_2)$ in terms of $Q(X_1)$; iterate
- Suppose we have a clique (X_1, X_2, X_3)

  For X_1 : $Q(X_1=x_1^i) = 1/z_1 \exp (\sum_{x2,x3} Q(x_2,x_3) \log \phi(x_1^i,x_2,x_3))$

  $= 1/z_1 \exp (\sum_{x2,x3} Q(x_2)Q(x_3) \log \phi(x_1^i,x_2,x_3))$

- If X_1 is also in another clique, say (X_1, X_4), add terms corresponding to $\sum_{x4} Q(x_4) \log \phi(x_1,x_4)$ to the above summation

GRID MRF



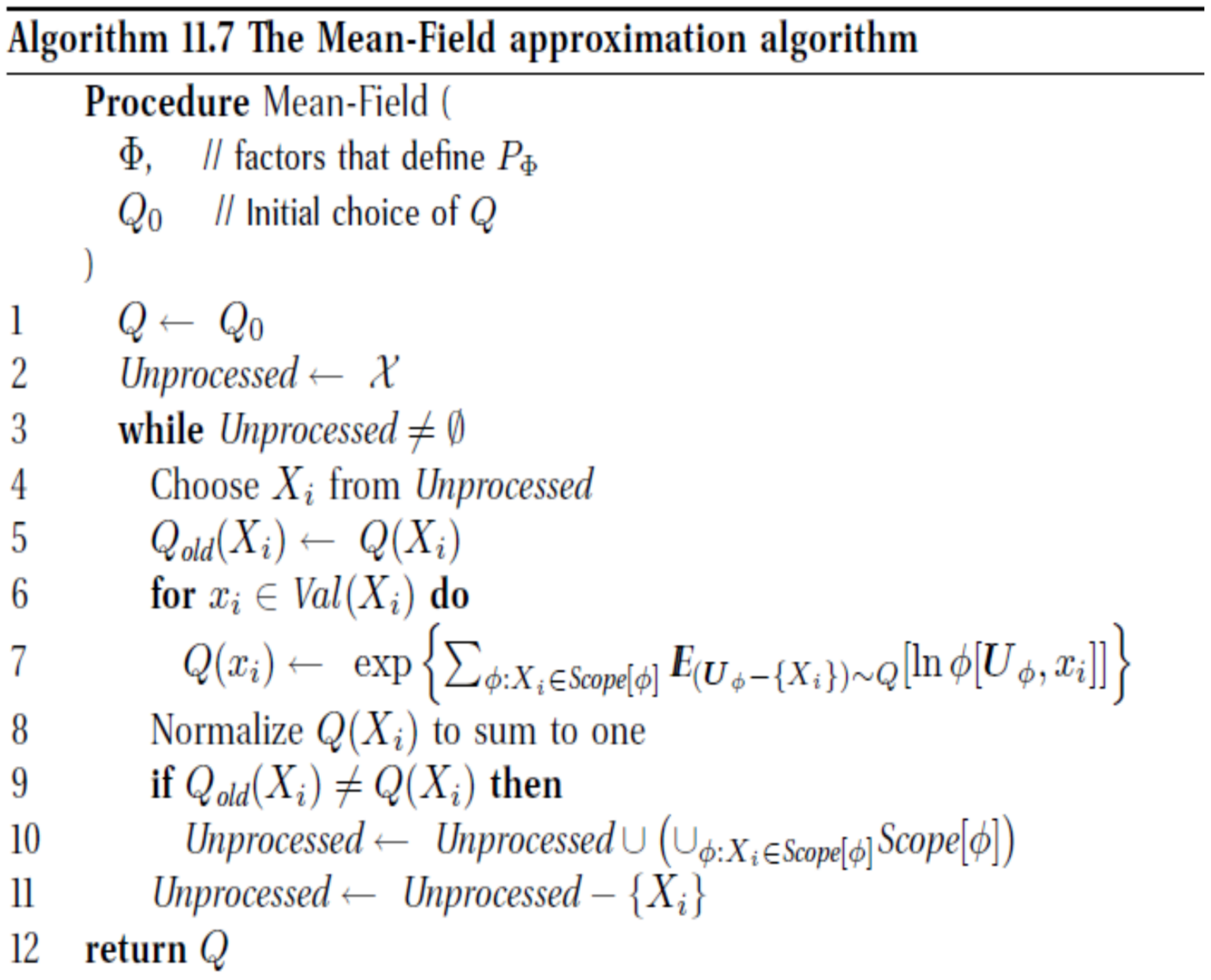
Grid Network Example

- Q for each node is a product of four factors in which it appears (sum in the log notation)

$$Q(a_{i,j}) = \frac{1}{Z_{i,j}} \exp \left\{ \begin{array}{l} \sum_{a_{i-1,j}} Q(a_{i-1,j}) \ln(\phi(a_{i-1,j}, a_{i,j})) + \\ \sum_{a_{i,j-1}} Q(a_{i,j-1}) \ln(\phi(a_{i,j-1}, a_{i,j})) + \\ \sum_{a_{i+1,j}} Q(a_{i+1,j}) \ln(\phi(a_{i,j}, a_{i+1,j})) + \\ \sum_{a_{i,j+1}} Q(a_{i,j+1}) \ln(\phi(a_{i,j}, a_{i,j+1})) \end{array} \right\}.$$

- The term in the exponent is the weighted “mean” of values of neighboring nodes; weight is function of affinity
- Iterative algorithm (11.7), see next slide
- Note: energy always decreases so algorithm is guaranteed to converge, but only to a local minimum

Algorithm 11.7

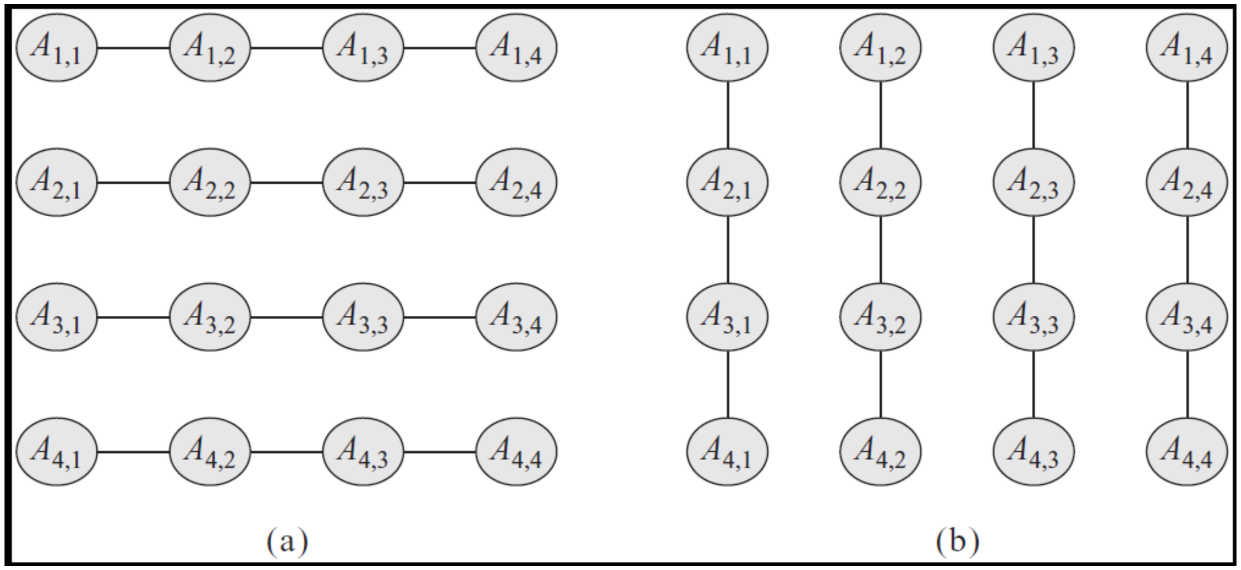


Algorithm 11.7 The Mean-Field approximation algorithm

```
Procedure Mean-Field (
  $\Phi$,   // factors that define $P_\Phi$
  $Q_0$    // Initial choice of Q
)
  $Q \leftarrow Q_0$
  $Unprocessed \leftarrow \mathcal{X}$
  while $Unprocessed \neq \emptyset$
    Choose $X_i$ from $Unprocessed$
    $Q_{old}(X_i) \leftarrow Q(X_i)$
    for $x_i \in Val(X_i)$ do
      $Q(x_i) \leftarrow \exp\left\{\sum_{\phi : X_i \in Scope[\phi]} \boldsymbol{E}_{(\boldsymbol{U}_\phi - \{X_i\}) \sim Q}[\ln \phi[\boldsymbol{U}_\phi, x_i]]\right\}$
    Normalize $Q(X_i)$ to sum to one
    if $Q_{old}(X_i) \neq Q(X_i)$ then
      $Unprocessed \leftarrow Unprocessed \cup \left(\cup_{\phi : X_i \in Scope[\phi]} Scope[\phi]\right)$
    $Unprocessed \leftarrow Unprocessed - \{X_i\}$
return $Q$
```

Structured Approximations

- Approximating distribution can be more complex (but still allow for efficient, exact inferences); see example below



$A_{1,1}$ $A_{1,2}$ $A_{1,3}$ $A_{1,4}$
$A_{2,1}$ $A_{2,2}$ $A_{2,3}$ $A_{2,4}$
$A_{3,1}$ $A_{3,2}$ $A_{3,3}$ $A_{3,4}$
$A_{4,1}$ $A_{4,2}$ $A_{4,3}$ $A_{4,4}$
(a)
(b)

- It is not difficult to derive conditions for maximizing the energy functional for any Q represented as a Gibbs distribution (eq. 11.61)
- Updates over the factors containing cliques in Q
  - May require inferences on the Q distribution after each step
  - In our earlier example, inference was easy as each node was not connected to the others
- We skip the details of this process

Structured Approximations vs LBP

- Structured Approximations
  - Iterative procedure with guarantee of convergence
  - Convergence may be to a local minimum
  - Trade-off between expressive power of the approximating function and computation time
- LBP
  - Easy to understand and implement
  - No guarantee of convergence
  - Seems to work "well" for many practical problems
- In general, empirical evaluation can be difficult as we may not be able to find the optimal, even for comparisons, due to inherent computational complexity.

Particle Based Approximate Inference

- In this approach, we simulate and collect specific instantiations of the variables, called *particles*, according to the network distributions, and use them to approximate the overall distribution.
- If we have "enough" particles, we can get a "good" approximation of the distribution function.
- Task will be to estimate some distribution P(**Y**= **y**) (note **Y** can be a set of variables)
  - More generally, compute the expectation (expected value) of some function f(*X*).
- Notation: ξ <**Y**> is the assignment in ξ to the variables **Y**
  - Note: ξ is pronounced as "ksi" but we often drop "k"
  - Choose f(ξ) = ***1***{ξ <**Y**> = **y** } to compute P(**Y**= **y**)

Sampling a single discrete random variable

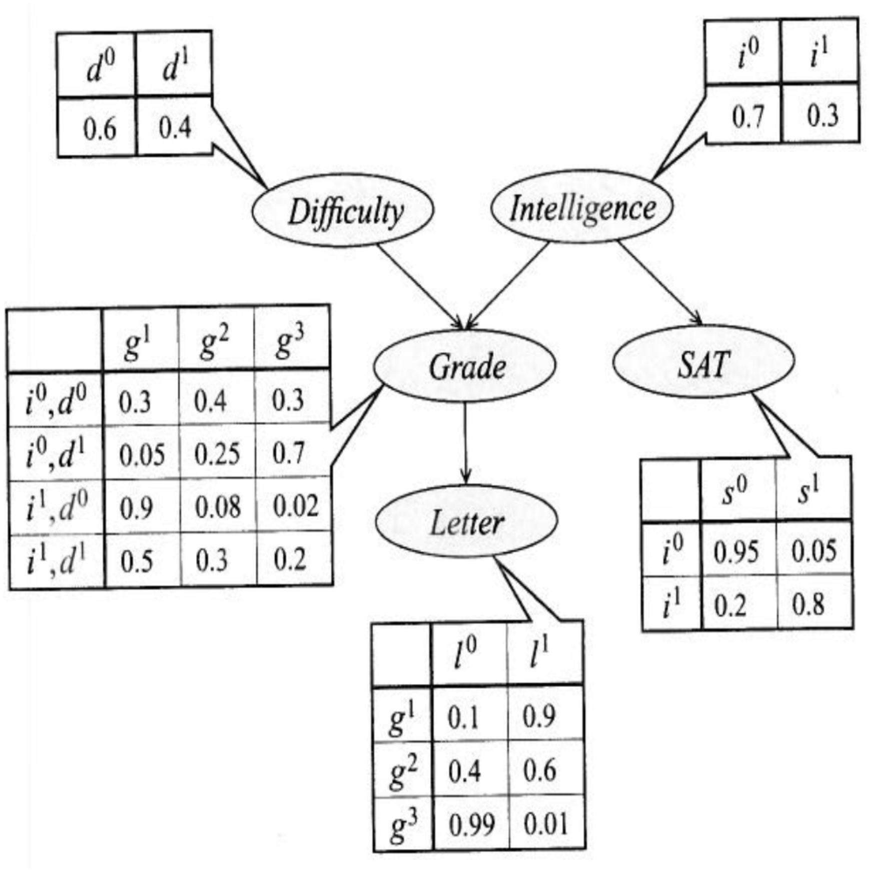
- How to generate random samples of x given **P**(x).
  - Consider a fair coin toss, for example
  - Number of H and T results s/b equal over a very long sequence
  - However, alternating sequence is of low probability; many unbalanced sequences should occur with varying probabilities
  - Coin has no memory; how to simulate its behavior?
- One approach is to sample from a uniform distribution random number generator in range of [0, 1] and map to desired P(X)
- How to sample from the uniform distribution?
  - Observe a physical phenomenon satisfying this law
  - Pseudo-random number generator
    - Deep mathematical topic; Example: $X_{n+1} = (aX_n + c) \bmod m$
      - Choice of parameters determines “quality”
  - In C/C++ , srand (time) to initialize, rand(1) to get first number etc.
  - Mersenne Twister (high quality, fast random number generator)

Mapping to Specific Distributions

- Consider X to represent results of a coin toss
  - Declare Head if $r < 0.5$, Tail otherwise for fair coin
  - Let P(head) = 0.7, then declare Head if $r < 0.7$
- Consider X to be sum of pair of dice, all probabilities are not necessarily equal
  - Make ranges proportional to probabilities of the 12 outcomes
- Continuous distributions: let $F(x)$ be the cumulative distr function
- Let a sample from [0,1] have value z, then:
- Choose smallest x for which $F(x) > z$, *i.e.* compute $F^{-1}(z)$
  - This provides the desired sample (Proof omitted)
- Computing $F^{-1}(z)$ may be difficult for some functions, such as a Gaussian; numerical methods may need to be used
  - More specialized methods exist for Gaussian functions
    - *e.g.* Add multiple uniform random numbers

Forward Sampling

- Generate random samples ξ[1], ξ[2],…, ξ[M] from P(X).
- Consider a BN, $P_B(X)$; example below



d^0	d^1
0.6	0.4

i^0	i^1
0.7	0.3

	g^1	g^2	g^3
i^0,d^0	0.3	0.4	0.3
i^0,d^1	0.05	0.25	0.7
i^1,d^0	0.9	0.08	0.02
i^1,d^1	0.5	0.3	0.2

	s^0	s^1
i^0	0.95	0.05
i^1	0.2	0.8

	l^0	l^1
g^1	0.1	0.9
g^2	0.4	0.6
g^3	0.99	0.01

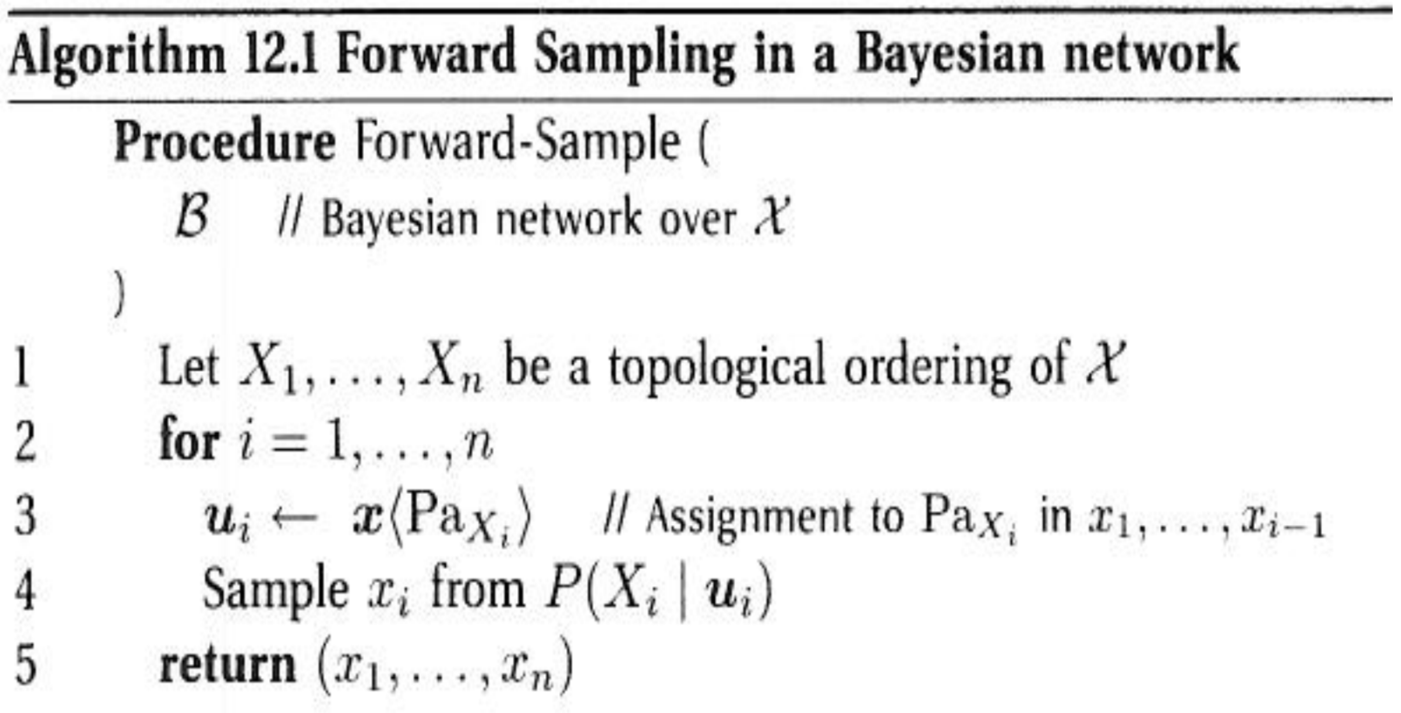
First sample D, let D = d^1

Sample I, let I= i^0

Sample G, from $P(G|i^0,d^1)$
and so on

See Alg 12.1

Forward Sampling Algorithm



Algorithm 12.1 Forward Sampling in a Bayesian network

```
Procedure Forward-Sample (
    $\mathcal{B}$    // Bayesian network over $\mathcal{X}$
)
    Let $X_1, \ldots, X_n$ be a topological ordering of $\mathcal{X}$
    for $i = 1, \ldots, n$
        $\boldsymbol{u}_i \leftarrow \boldsymbol{x}\langle \mathrm{Pa}_{X_i} \rangle$    // Assignment to $\mathrm{Pa}_{X_i}$ in $x_1, \ldots, x_{i-1}$
        Sample $x_i$ from $P(X_i \mid \boldsymbol{u}_i)$
    return $(x_1, \ldots, x_n)$
```

Expectation from Samples

- From the set of particles, D = {ξ[1], ξ[2],…, ξ[M]}, we can compute expectation of any function of samples:

$$\hat{\boldsymbol{E}}_{\mathcal{D}}(f) = \frac{1}{M}\sum_{m=1}^{M} f(\xi[m]).$$

- If we want to compute P(**y**), f is the indicator function equal to 1 when assignments to **Y** in ξ[M] = **y** ; alternate notation is **y**[m] or ξ[M] <**y** >

$$\hat{P}_{\mathcal{D}}(\boldsymbol{y}) = \frac{1}{M}\sum_{m=1}^{M} \boldsymbol{1}\{\boldsymbol{y}[m] = \boldsymbol{y}\},$$

- P(G= g^1, i^0) = (1/M) * #(D=?, I= i^0 , G= g^1 , L=?, S= ?)
- P(G= g^1| i^0) = #(D=?, I= i^0 , G= g^1 , L=?, S= ?)/
  #(D=?, I= i^0 , G= ? , L=?, S= ?)

Error Analysis

- Absolute error (P(**y**) ± ε) or relative error (P(**y**) is between estimate ρ*(1+ ε) and ρ/(1+ ε))
- Probability 1 – δ that estimate is within error bound ε (absolute/relative)
- For absolute error, relation between δ, ε and M is given by

$$M \geq \frac{\ln(2/\delta)}{2\epsilon^2}.$$

  - Example: set δ = .05, ε= .1, M >= log40/2*.01 = 185; change ε to .01, M= 18,500 etc

- For relative error: $M \geq 3\frac{\ln(2/\delta)}{P(y)\epsilon^2}.$
  - Note: number of samples depends inversely on the actual value of P(y)
    - Note: we don't actually know P(y) so hard to estimate M

Rejection Sampling

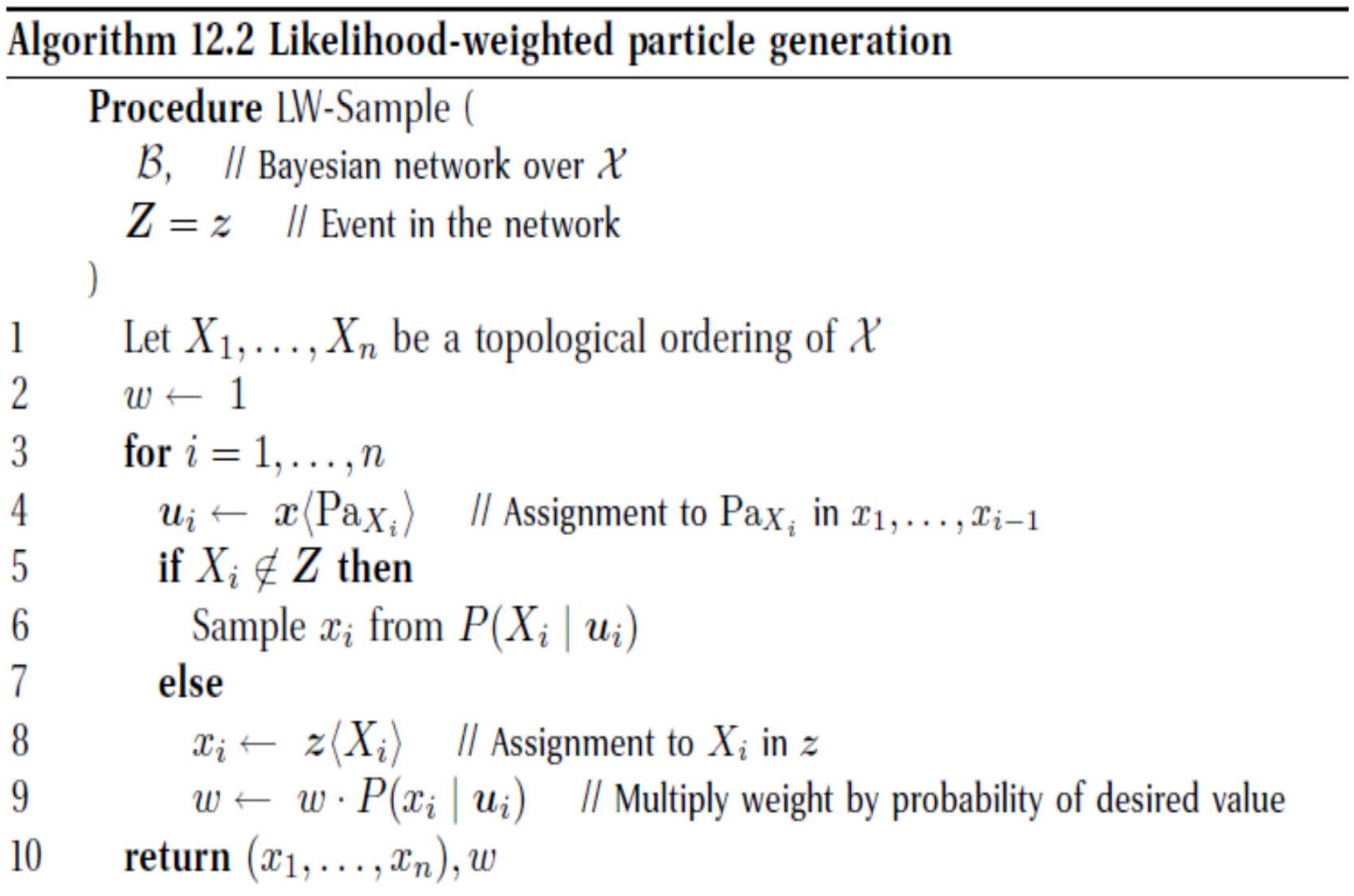
- To compute conditional probabilities P(**Y**=**y** | **E**=**e**), we compute #[**Y**=**y** | **E**=**e**] / #[**E**=**e**]
- We throw away samples where **E** ≠ **e**
- If P(**e**) is small, we will throw away a large fraction of the samples; getting enough useful samples for reliable estimation will require a very large number of total samples
- Intuitive Solution: Set variables in **E** to have value **e**, then generate samples
  - Problem: generated samples are biased, see next slide

Likelihood Weighting

- We can just set evidence variables to the observed values to avoid rejecting samples
- However, this biases the samples. Suppose that evidence is S=s^1;
- Apply normal forward sampling, we would start from prior distributions of D and I, so I=i^1 in only 30% of the cases; this is clearly not consistent with S=s^1.
- In rejection sampling, if we started with I=i^1 then S=s^1 in 80% of the samples (from CPD); if we started with I=i^0 then S=s^1 in only 5% samples
- Weight samples by their likelihood: (i^1,s^1) by .8, (i^0,s^1) by .05
  - This weighting compensation is intuitive, formal justification comes later
  - Generalizes to case where more than one variable is evidence variable (product of probabilities for generating evidence variables given their parents; specifics in Algorithm 12.2, next slide).

$$\hat{P}_{\mathcal{D}}(\boldsymbol{y} \mid \boldsymbol{e}) = \frac{\sum_{m=1}^{M} w[m] \boldsymbol{I}\{\boldsymbol{y}[m] = \boldsymbol{y}\}}{\sum_{m=1}^{M} w[m]}$$

LW Algorithm



Algorithm 12.2 Likelihood-weighted particle generation

```
Procedure LW-Sample (
    $\mathcal{B}$,     // Bayesian network over $\mathcal{X}$
    $\boldsymbol{Z} = \boldsymbol{z}$     // Event in the network
)
1     Let $X_1, \ldots, X_n$ be a topological ordering of $\mathcal{X}$
2     $w \leftarrow 1$
3     for $i = 1, \ldots, n$
4       $\boldsymbol{u}_i \leftarrow \boldsymbol{x}\langle \mathrm{Pa}_{X_i} \rangle$     // Assignment to $\mathrm{Pa}_{X_i}$ in $x_1, \ldots, x_{i-1}$
5       if $X_i \notin \boldsymbol{Z}$ then
6         Sample $x_i$ from $P(X_i \mid \boldsymbol{u}_i)$
7       else
8         $x_i \leftarrow \boldsymbol{z}\langle X_i \rangle$     // Assignment to $X_i$ in $\boldsymbol{z}$
9         $w \leftarrow w \cdot P(x_i \mid \boldsymbol{u}_i)$     // Multiply weight by probability of desired value
10    return $(x_1, \ldots, x_n), w$
```

Next Class

- Read sections 12.2 and 12.3 of the KF book