Lecture 8 - Reasoning under Uncertainty (Part II)

Jesse Hoey
School of Computer Science
University of Waterloo

February 16, 2018

Readings: Poole & Mackworth (2nd ed.) Chapt. 8.5 - 8.9
A node repeats over time

explicit encoding of time

Chain has length = amount of time you want to model

Event-driven times or Clock-driven times

e.g. Markov chain
Markov assumption

\[ P(S_{t+1}|S_1, \ldots, S_t) = P(S_{t+1}|S_t) \]

This distribution gives the **dynamics** of the Markov chain
Hidden Markov Models (HMMs)

Add: observations $O_t$ (always observed, so the node is square) and observation function $P(O_t|S_t)$

Given a sequence of observations $O_1, \ldots, O_t$, can estimate filtering:

$$P(S_t|O_1, \ldots, O_t)$$

or smoothing, for $k < t$

$$P(S_k|O_1, \ldots, O_t)$$
Speech Recognition

- Most well known application of HMMs
- Observations: audio features
- States: phonemes
- Dynamics: models e.g. co-articulation
- HMMs: words
- Can build hierarchical models (e.g. sentences)
Belief Monitoring in HMMs

\[ \alpha_i = P(S_i | o_0 \ldots, o_i) \]
\[ \propto P(S_i, o_0, \ldots, o_i) \]
\[ = P(o_i | S_i) \sum_{S_{i-1}} P(S_{i-1}, o_0, \ldots, o_{i-1}) \]
\[ = P(o_i | S_i) \sum_{S_{i-1}} P(S_i | S_{i-1}) P(S_{i-1}, o_0, \ldots, o_{i-1}) \]
\[ \propto P(o_i | S_i) \sum_{S_{i-1}} P(S_i | S_{i-1}) \alpha_{i-1} \]
Belief Monitoring in HMMs

\[
\beta_{i+1} = P(o_{i+1} \ldots, o_T | S_i)
= \sum_{S_{i+1}} P(S_{i+1}, o_{i+1}, \ldots, o_T | S_i)
= \sum_{S_{i+1}} P(o_{i+1} | S_{i+1}, o_{i+2}, \ldots, o_T, S_i)P(S_{i+1}, o_{i+2}, \ldots, o_T | S_i)
= \sum_{S_{i+1}} P(o_{i+1} | S_{i+1})P(o_{i+2}, \ldots, o_T | S_{i+1}, S_i)P(S_{i+1} | S_i)
= \sum_{S_{i+1}} P(o_{i+1} | S_{i+1})P(S_{i+1} | S_i)\beta_{i+2}
\]
Belief Monitoring in HMMs

together:

\[ \alpha_i \beta_{i+1} = P(o_{i+1}, \ldots, o_T|S_i)P(S_i|o_0, \ldots, o_i) \propto P(S_i|O) \]
Dynamic Bayesian Networks (DBNs)

Many examples, and they can be solved with variable elimination, but this may become too complex with enough variables.
Dynamic Bayesian Networks (DBNs)

Many examples, and they can be solved with variable elimination, but this may become too complex with enough variables.

- Event-driven times or Clock-driven times
Example: localization

- Suppose a robot wants to determine its location based on its actions and its sensor readings: Localization
- This can be represented by the augmented HMM:
Example localization domain

- Circular corridor, with 16 locations:

```
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

- Doors at positions: 2, 4, 7, 11.
- Noisy Sensors
- Stochastic Dynamics
- Robot starts at an unknown location and must determine where it is, known as the “kidnapped robot” problem.
Example Sensor Model

- $P(\text{Observe Door} \mid \text{At Door}) = 0.8$
- $P(\text{Observe Door} \mid \text{Not At Door}) = 0.1$
Example Dynamics Model

- $P(Loc_{t+1} = l | Action_t = goRight \land Loc_t = l) = 0.1$
- $P(Loc_{t+1} = l + 1 | Action_t = goRight \land Loc_t = l) = 0.8$
- $P(Loc_{t+1} = l + 2 | Action_t = goRight \land Loc_t = l) = 0.074$
- $P(Loc_{t+1} = l' | Action_t = goRight \land Loc_t = l) = 0.002$ for any other location $l'$.
  - All location arithmetic is modulo 16.
  - The action $goLeft$ works the same but to the left.
Example sequence

observe door, go right, observe no door, go right, observe door
where is the robot?

\[ P(\text{Loc}_2 = 4 | O_0 = d, A_0 = r, O_1 = \neg d, A_1 = r, O_2 = d) = 0.42 \]
Combining sensor information

- **Example:** we can combine information from a light sensor and the door sensor. **Sensor Fusion**

  \[ \text{Loc}_t \text{ robot location at time } t \]
  \[ \text{D}_t \text{ door sensor value at time } t \]
  \[ \text{L}_t \text{ light sensor value at time } t \]
Figure 2: A portion of a Bayesian user model for inferring the likelihood that a user needs assistance, considering profile information as well as observations of recent activity.

Probability Distribution and Monte Carlo

John von Neumann
1903 - 1957

Stanislaw Ulam
1909-1984

ENIAC
1949

Monte Carlo
1949
Idea: probabilities ↔ samples

Get probabilities from samples:

<table>
<thead>
<tr>
<th>( X )</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>( n_1 )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( x_k )</td>
<td>( n_k )</td>
</tr>
<tr>
<td>total</td>
<td>( m )</td>
</tr>
</tbody>
</table>

\[
\text{probability} \quad \leftrightarrow \quad \text{count} / \text{total} \\
\begin{array}{c|c}
\hline
X & probability \\
\hline
x_1 & n_1 / m \\
\vdots & \vdots \\
x_k & n_k / m \\
\hline
\end{array}
\]

If we could sample from a variable’s (posterior) probability, we could estimate its (posterior) probability.
Generating samples from a distribution

For a variable $X$ with a discrete domain or a (one-dimensional) real domain:

- Totally order the values of the domain of $X$.
- Generate the cumulative probability distribution: $f(x) = P(X \leq x)$.
- Select a value $y$ uniformly in the range $[0, 1]$.
- Select the $x$ such that $f(x) = y$. 

![Cumulative Distribution Function](image)
Hoeffding Bound

$p$ is true probability, $s$ is sample average, $n$ is number of samples

- $P(|s - p| > \epsilon) \leq 2e^{-2n\epsilon^2}$
- if we want an error greater than $\epsilon$ in less than a fraction $\delta$ of the cases, solve for $n$:
  \[
  2e^{-2n\epsilon^2} < \delta
  \]
  \[
  n > \frac{-\ln \delta}{2\epsilon^2}
  \]

we have

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>cases with error $&gt; \epsilon$</th>
<th>samples needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>1/20</td>
<td>184</td>
</tr>
<tr>
<td>0.01</td>
<td>1/20</td>
<td>18,445</td>
</tr>
<tr>
<td>0.1</td>
<td>1/100</td>
<td>265</td>
</tr>
</tbody>
</table>
Forward sampling in a belief network

- Sample the variables one at a time; sample parents of $X$ before you sample $X$.
- Given values for the parents of $X$, sample from the probability of $X$ given its parents.
To get \( P(H = h_i | E = e_i) \) simply

- count the number of samples that have \( H = h_i \) and \( E = e_i \), \( N(h_i, e_i) \)
- divide by the number of samples that have \( E = e_i \), \( N(e_i) \)

\[
P(H = h_i | E = e_i) = \frac{P(H = h_i \land E = e_i)}{P(E = e_i)} = \frac{N(h_i, e_i)}{N(e_i)}
\]
Forward Sampling

Inference via sampling

\[
P(\text{cancer|database})
\]

number of samples
Rejection Sampling

To estimate a posterior probability given evidence
\[ Y_1 = v_1 \land \ldots \land Y_j = v_j : \]
If, for any \( i \), a sample assigns \( Y_i \) to any value other than \( v_i \), reject that sample.
The non-rejected samples are distributed according to the posterior probability.
in the Hoeffding bound, \( n \) is the number of non-rejected samples
If we draw $N$ samples $s_{i=1...N}$ by

- sampling $A$: $a_{i=1...N}$
- sampling from $E$ given $A$: $e_{i=1...N}$

then

- $N_T = 0.4N$ of them will have $A = true$, and of these 10% will have $E = true$
- $N_F = 0.6N$ of them will have $A = false$, and of these 30% will have $E = true$
Example Network

so we have

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>E</th>
<th>( N_{AE} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
<td>( N_{TF} = 0.4 \times 0.9 \times N )</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>( N_{TT} = 0.4 \times 0.1 \times N )</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>( N_{FF} = 0.6 \times 0.7 \times N )</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>( N_{FT} = 0.6 \times 0.3 \times N )</td>
<td></td>
</tr>
</tbody>
</table>

We want to compute

\[
P(a|e) = P(A = true|E = true) \propto \sum_{s_i} \delta(a_i = true)\delta(e_i = true)
\]

\[
P(a|e) = \frac{P(a \land e)}{P(e)} = \frac{N_{TT}}{N_{TT} + N_{FT}}
\]

\[
= \frac{0.1 \times 0.4 \times N}{0.1 \times 0.4 \times N + 0.3 \times 0.6 \times N} = 0.182
\]
we can do better since we can “weight” the samples

weights = prob. that the evidence is observed

$N_T$ samples with $A = true$ have weight of $w_t = 0.1$
this is $P(E = true | A = true)$

$N_F$ samples with $A = false$ have weight of $w_f = 0.3$
this is $P(E = true | A = false)$

can do better because we don’t need to generate the 90% of samples (when $A = true$) that don’t agree with the evidence - we simply assign all samples a weight of 0.1

thus, we are mixing exact inference (the 0.1) with sampling.
Now, compute sum of all weights of the samples with \( A = true \)
\[
W_T = \sum_i w_t \delta(a_i = true) = N_T \times 0.1
\]

Now, compute sum of all weights of the samples with \( A = false \)
\[
W_F = \sum_i w_f \delta(a_i = false) = N_F \times 0.3
\]

Finally, compute
\[
P(a|e) = \frac{W_T}{W_T + W_F} = \frac{0.1 \times 0.4 \times N}{0.1 \times 0.4 \times N + 0.3 \times 0.6 \times N}
\]
In fact, the As don’t need to even be sampled from $P(A)$
Can be sampled from some $q(A)$, say $q(A = \text{true}) = 0.5$
and each sample will have a new weight $P(a)/q(a)$
$q(A)$ is a *proposal* distribution.
Importance weights

<table>
<thead>
<tr>
<th>P(A)</th>
<th>A</th>
<th>E</th>
<th>A</th>
<th>P(E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
</tr>
</tbody>
</table>

- Helps when it is hard to sample from $P(A)$, but we can evaluate $P^*(A) \propto P(A)$ given a sample
- Rejection sampling uses $q = P$
- Rejection sampling uses all variables including observed ones, and all weights on samples are set to 1.0
Importance weights

- $N_T' = q(a) N$ samples with $A = true$ have weight of
  \[ 0.1 \times \frac{P^*(a)}{q(a)} = 0.1 \times \frac{0.4}{0.5} \]
- $N_F' = q(\overline{a}) N$ samples with $A = false$ have weight of
  \[ 0.3 \times \frac{P^*(a)}{q(\overline{a})} = 0.3 \times \frac{0.6}{0.5} \]
Importance weights

<table>
<thead>
<tr>
<th>A</th>
<th>P(E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>0.1</td>
</tr>
<tr>
<td>F</td>
<td>0.3</td>
</tr>
</tbody>
</table>

- total weight of all samples with $A = true$

$$W'_T = \sum_i w_i \delta(a_i = true) = N'_T \times 0.1 \times \frac{\alpha^{0.4}}{0.5}$$

$$= 0.5N \times 0.1 \times \frac{\alpha^{0.4}}{0.5} = 0.1 \times \alpha \times 0.4 \times N$$

- total weight of all samples with $A = false$

$$W'_F = \sum_i w_i \delta(a_i = true) = N'_F \times 0.3 \times \frac{\alpha^{0.6}}{0.5}$$

$$= 0.5N \times 0.3 \times \frac{\alpha^{0.6}}{0.5} = 0.3 \times \alpha \times 0.6 \times N$$
finally, compute

\[ P(a|e) = \frac{W'_T}{W'_T + W'_F} = \frac{0.1 \times \alpha \times 0.4 \times N}{0.1 \times \alpha \times 0.4 \times N + 0.3 \times \alpha \times 0.6 \times N} \]
Sometimes we may want to choose a proposal distribution that is different than the actual probability distribution.

We may want to “skew” the proposal - because we may have some additional knowledge about the data, for example.

or, we can generate proposals from the data itself using some procedural knowledge that is not directly encoded in the BN.

Can be important in multiple/many dimensions,
Stochastic sampling

Recall variable elimination: To compute \( P(Z, Y_1 = v_1, \ldots, Y_j = v_j) \), we sum out the other variables, \( Z_1, \ldots, Z_k = \{X_1, \ldots, X_n\} - \{Z\} - \{Y_1, \ldots, Y_j\} \).

\[
P(Z, Y_1 = v_1, \ldots, Y_j = v_j) = \sum_{Z_k} \cdots \sum_{Z_1} \prod_{i=1}^{n} P(X_i|\text{parents}(X_i)) Y_1 = v_1, \ldots, Y_j = v_j
\]

Now, we sample \( Z_{l+1}, \ldots, Z_k \) and sum \( Z_1, \ldots, Z_l \),

\[
= \sum_{s_l = \{z_{l+1}, i, \ldots, z_k, i\}} \left[ \sum_{Z_1 \cdots Z_l} \prod_{i=1}^{l} P(X_i|\text{parents}(X_i)) Y_1 = v_1, \ldots, Y_j = v_j \right] \frac{P(Z_{l+1}, i, \ldots, Z_k, i)}{q(Z_{l+1}, i, \ldots, Z_k, i)}
\]
Compute $P(B|D = \text{true}, A = \text{false})$ by sampling $C$ and $M$.

- use $q(C = \text{true}) = P(C = \text{true}) = 0.32$
  and $q(M = \text{true}) = P(M = \text{true}) = 0.08$
- use $q(C = \text{true}) = 0.5$
  and $q(M = \text{true}) = P(M = \text{true}) = 0.08$
- use $q(C = \text{true}) = q(M = \text{true}) = 0.5$

$$P(B|D = \text{true}, A = \text{false}) \propto \sum_{s_i = \{c_i, m_i\}} P(B, D = \text{true}, A = \text{false}|c_i, m_i)$$
Stochastic Sampling for HMMs (and other DBNS)

Sequential Monte Carlo or “Particle Filter”
- sequential stochastic sampling
- keep track of $P(S_t)$ at the current time $t$
- represent $P(S_t)$ with a set of samples
- update as new observations $o_{t+1}$ arrive
  1. predict $P(S_{t+1}) \propto P(S_{t+1}|S_t)$
  2. compute weights as $P(o_{t+1}|S_{t+1})$
  3. resample according to weights
Particle Filtering

$P(X)$

sample i: \( \{ x_i \} \)
Particle Filtering

**step 0: prior belief**

\[ P(X) \]

sample \( i: \{x_i, w_i\} \)
Particle Filtering

**step 1: hypothesis (prediction)**

P(X)

sample i: \{x_i, w_i\}
Particle Filtering

**step 2: stochastic spreading**

\[ P(X) \]

**sample i:** \( \{x_i, w_i\} \)
step 3: evidence

sample i: \{x_i, w_i\}
Particle Filtering

step 4: resample

\[ \text{sample } i: \{ x_i, w_i \} \]
Bayesian Sequential Updates

1. Predict
2. Spread
3. Evidence
4. Predict

$P(X)$

$X$

1

2

3

4

$P(X)$

Evidence
Resampling

- avoids degeneracies in the samples
- all importance weights $\to 0$ except one
- performance of the algorithm depends on the resampling method.
Learning with Uncertainty (Poole & Mackworth (2nd ed.) chapter 10)