Exact inference in probabilistic graphical models

Marcus Frean
Victoria University of Wellington
Hidden Markov models
Hidden Markov models
Hidden Markov models
Hidden Markov models

we want posterior probabilities: \( p(x_t | y_{1..T}) \)

we want better numbers to put in the squares...
\[ p(x_t|y_{1..T}) \propto p(x_t, y_{1..t-1}) \cdot p(y_t|x_t) \cdot p(y_{t+1..T}|x_t) \]

i.e. multiply together "messages" arriving from other nodes in this graph:

\[ \rightsquigarrow \uparrow \leftarrow \]
generated by "forward" and "backward" recursions
Function of $n$ variables $x$, that can take $k$ distinct values

$F(x_1, x_2, \ldots x_n)$
sums
sums
How many sums are involved in adding up $F$?

\[
\sum_{x_1} \sum_{x_2} \ldots \sum_{x_n} F(x_1, x_2, \ldots x_n)
\]

- exponential with $n$
- intractable
slo}s of products

Suppose $F$ “breaks up” into a product, e.g.

$$F(x_1, x_2, x_3) = \Phi_A(x_1, x_2) \cdot \Phi_B(x_2, x_3)$$

Now,

$$\sum \sum \sum F = \sum \sum \Phi_A(x_1, x_2) \sum \Phi_B(x_2, x_3)$$

☆ this buys tractability

☆ think of it as a graph
sums of products by message passing (1)

\[
\sum_{x_1} \sum_{x_2} \sum_{x_3} F = \sum_{x_1} \sum_{x_2} \Phi_A(x_1, x_2) \sum_{x_3} \Phi_B(x_2, x_3)
\]

➤ this leaves a vector having the dimensions of \(x_2\)
sums of products by message passing (1)

\[ \sum_{x_1} \sum_{x_2} \sum_{x_3} F = \sum_{x_1} \sum_{x_2} \Phi_A(x_1, x_2) \sum_{x_3} \Phi_B(x_2, x_3) \]

➤ this leaves a vector having the dimensions of \( x_2 \)
➤ think of result as a message going off to \( x_2 \)
soms of products by message passing (1)

\[
\sum_{x_1} \sum_{x_2} \sum_{x_3} F = \sum_{x_1} \sum_{x_2} \Phi_A(x_1, x_2) \sum_{x_3} \Phi_B(x_2, x_3)
\]

do this first

➤ this leaves a vector having the dimensions of \(x_2\)

➤ think of result as a message going off to \(x_2\)

➤ \(x_2\) simply passes it on to \(A\)
sums of products by message passing (2)

\[
\sum_{x_1} \sum_{x_2} \sum_{x_3} F = \sum_{x_1} \sum_{x_2} \Phi_A(x_1, x_2) \cdot msg_{2 \rightarrow A}(x_2)
\]

do this next
sums of products by message passing (2)

\[
\sum \sum \sum F = \sum x_1 \sum x_2 \Phi_A(x_1, x_2) \cdot \text{msg}_{2\rightarrow A}(x_2)
\]

do this next

➤ this leaves a vector having the dimensions of \(x_1\)
Sums of products by message passing (3)

\[ \sum_{x_1} \sum_{x_2} \sum_{x_3} F = \sum_{x_1} \text{msg}_{A \rightarrow 1}(x_1) \]

do this last
What if we fix (say) $x_3=1$?

$$
\sum_{x_1} \sum_{x_2} \sum_{x_3} F = \sum_{x_1} \sum_{x_2} \Phi_A(x_1, x_2) \sum_{x_3} \Phi_B(x_2, x_3)
$$

but $x_3 = 1$!
revise the first step...

\[
\sum_{x_1} \sum_{x_2} \sum_{x_3} F = \sum_{x_1} \sum_{x_2} \Phi_A(x_1, x_2) \sum_{x_3} \Phi_B(x_2, x_3) \delta(x_3 = 1)
\]

➤ a new message coming from \(x_3\) to \(B\), consisting of zeros, except for the one we want to specify a value for

➤ multiply \(\Phi_B\) by the incoming message, and sum the result over the dimension corresponding to \(x_2\)
step 1 again...

- even if we’re *not* fixing a value for $x_3$, send a message from it anyway, consisting of $(1, 1, \ldots, 1)$
- both $A$ and $B$ now take an incoming message, multiply it by their table, and sum out the “incoming” dimension
another example

\[
F(x_1, x_2, x_3, x_4, x_5) = \Phi_A(x_1, x_2, x_3) \Phi_B(x_3, x_4) \Phi_C(x_4, x_5)
\]

\[
\sum_{x_1 \ldots x_5} F = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \Phi_A(x_1, x_2, x_3) \Phi_B(x_3, x_4) \Phi_C(x_4, x_5)
\]
another example (cont)

\[ F(x_1, x_2, x_3, x_4, x_5) = \Phi_A(x_1, x_2, x_3) \Phi_B(x_3, x_4) \Phi_C(x_4, x_5) \]

\[ \sum F = \sum_{x_1} \sum_{x_2} \sum_{x_3} \Phi_A(x_1, x_2, x_3) \sum_{x_4} \Phi_B(x_3, x_4) \sum_{x_5} \Phi_C(x_4, x_5) \]
unless we’re prescribing values, variable nodes send “ones”
\[ \sum_{x_1 \ldots x_5} F = \sum_{x_1} \sum_{x_2} \sum_{x_3} \Phi_A(x_1, x_2, x_3) \sum_{x_4} \Phi_B(x_3, x_4) \sum_{x_5} \Phi_C(x_4, x_5) \]

done by B

done by C
\[ \sum_{x_1 \ldots x_5} F = \sum_{x_1} \sum_{x_2} \sum_{x_3} \Phi_A(x_1, x_2, x_3) \text{ msg}_{B \to 3}(x_3) \text{ msg}_{C \to 3}(x_3) \]

* node 3 *multiplies* its incoming messages together, and sends off the result to \( A \)
\[
\sum_{x_1 \ldots x_5} F = \sum_{x_1} \sum_{x_2} \sum_{x_3} \Phi_A(x_1, x_2, x_3) \cdot \text{msg}_{3 \rightarrow A}(x_3) \cdot \text{msg}_{2 \rightarrow A}(x_2)
\]

\(\text{done by } A\)

\(\bullet\) just like before, except it multiplies by both incoming messages, and sums out along both those dimensions
re-use of messages

✪ there’s nothing special about $x_1$
✪ send messages to and from all nodes!
✪ $n$ times the information for only twice the work
✪ can do this in a variety of ways
sum-product algorithm

Messages are sent to, and from, each node
sum-product algorithm

- messages are sent to, and from, each node
- messages always consist of a function over the associated variable
sum-product algorithm

- messages are sent to, and from, each node
- messages always consist of a function over the associated variable
- each outgoing message is always a simple function of incoming messages on all the other edges
sum-product algorithm

- messages are sent to, and from, each node
- messages always consist of a function over the associated variable
- each outgoing message is always a simple function of incoming messages on all the other edges
- but that function is different for the two types of node...
variable nodes are MULTIPLIERS

➢ to generate a message on an edge, multiply (1, 1, . . . , 1) by the incoming messages on all other edges

➢ if the variable is to be preset ("observed") send a one at the observed value only, as in (0, 1, 0, . . . , 0)
factor nodes are SUMMERS

Each dimension of $\Phi$ corresponds to one of its neighbours.

To generate a message on an edge,

1. multiply $\Phi$ by each of the incoming messages on other edges, then

2. sum out the incoming variables
graphical models of probability distributions

A particular example of a function that factorizes is a Markov random field

\[
p(x_1, x_2, x_3) = \frac{1}{Z} \Phi_A(x_1, x_2) \Phi_B(x_2, x_3)
\]

➤ Z is normalisation

➤ here \(x_1\) and \(x_3\) are conditionally independent given \(x_2\)
graphical models of probability distributions

A particular example of a function that factorizes is a Markov random field

\[ p(x_1, x_2, x_3) = \frac{1}{Z} \Phi_A(x_1, x_2) \Phi_B(x_2, x_3) \]

➤ Z is normalisation
➤ here \( x_1 \) and \( x_3 \) are conditionally independent given \( x_2 \)
➤ we usually want to find marginal probabilities like \( p(x_1) \), perhaps conditioned on observations like \( p(x_1|x_3 = 1) \)
belief propagation

➤ “belief propagation” ≡ sum-product algorithm, without the final summation!
belief propagation

➤ “belief propagation” ≡ sum-product algorithm, without the final summation!

➤ Once a variable node $i$ has all its incoming messages, it can calculate $p(x_i|\text{obs})$ by multiplying them together and normalising
belief propagation

➤ “belief propagation” ≡ sum-product algorithm, without the final summation!

➤ Once a variable node $i$ has all its incoming messages, it can calculate $p(x_i|\text{obs})$ by multiplying them together and normalising

➤ Exact for trees (no loops in the graph)
our whizzy code

Tony Vignaux and I have written python code for running belief propagation in arbitrary graphs.

```python
# First we define the variables
b = Multiplier('burglar',2)
e = Multiplier('earthquake',2)
a = Multiplier('alarm',2)

# And now the factors, each with a phi matrix.
B = Summer('B', [b], array([.2,.8]))
E = Summer('E', [e], array([.4,.6]))
A = Summer('A', [b,e,a], array([[.9,.1],[.6,.4]],[[.3,.7]
```
our whizzy code

node burglar has been observed
from node:  B  msg:  [ 0.2  0.8]
from node:  A  msg:  [ 1.  1.]
from node:  OBS  msg:  [ 0.  1.]
posterior:  [ 0.  1.]

node earthquake
from node:  E  msg:  [ 0.4  0.6]
from node:  A  msg:  [ 0.8  0.8]
posterior:  [ 0.4  0.6]

node alarm
from node:  A  msg:  [ 0.336  0.464]
posterior:  [ 0.42  0.58]

You're welcome to this email.
what about loopy graphs?

➤ remove loops by merging variables, OR
➤ resort to MCMC sampling (Gibbs), OR
➤ just run belief propagation anyway... (= turbodecoding)
directed graphs: belief nets
Factorisations arrived at by the product rule, *e.g.*

\[ p(x_1, x_2, x_3) = p(x_3|x_2) \ p(x_2|x_1) \ p(x_1) \]
directed graphs: belief nets
Factorisations arrived at by the product rule, e.g.

\[ p(x_1, x_2, x_3) = p(x_3|x_2) \cdot p(x_2|x_1) \cdot p(x_1) \]

Belief net:
directed graphs: belief nets
Factorisations arrived at by the product rule, e.g.

\[ p(x_1, x_2, x_3) = p(x_3|x_2) \ p(x_2|x_1) \ p(x_1) \]

Belief net:

So far so good, but this graph has no more “expressive power” than a Markov Random Field
another directed graph: “explaining away”

\[ p(x_1, x_2, x_3) = p(x_1) \ p(x_2) \ p(x_3|x_1, x_2) \]
another directed graph: “explaining away”

\[ p(x_1, x_2, x_3) = p(x_1) \ p(x_2) \ p(x_3|x_1, x_2) \]

Structural assumption: \( x_1 \) is independent of \( x_2 \), but becomes dependent once \( x_3 \) is observed.

Crucial to inferences about causality.
what do arrows mean?

➤ how can 1 and 2 be independent? Surely 1 is sending a message, thus affecting the message going to 2...
what do arrows mean?

➤ how can 1 and 2 be *independent*? Surely 1 is sending a message, thus affecting the message going to 2...

➤ it works out only because this factor is *normalised*: provided $x_3$ sends $(1, 1 \ldots 1)$, $x_1$’s message has no effect on the message that goes to $x_2$

➤ So... I think arrows mean local normalisation, which means ‘explaining away’
aside: the meaning of messages

In a belief net the messages are easily interpreted:

» messages passing in the ‘forward’ direction are of the form

\[ p(x, \text{obs}^{\text{behind}}) \]

» messages passing in the ‘backward’ direction are of the form

\[ p(\text{obs}^{\text{ahead}} | x) \]
e.g. Hidden Markov models

HMMs are belief nets, with strong simplifying assumptions:

- no loops...
- no multiple parents, so no “explaining away”...
- transitions and emission factors are the same for all t
e.g. interpretation of messages in HMMs

\[
p(x_t|y_{1..T}) \propto p(x_t, y_{1..t-1}) \cdot p(y_t|x_t) \cdot p(y_{t+1..T}|x_t)
\]
can we learn the factors from data?

- yes, via EM in belief nets (and therefore HMMs)
can we learn the factors from data?

➤ yes, via EM in belief nets (and therefore HMMs)
➤ the basic notion / hand-wave: replace $p(x_i|\text{parents}_i)$ with $p(x_i|\text{parents}_i, \text{all observations})$
➤ The latter is easily obtained: multiply the existing factor by each of its incoming messages, and re-normalise it.

(in HMMs this is called the Baum-Welch algorithm)
EM algorithm in belief nets

- **E STEP:**
  run belief propagation to get all the messages

- **M STEP:**
  $\Phi'$ is product* of $\Phi$ and all the incoming messages

This increases a lower bound on the log likelihood, and (sometimes) works tolerably well in HMMs...
EM example 1

states (using ML model)

obs. expected from ML states (likelihood: -0.7022)

Pr(X→X)

actual state sequence

Pr(X→Y)

observation sequence

Right
EM example 1

Wrong
EM example 1

Red is EM, black is a stupid stochastic gradient ascender.
This is silly...
EM example 2

That’s not bad...
questions?

- Sums of products
- Graphical models of probability distributions
- Directed graphs
- HMM

You are here: ✗