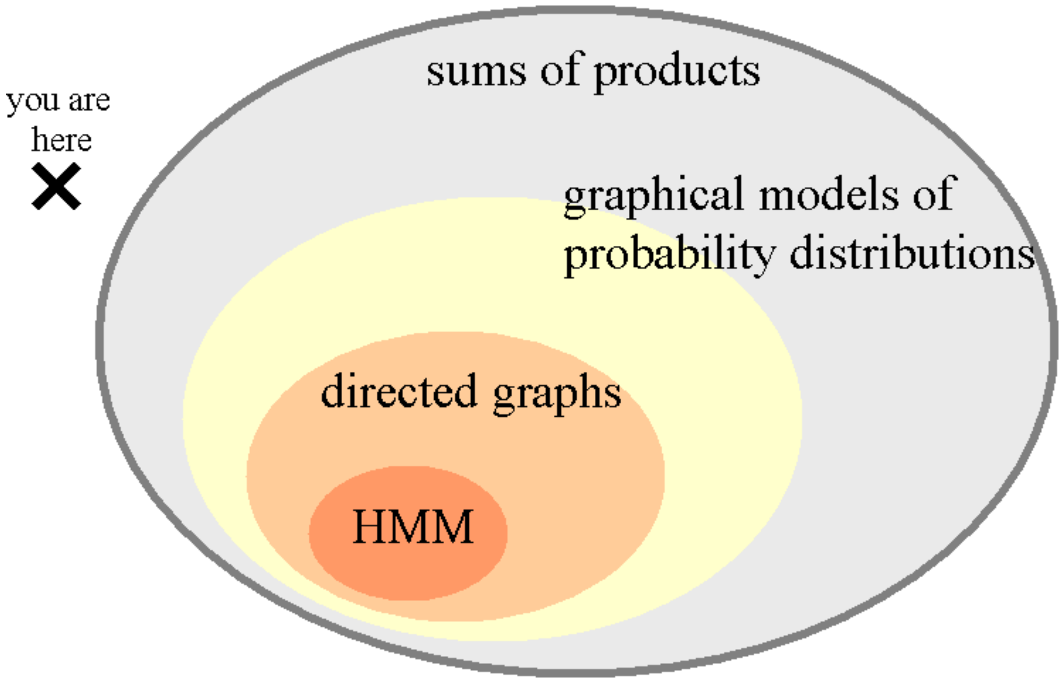
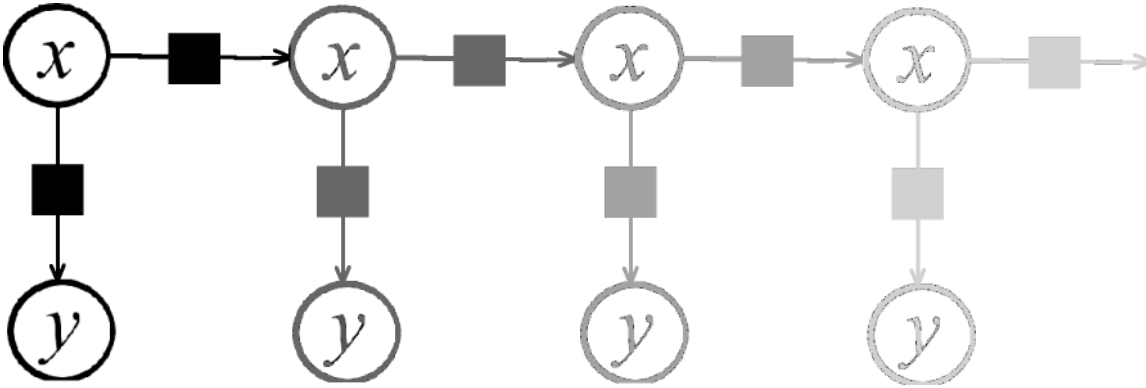
Exact inference in probabilistic graphical models

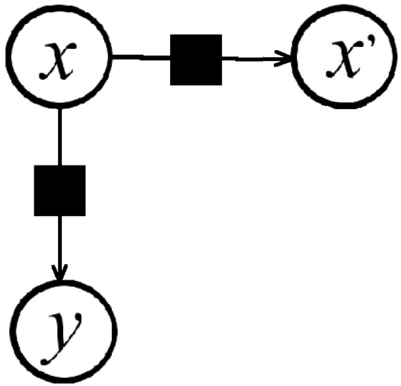
Marcus Frean
Victoria University of Wellington



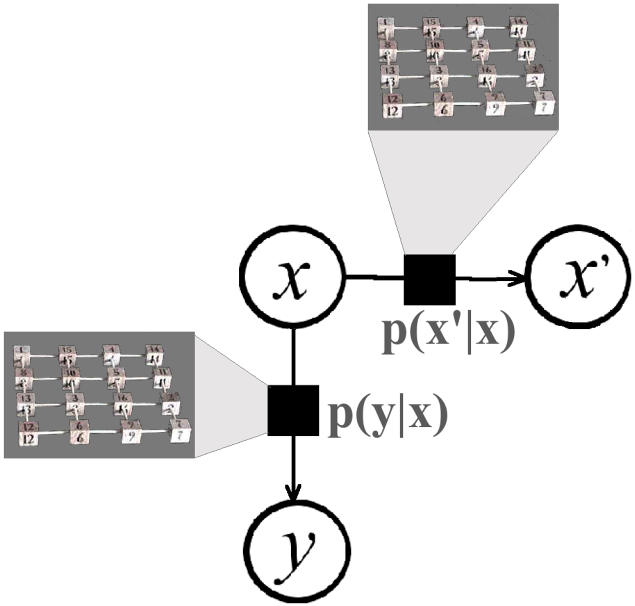
Hidden Markov models



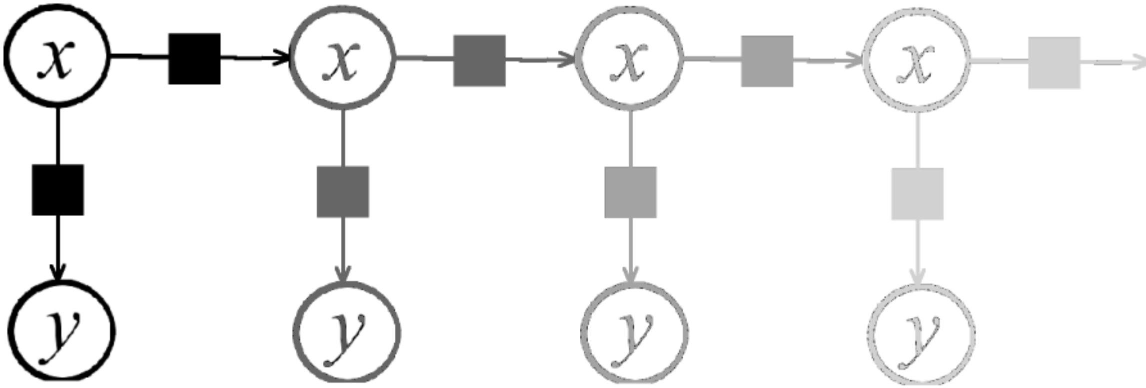
Hidden Markov models



Hidden Markov models



Hidden Markov models

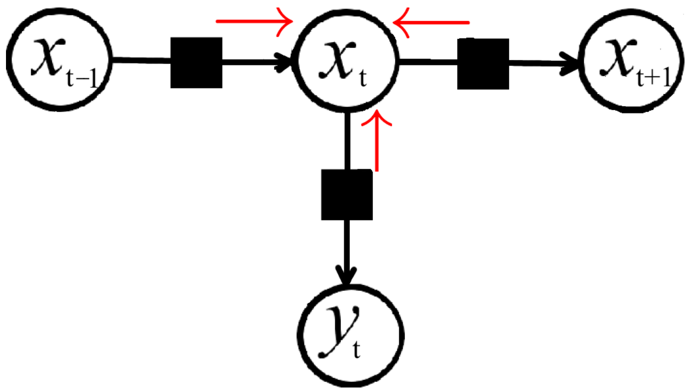


- we want posterior probabilities: $p(x_t|\mathbf{y}_{1..T})$
- we want better numbers to put in the squares...

messages

$$p(x_t|\mathbf{y}_{1..T}) \quad \propto \quad \underbrace{p(x_t, \mathbf{y}_{1..t-1})}_{\rightarrow} \cdot \underbrace{p(y_t|x_t)}_{\uparrow} \cdot \underbrace{p(\mathbf{y}_{t+1..T}|x_t)}_{\leftarrow}$$

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

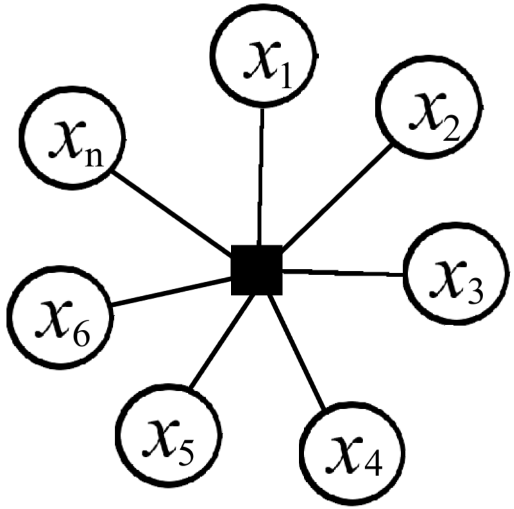


✲ $\rightarrow$ $\uparrow$ $\leftarrow$ generated by "forward" and "backward" recursions

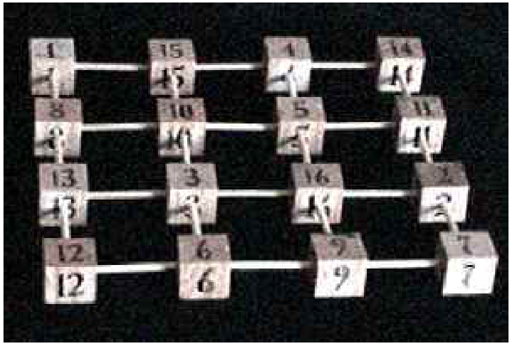
sums

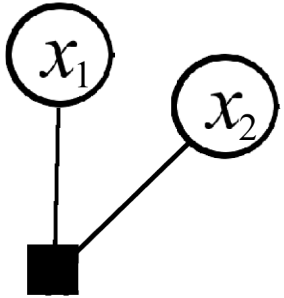
$$F(x_1, x_2, \dots x_n)$$

Function of n variables x, that can take k distinct values

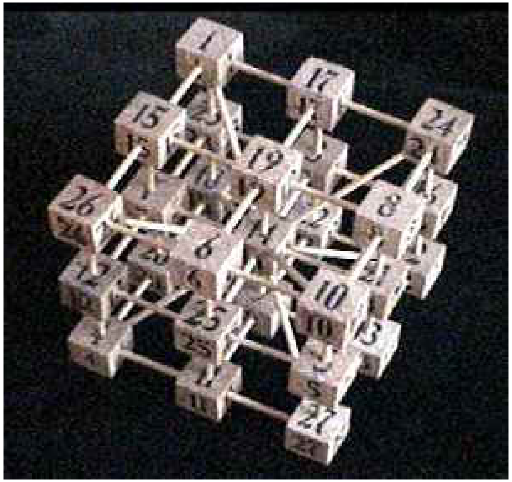


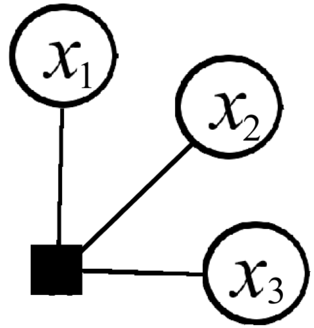
sums



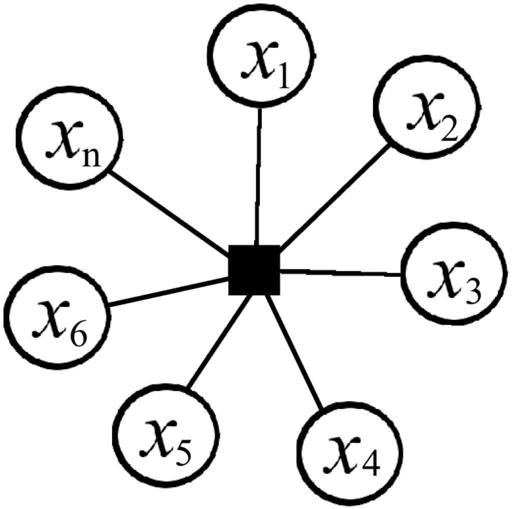


sums





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

sums of products

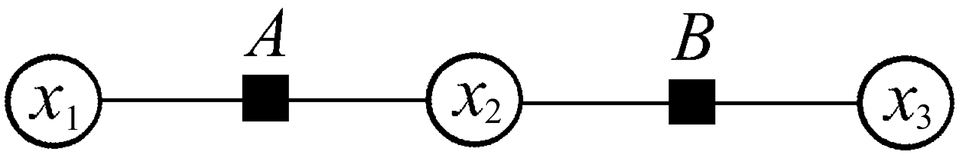
Suppose F “breaks up” into a product, *e.g.*

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

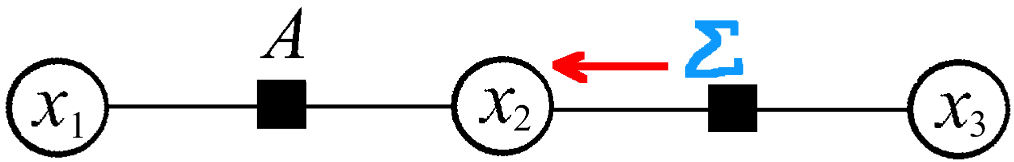
Now,

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

- ✲ this buys tractability
- ✲ think of it as a graph



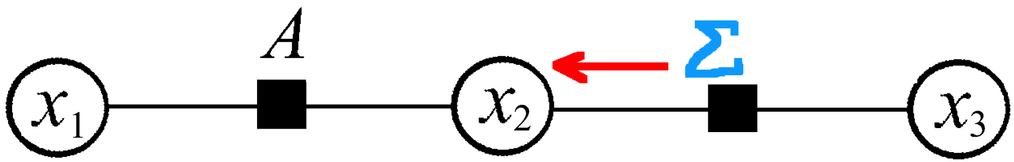
sums of products by message passing (1)



$$\sum_{x_1}\sum_{x_2}\sum_{x_3} F \;=\; \sum_{x_1}\sum_{x_2}\Phi_A(x_1,x_2)\;\underbrace{\sum_{x_3}\Phi_B(x_2,x_3)}_{\text{do this first}}$$

➤ this leaves a vector having the dimensions of x_2

sums of products by message passing (1)

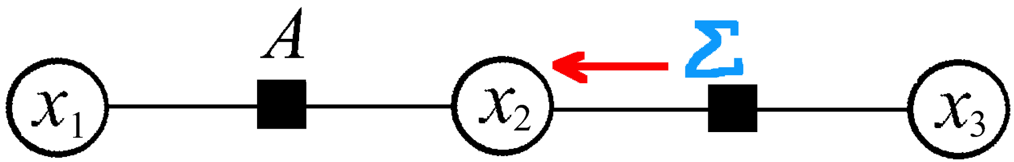


$$\sum_{x_1}\sum_{x_2}\sum_{x_3} F \;=\; \sum_{x_1}\sum_{x_2} \Phi_A(x_1, x_2) \underbrace{\sum_{x_3} \Phi_B(x_2, x_3)}_{\text{do this first}}$$

➤ this leaves a vector having the dimensions of x_2

➤ think of result as a message going off to x_2

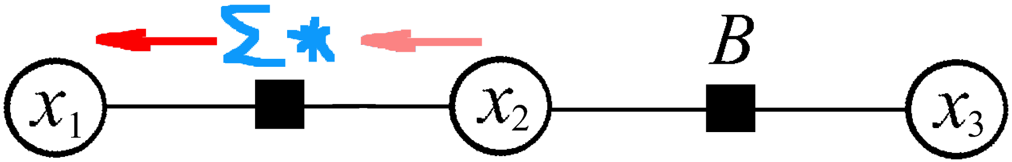
sums of products by message passing (1)



$$\sum_{x_1}\sum_{x_2}\sum_{x_3} F \;=\; \sum_{x_1}\;\sum_{x_2}\Phi_A(x_1,x_2)\;\underbrace{\sum_{x_3}\Phi_B(x_2,x_3)}_{\text{do this first}}$$

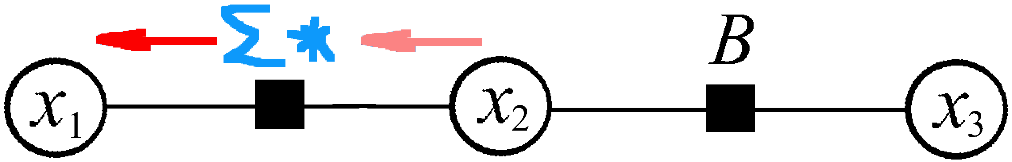
- this leaves a vector having the dimensions of x_2
- think of result as a message going off to x_2
- x_2 simply passes it on to A

sums of products by message passing (2)



$$\sum_{x_1}\sum_{x_2}\sum_{x_3} F \quad = \quad \sum_{x_1} \underbrace{\sum_{x_2} \Phi_A(x_1, x_2) \;\cdot\; \mathrm{msg}_{2\to A}(x_2)}_{\text{do this next}}$$

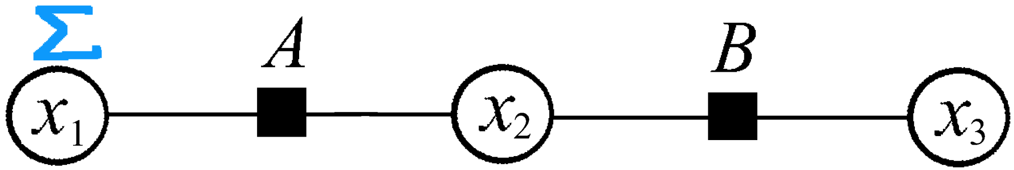
sums of products by message passing (2)



$$\sum_{x_1}\sum_{x_2}\sum_{x_3} F = \sum_{x_1} \underbrace{\sum_{x_2} \Phi_A(x_1, x_2) \cdot \mathrm{msg}_{2\to A}(x_2)}_{\text{do this next}}$$

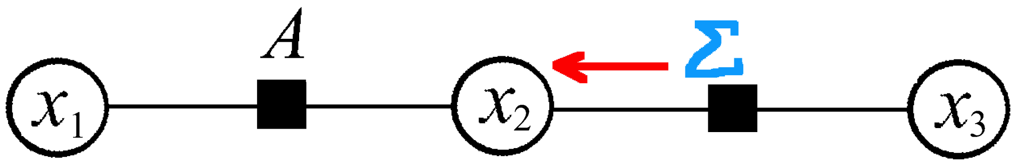
➤ this leaves a vector having the dimensions of x_1

sums of products by message passing (3)



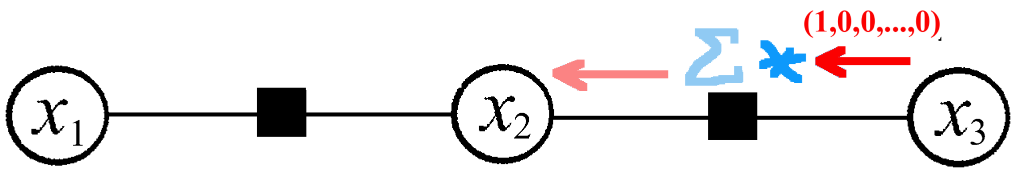
$$\sum_{x_1}\sum_{x_2}\sum_{x_3} F \quad = \quad \underbrace{\sum_{x_1} \mathrm{msg}_{A\to 1}(x_1)}_{\text{do this last}}$$

What if we fix (say) x_3=1?



$$\sum_{x_1}\sum_{x_2}\sum_{x_3} F \quad = \quad \sum_{x_1}\ \sum_{x_2} \Phi_A(x_1, x_2)\ \underbrace{\sum_{x_3} \Phi_B(x_2, x_3)}_{\text{but } x_3 = 1\ !}$$

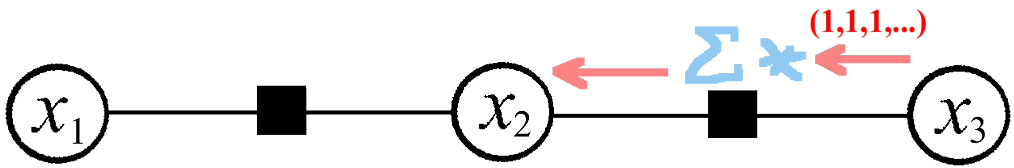
revise the first step...



$$\sum_{x_1}\sum_{x_2}\sum_{x_3} F = \sum_{x_1}\sum_{x_2}\Phi_A(x_1,x_2)\underbrace{\sum_{x_3}\Phi_B(x_2,x_3)\delta(x_3=1)}_{\text{do this first}}$$

- ➤ a new message coming from x_3 to B, consisting of zeros, except for the one we want to specify a value for
- ➤ multiply Φ_B by the incoming message, and sum the result over the dimension corresponding to x_2

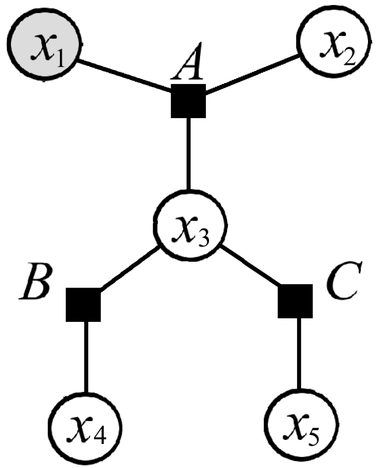
step 1 again...



- 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

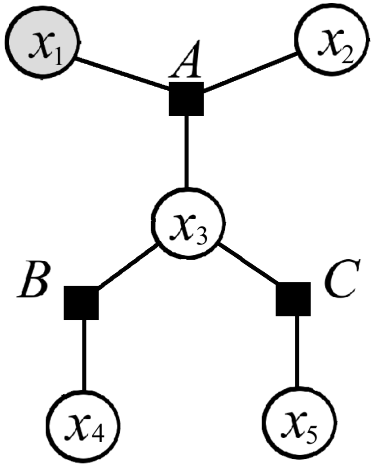
$$F(x_1, x_2, x_3, x_4, x_5) \quad = \quad \Phi_A(x_1, x_2, x_3)\ \Phi_B(x_3, x_4)\ \Phi_C(x_4, x_5)$$



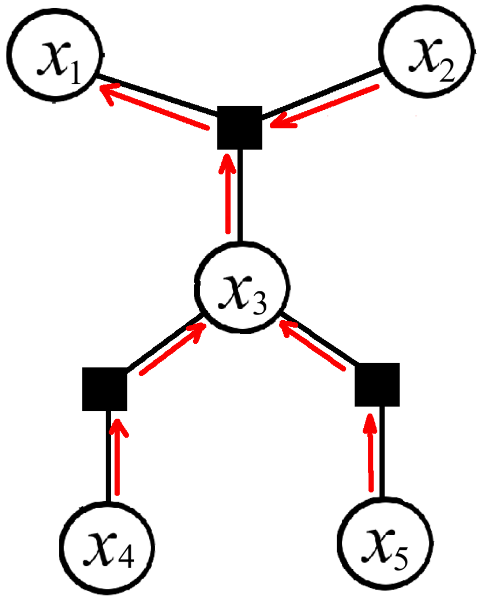
$$\sum_{x_1 \ldots x_5} F \quad = \quad \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} \ \Phi_A(x_1, x_2, x_3)\ \Phi_B(x_3, x_4)\ \Phi_C(x_4, x_5)$$

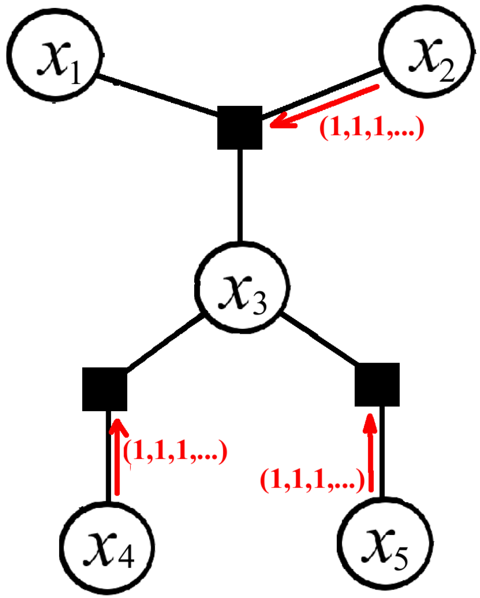
another example (cont)

$$F(x_1, x_2, x_3, x_4, x_5) \quad = \quad \Phi_A(x_1, x_2, x_3)\ \Phi_B(x_3, x_4)\ \Phi_C(x_4, x_5)$$

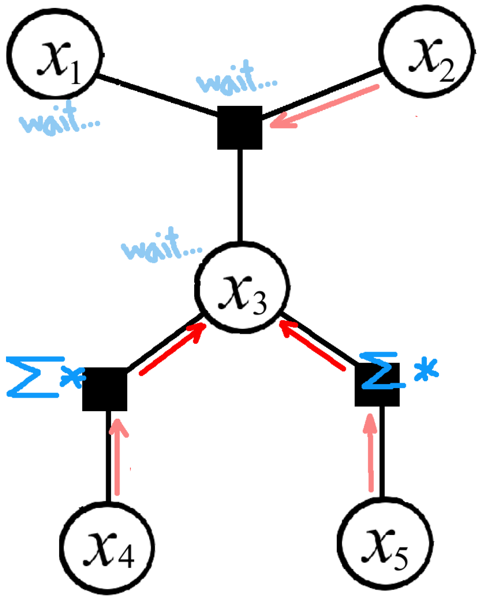


$$\sum_{x_1 \ldots x_5} F \;=\; \sum_{x_1} \;\; \sum_{x_2} \sum_{x_3} \Phi_A(x_1, x_2, x_3) \; \sum_{x_4} \Phi_B(x_3, x_4) \; \sum_{x_5} \Phi_C(x_4, x_5)$$

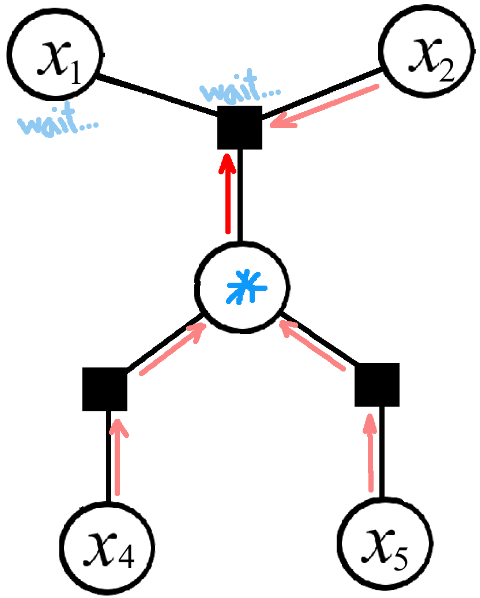

x_1
x_2
x_3
x_4
x_5



☼ unless we're prescribing values, variable nodes send "ones"

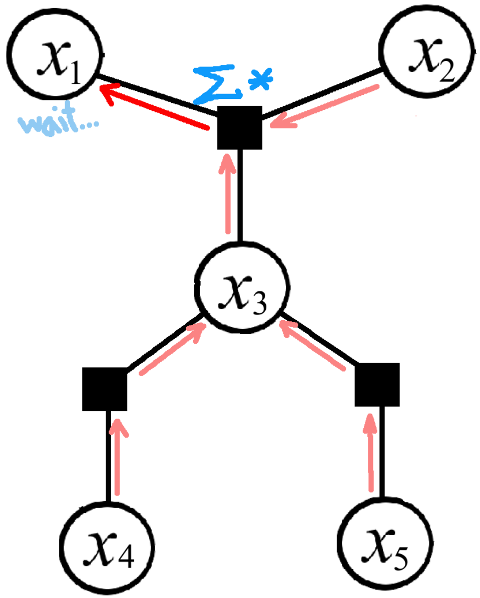


$$\sum_{x_1 \ldots x_5} F \;=\; \sum_{x_1} \;\; \sum_{x_2} \sum_{x_3} \Phi_A(x_1, x_2, x_3) \underbrace{\sum_{x_4} \Phi_B(x_3, x_4)}_{\text{done by } B} \underbrace{\sum_{x_5} \Phi_C(x_4, x_5)}_{\text{done by } C}$$



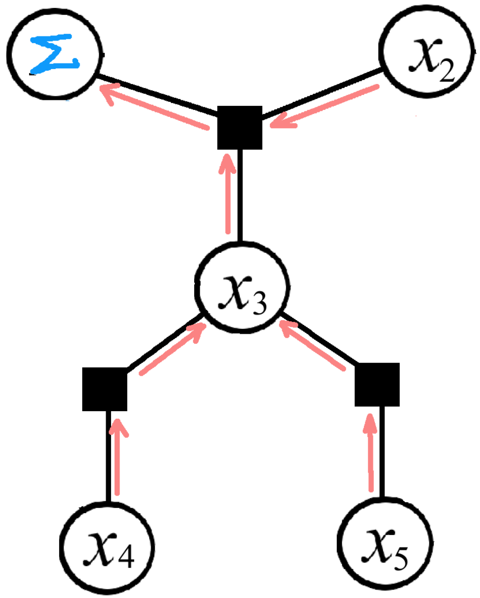
$$\sum_{x_1 \ldots x_5} F = \sum_{x_1} \sum_{x_2} \sum_{x_3} \Phi_A(x_1, x_2, x_3) \underbrace{\mathrm{msg}_{B \to 3}(x_3)\, \mathrm{msg}_{C \to 3}(x_3)}_{\text{done by node } x_3}$$

✲ node 3 *multiplies* its incoming messages together, and sends off the result to A



$$\sum_{x_1 \ldots x_5} F = \sum_{x_1} \underbrace{\sum_{x_2}\sum_{x_3} \Phi_A(x_1, x_2, x_3) \cdot \mathrm{msg}_{3\to A}(x_3) \cdot \mathrm{msg}_{2\to A}(x_2)}_{\text{done by } A}$$

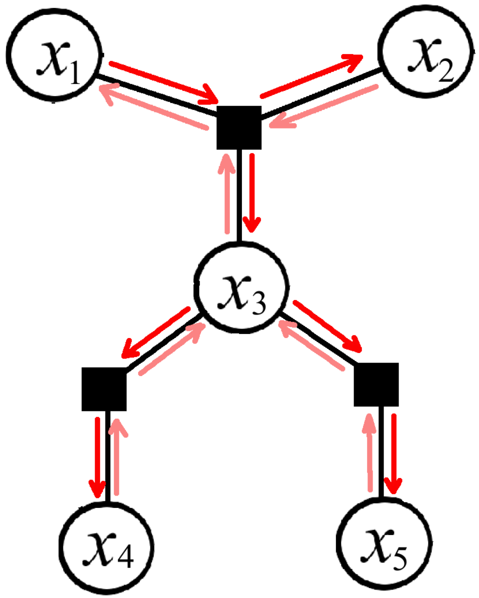
✲ just like before, except it multiplies by *both* incoming messages, and sums out along *both* those dimensions



Σ
x_2
x_3
x_4
x_5

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

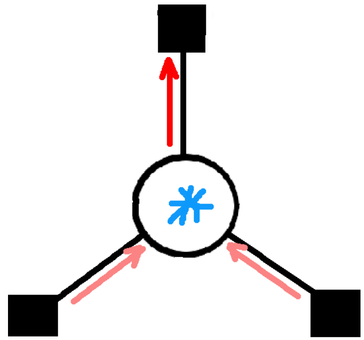
- messages are sent to, and from, each node
- messages always consist of a function over the associated *variable*
- each outgoing message is always a simple function of incoming messages on all the *other* edges

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 $(1,1,\ldots,1)$ by the incoming messages on all other edges



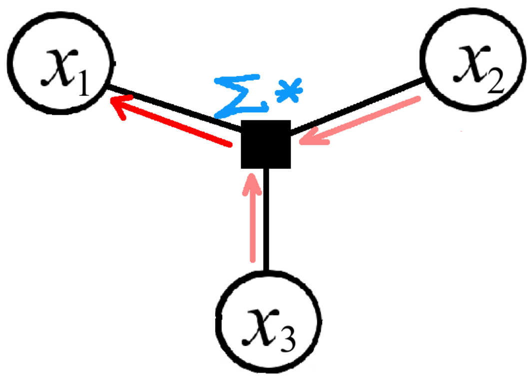
- if the variable is to be **preset** ("observed") send a one at the observed value only, as in $(0,1,0,\ldots,0)$

factor nodes are SUMMERS

Each dimension of Φ corresponds to one of its neighbours.

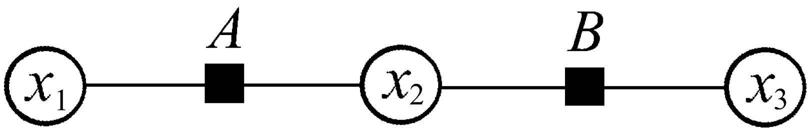
To generate a message on an edge,

1. multiply Φ 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*

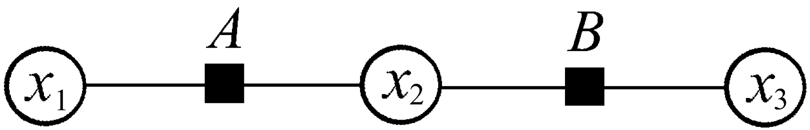


$$p(x_1, x_2, x_3) = \frac{1}{Z}\ \Phi_A(x_1, x_2)\ \Phi_B(x_2, x_3)$$

- 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” $\equiv$ sum-product algorithm, without the final summation!

belief propagation

- "belief propagation" $\equiv$ 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" $\equiv$ 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.

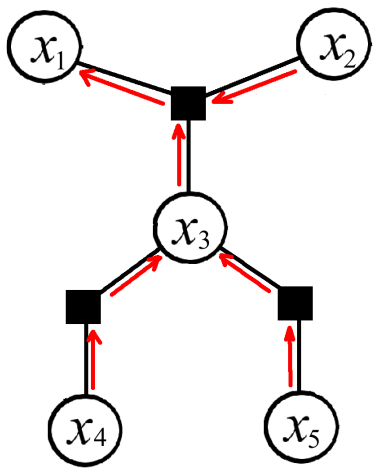
```
# First we define the variables
b = Multiplier('burglar',2)
e = Multiplier('earthquake',2)
a = Multiplier('alarm',2)

# 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]
```

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.*

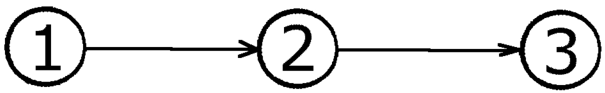
$$p(x_1, x_2, x_3) \;=\; p(x_3|x_2)\;\; p(x_2|x_1)\;\; p(x_1)$$

directed graphs: belief nets

Factorisations arrived at by the product rule, *e.g.*

$$p(x_1, x_2, x_3) \quad = \quad p(x_3|x_2) \;\; p(x_2|x_1) \;\; p(x_1)$$

Belief net:

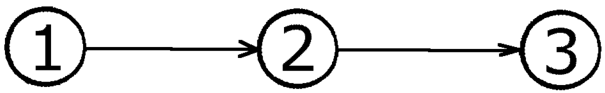


directed graphs: belief nets

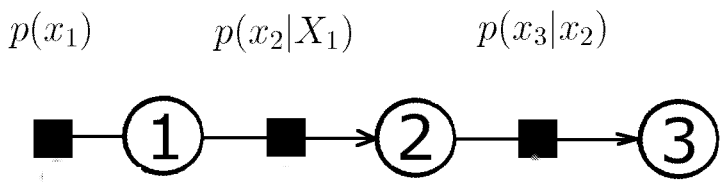
Factorisations arrived at by the product rule, *e.g.*

$$p(x_1, x_2, x_3) = p(x_3|x_2)\ p(x_2|x_1)\ p(x_1)$$

Belief net:



Factor graph:



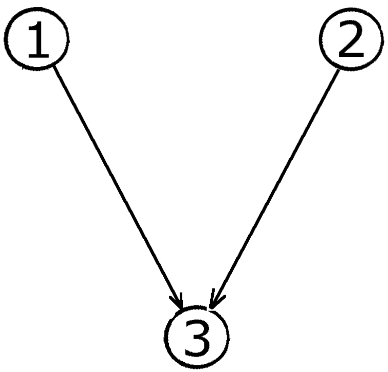
So far so good, but this graph has no more "expressive power" than a Markov Random Field

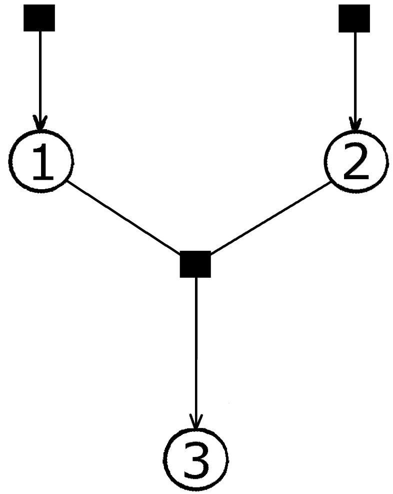
another directed graph: “explaining away”

$$p(x_1, x_2, x_3) = p(x_1)\ p(x_2)\ p(x_3|x_1, x_2)$$

another directed graph: “explaining away”

$$p(x_1, x_2, x_3) \quad = \quad p(x_1) \;\; p(x_2) \;\; p(x_3|x_1, x_2)$$



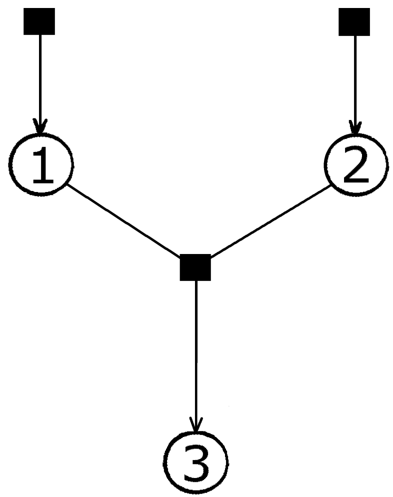


Structural assumption: x_1 is **independent** of x_2, but **becomes dependent** once x_3 is observed.

✲ crucial to inferences about causality.

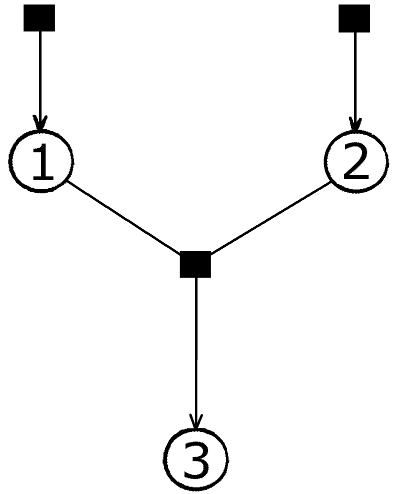
what do arrows mean?

- 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:

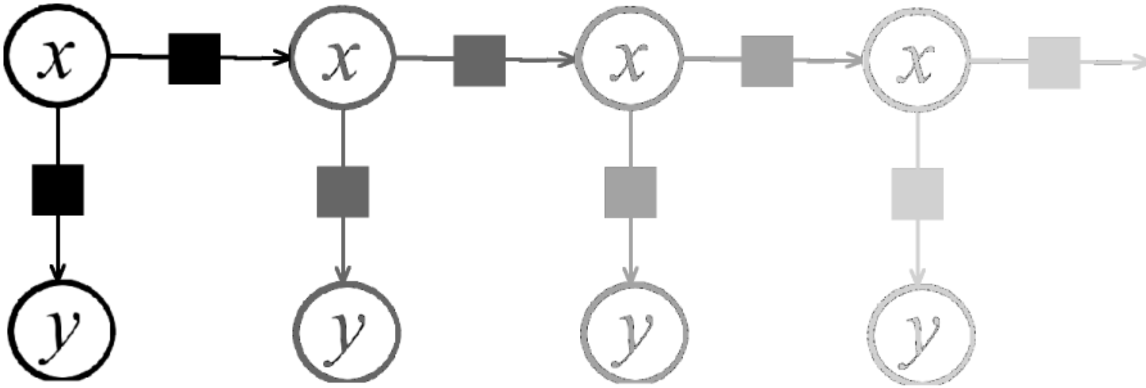
- messages passing in the 'forward' direction are of the form

$$p(x, \mathrm{obs}^{\mathrm{behind}})$$

- messages passing in the 'backward' direction are of the form

$$p(\mathrm{obs}^{\mathrm{ahead}} | x)$$

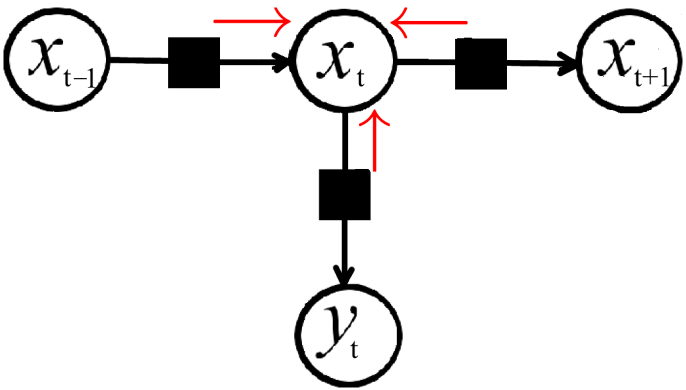
e.g. Hidden Markov models



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



$$p(x_t|\mathbf{y}_{1..T}) \quad \propto \quad \underbrace{p(x_t, \mathbf{y}_{1..t-1})}_{\rightarrow} \cdot \underbrace{p(y_t|x_t)}_{\uparrow} \cdot \underbrace{p(\mathbf{y}_{t+1..T}|x_t)}_{\leftarrow}$$

can we learn the factors from data?

- yes, via EM in belief nets (and therefore HMMs)

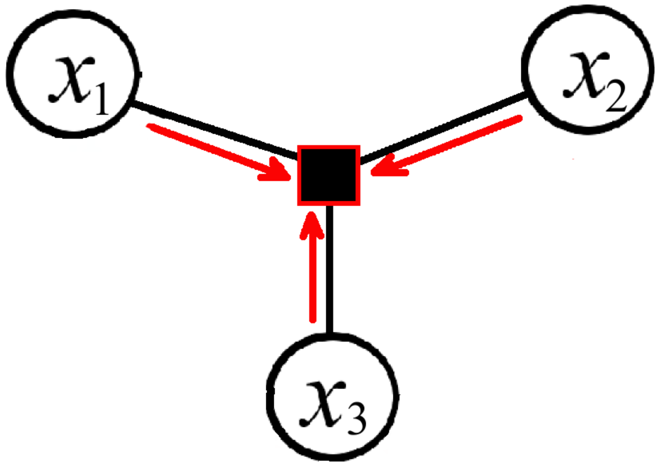
can we learn the factors from data?

- 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)

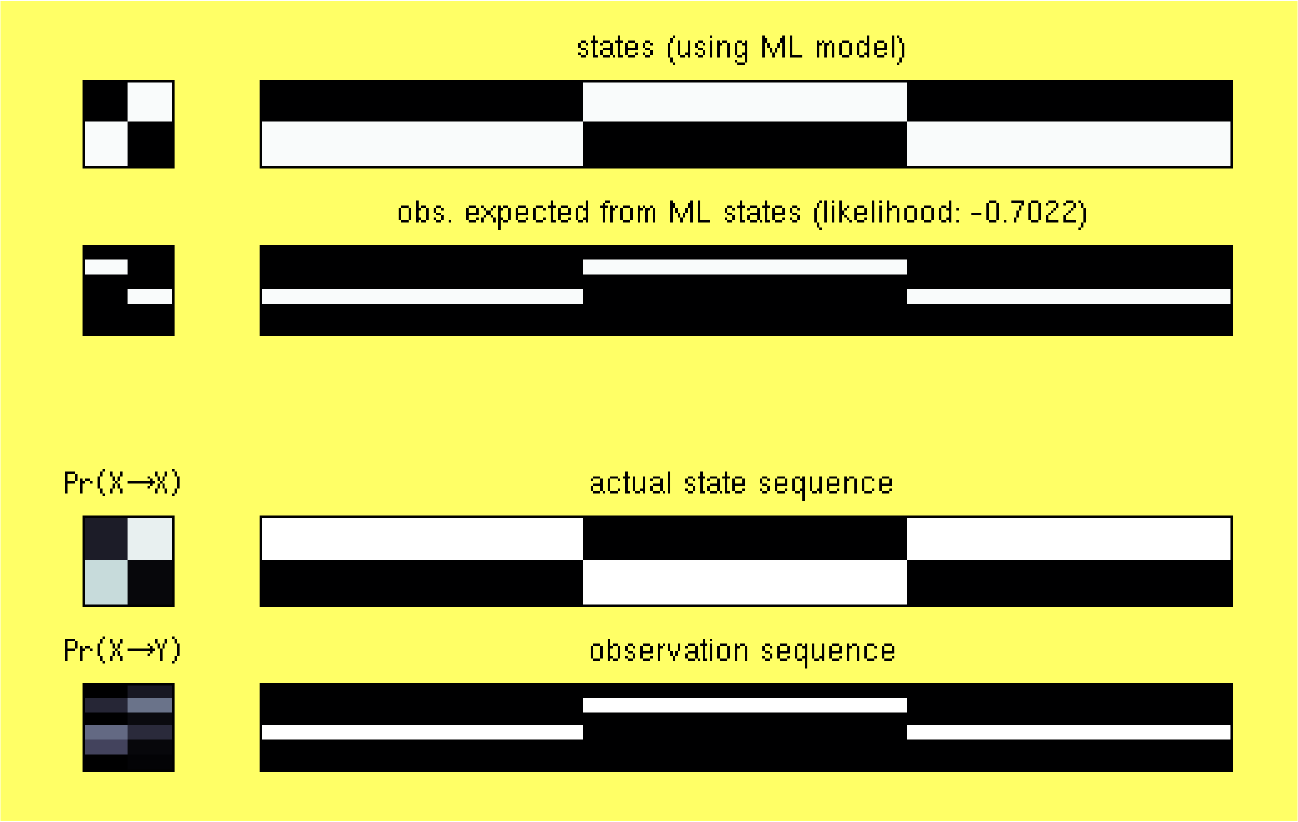
EM algorithm in belief nets

- E STEP:
  run belief propagation to get all the messages
- M STEP:
  Φ' is product* of Φ and all the incoming messages



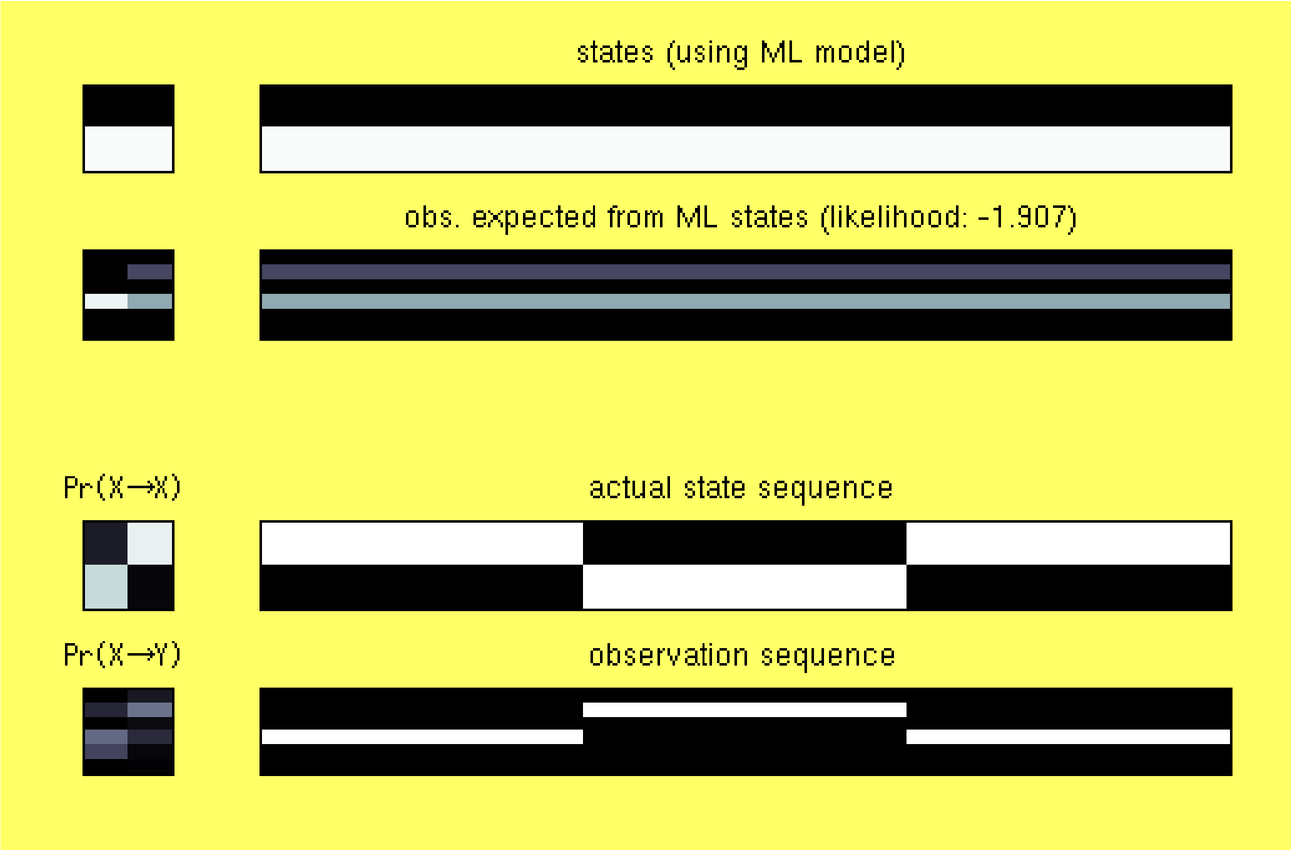
This increases a lower bound on the log likelihood, and (sometimes) works tolerably well in HMMs...

EM example 1



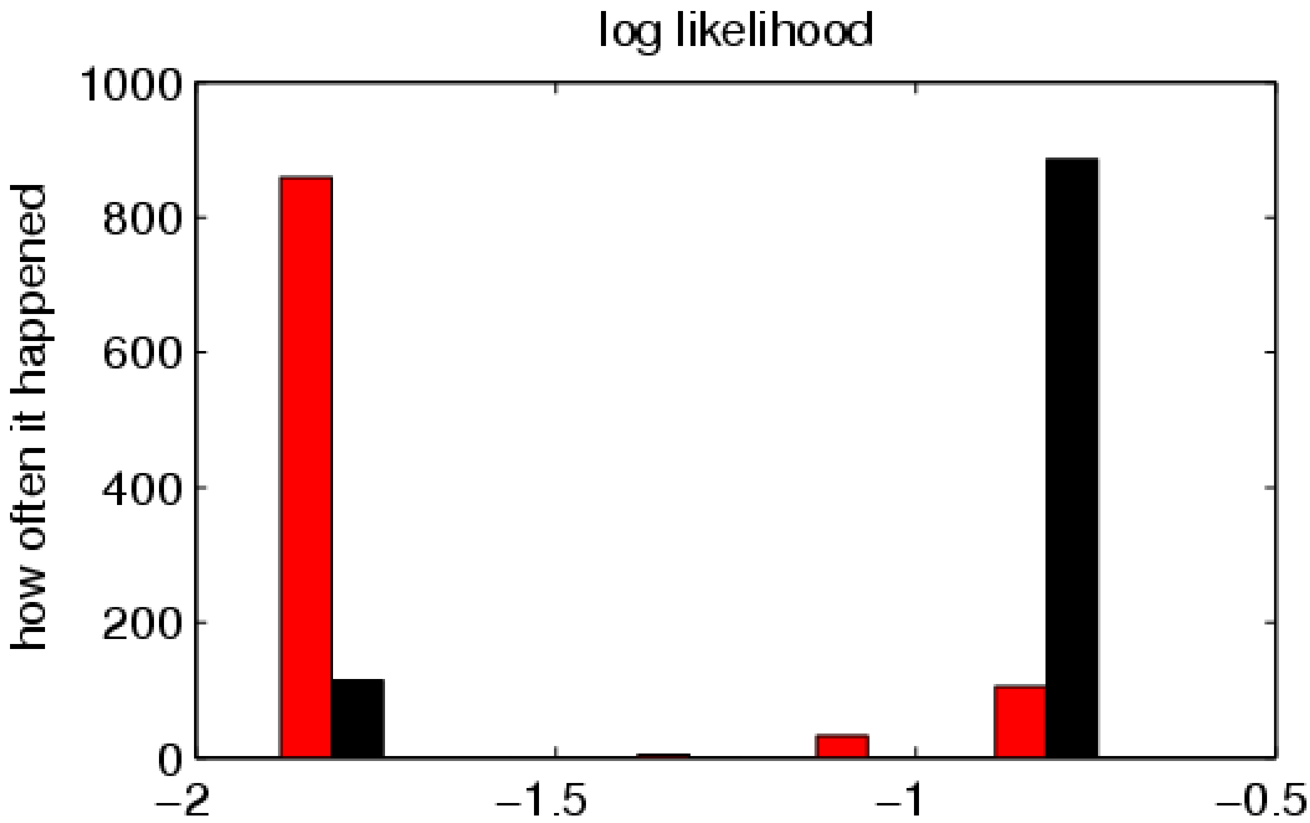
Right

EM example 1



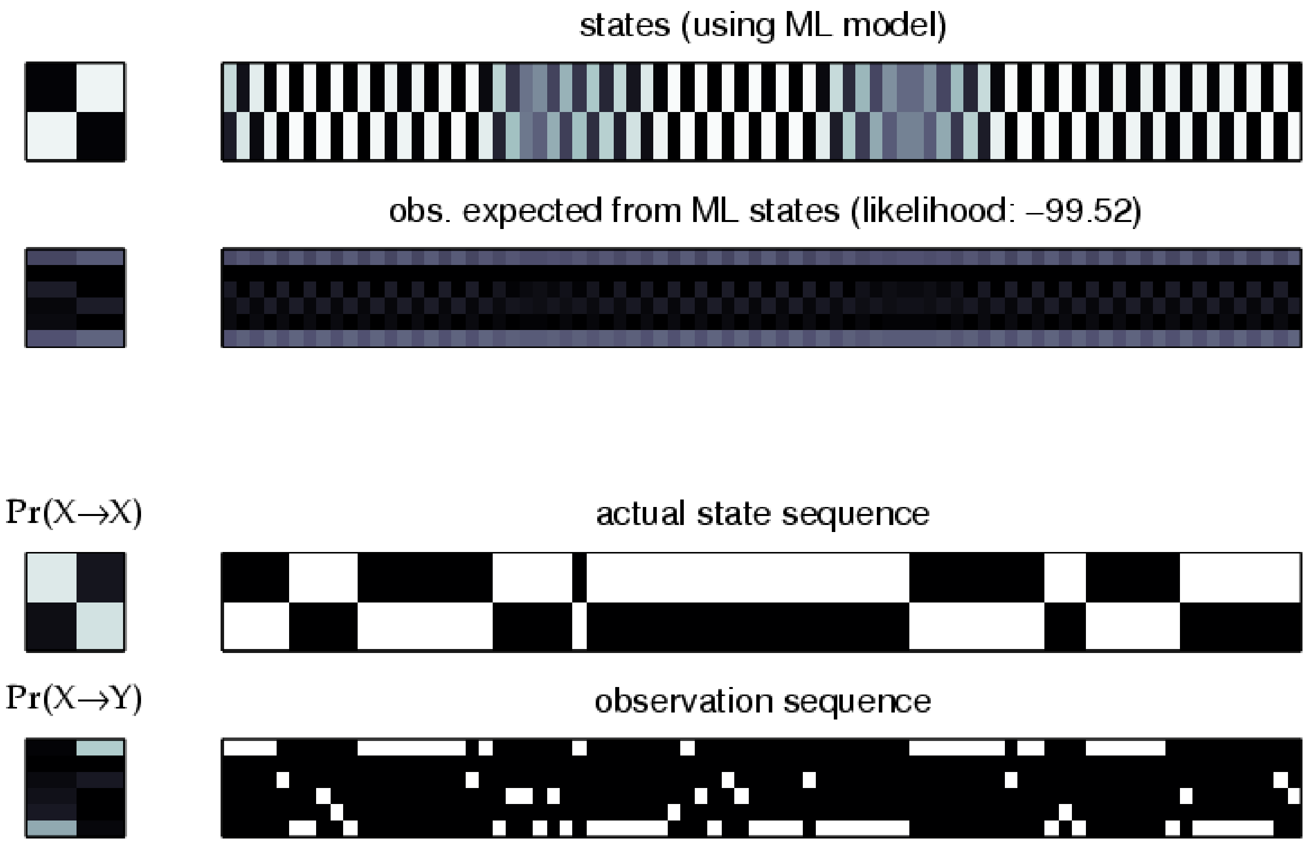
Wrong

EM example 1



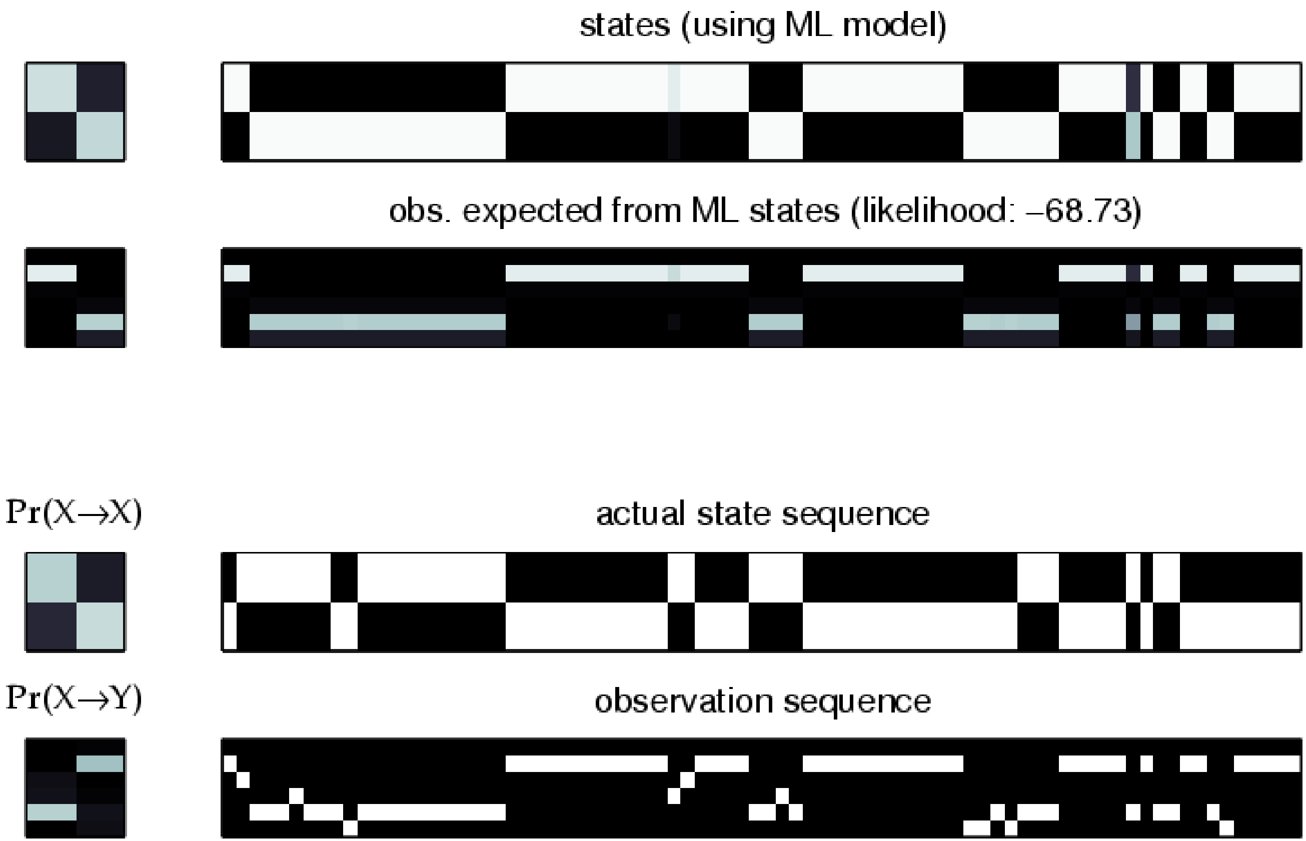
Red is EM, black is a stupid stochastic gradient ascender

EM example 2



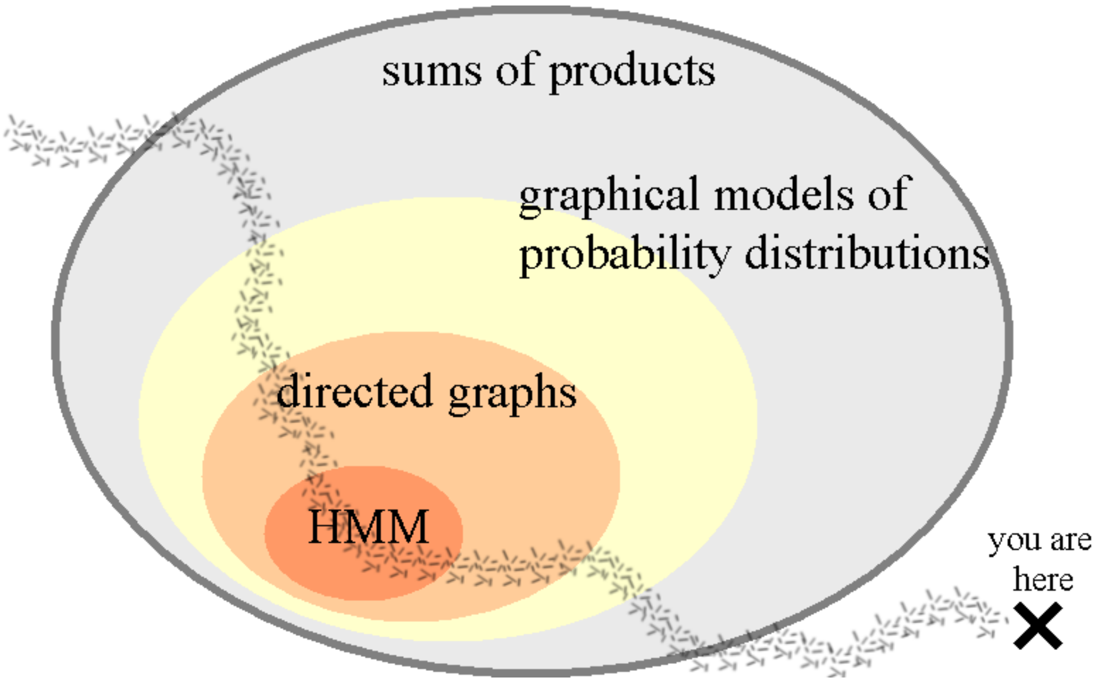
This is silly...

EM example 2



That's not bad...

questions?


sums of products
graphical models of probability distributions
directed graphs
HMM
you are here