

LECTURE 4:

INFERENCE IN CHAINS AND TREES, MESSAGE PASSING, BELIEF PROPAGATION

Sam Roweis

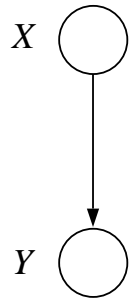
Tuesday July 25, 2006
Machine Learning Summer School, Taiwan

PROBABILISTIC INFERENCE

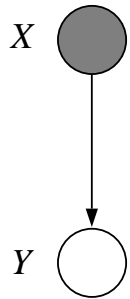
- Partition the random variables in a domain \mathbf{X} into three disjoint subsets, $\mathbf{x}_E, \mathbf{x}_F, \mathbf{x}_R$. The general *probabilistic inference* problem is to compute the posterior $p(\mathbf{x}_F | \mathbf{x}_E)$ over *query nodes* \mathbf{x}_F .
- This involves *conditioning* on *evidence nodes* \mathbf{x}_E and *integrating (summing) out marginal nodes* \mathbf{x}_R .
- If the joint distribution is represented as a huge table, this is trivial: just select the appropriate indicies in the columns corresponding to \mathbf{x}_E based on the values, sum over the columns corresponding to \mathbf{x}_R , and renormalize the resulting table over \mathbf{x}_F .
- If the joint is a known continuous function this can sometimes be done analytically. (e.g. Gaussian: eliminate rows/cols corresponding to \mathbf{x}_R ; apply conditioning formulas for $p(\mathbf{x}_F | \mathbf{x}_E)$).
- But what if the joint distribution over \mathbf{X} is represented by a directed or undirected graphical model?

SIMPLE CASE: BAYES RULE

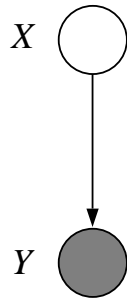
- For simple models, we can derive the inference formulas by hand using Bayes rule (e.g. responsibility in mixture models).



(a)



(b)



(c)

$$a) \quad p(x, y) = p(x)p(y|x)$$

$$b) \quad p(y|x)$$

$$c) \quad p(x|y) = \frac{p(x)p(y|x)}{\sum_x p(x)p(y|x)}$$

This is called “reversing the arrow”.

- In general, the calculation we want to do is:

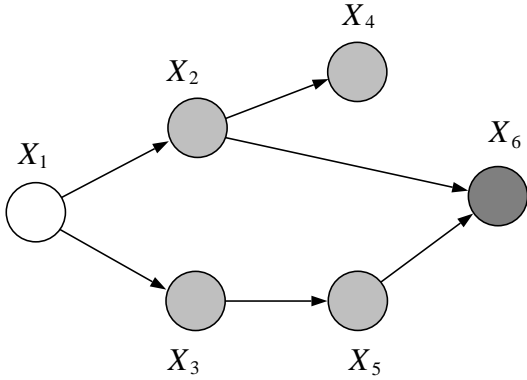
$$p(\mathbf{x}_F | \mathbf{x}_E) = \frac{\sum_{\mathbf{x}_R} p(\mathbf{x}_E, \mathbf{x}_F, \mathbf{x}_R)}{\sum_{\mathbf{x}_F, \mathbf{x}_R} p(\mathbf{x}_E, \mathbf{x}_F, \mathbf{x}_R)}$$

- Q: Can we do these sums efficiently?

Can we avoid repeating unnecessary work each time we do inference?

A: Yes, if we exploit the factorization of the joint distribution.

EXAMPLE



The key is to factor and then apply the distributive law.

$$\begin{aligned}
 p(\mathbf{x}_1 | \bar{\mathbf{x}}_6) &= p(\mathbf{x}_1, \bar{\mathbf{x}}_6) / p(\bar{\mathbf{x}}_6) \\
 &= p(\mathbf{x}_1, \bar{\mathbf{x}}_6) / \sum_{\mathbf{x}_1'} p(\mathbf{x}_1', \bar{\mathbf{x}}_6)
 \end{aligned}$$

$$\begin{aligned}
 p(\mathbf{x}_1, \bar{\mathbf{x}}_6) &= \sum_{\mathbf{x}_2} \sum_{\mathbf{x}_3} \sum_{\mathbf{x}_4} \sum_{\mathbf{x}_5} p(\mathbf{x}_1) p(\mathbf{x}_2 | \mathbf{x}_1) p(\mathbf{x}_3 | \mathbf{x}_1) p(\mathbf{x}_4 | \mathbf{x}_2) p(\mathbf{x}_5 | \mathbf{x}_3) p(\bar{\mathbf{x}}_6 | \mathbf{x}_2, \mathbf{x}_5) \\
 &= p(\mathbf{x}_1) \sum_{\mathbf{x}_2} p(\mathbf{x}_2 | \mathbf{x}_1) \sum_{\mathbf{x}_3} p(\mathbf{x}_3 | \mathbf{x}_1) \sum_{\mathbf{x}_4} p(\mathbf{x}_4 | \mathbf{x}_2) \sum_{\mathbf{x}_5} p(\mathbf{x}_5 | \mathbf{x}_3) p(\bar{\mathbf{x}}_6 | \mathbf{x}_2, \mathbf{x}_5) \\
 &= p(\mathbf{x}_1) \sum_{\mathbf{x}_2} p(\mathbf{x}_2 | \mathbf{x}_1) \sum_{\mathbf{x}_3} p(\mathbf{x}_3 | \mathbf{x}_1) \Phi_5(\mathbf{x}_2, \mathbf{x}_3) \sum_{\mathbf{x}_4} p(\mathbf{x}_4 | \mathbf{x}_2) \\
 &= p(\mathbf{x}_1) \sum_{\mathbf{x}_2} p(\mathbf{x}_2 | \mathbf{x}_1) \Phi_4(\mathbf{x}_2) \sum_{\mathbf{x}_3} p(\mathbf{x}_3 | \mathbf{x}_1) \Phi_5(\mathbf{x}_2, \mathbf{x}_3) \\
 &= p(\mathbf{x}_1) \sum_{\mathbf{x}_2} p(\mathbf{x}_2 | \mathbf{x}_1) \Phi_4(\mathbf{x}_2) \Phi_3(\mathbf{x}_1, \mathbf{x}_2) \\
 &= p(\mathbf{x}_1) \Phi_2(\mathbf{x}_1)
 \end{aligned}$$

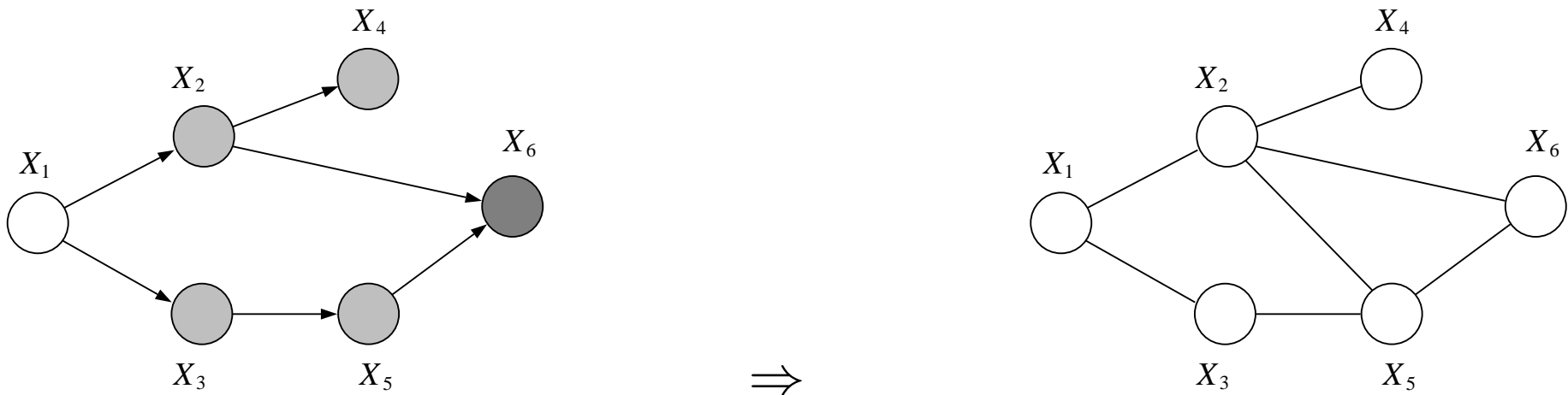
MARGINALIZATION WITHOUT EVIDENCE

- Marginalization of joint distributions represented by graphical models is a special case of probabilistic inference.
- To compute the marginal $p(x_i)$ of a single node, we set it to be the query node and set the evidence set to be empty.
- In directed models, we can ignore all nodes downstream from the query node, and marginalize only the part of the graph before it.
- If the node has no parents, we can read off its marginal directly.
- In directed models, we often know that a certain sum must evaluate to unity, since it is a conditional probability.
- For example, consider the term $\Phi_4(\mathbf{x}_2)$ in our six node example:

$$\Phi_4(\mathbf{x}_2) = \sum_{\mathbf{x}_4} p(\mathbf{x}_4|\mathbf{x}_2) \equiv 1$$

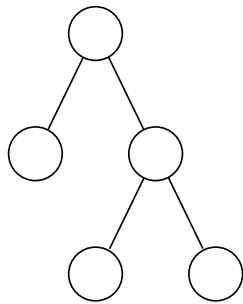
MORALIZATION

- For directed graphs, the parents may not be explicitly connected, but they are involved in the same potential function $p(x_i | \mathbf{x}_{\pi_i})$.
- Thus to correctly account for all the dependencies, we first must connect all the parents of every node and then drop the directions on the links.
- This step is known as “Moralization” and it is essential: since conditioning couples parents in directed models (“explaining away”) we need a mechanism for respecting this when we do inference.

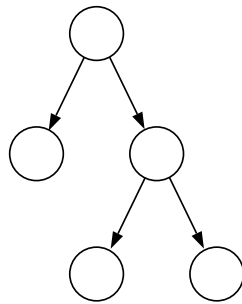


EFFICIENT INFERENCE ON TREE STRUCTURED DAGS

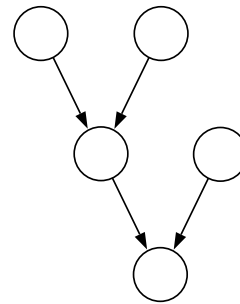
- We want to develop inference algorithms which are correct and efficient when we perform multiple queries (e.g. during learning with latent variables and EM).
- For now, we will focus on tree-structured graphical models, which include all two-node models and all chains as well.
- Exact inference on trees is the basis for the *junction tree algorithm* which solves the general exact inference problem for directed acyclic graphs and for many *approximate* algorithms which can work on intractable or cyclic graphs.



(a)



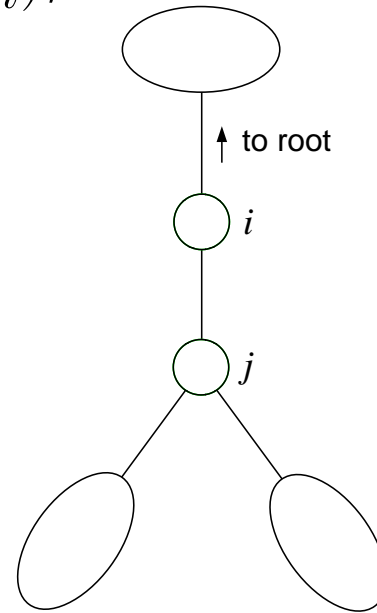
(b)



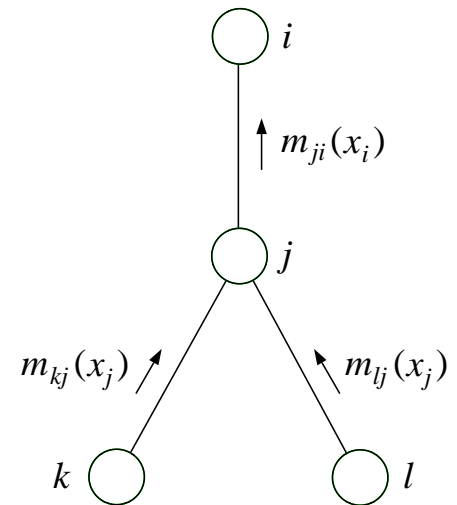
(c)

EFFICIENT SUMMATION ON TREES: LEAVES \rightarrow ROOT

- Consider summing out node j which has node i as its parent:
- Which nodes appear in the factors created by summing over j ?
 - nothing in the subtree below j (already summed out)
 - nothing from other subtrees, since the graph is a tree
 - only i , which relates i and j through $p(j|i)$
- Call the factor that is created $m_{ji}(x_i)$, and think of it as a *message* that j passes to i when j is summed.
- This message is created by summing over j the product of all earlier messages $m_{kj}(x_j)$ sent to j as well as $\delta(x_j = x_j^{obs})$ (if j is an evidence node).



(a)



(b)

INFERENCE = EFFICIENT SUMMATION = MESSAGE PASSING

- On a tree, inference can be thought of as passing messages up to the query node at the root from the other nodes at the leaves or interior. Since we ignore subtrees with no evidence, observed (evidence) nodes are always at the leaves.
- The message $m_{ji}(x_i)$ is created when we sum over x_j

$$m_{ji}(x_i) = \sum_{x_j} \left(\psi^E(x_j) p(x_j|x_i) \prod_{k \in c(j)} m_{kj}(x_j) \right)$$

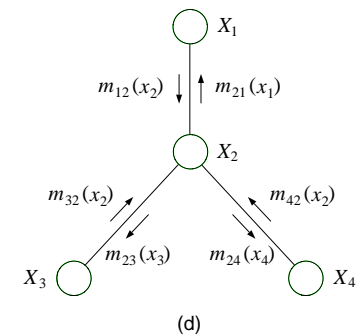
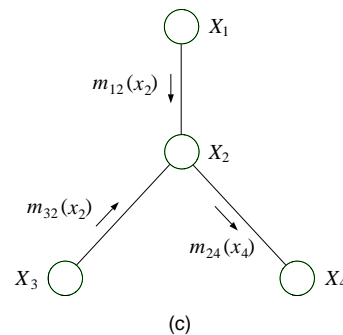
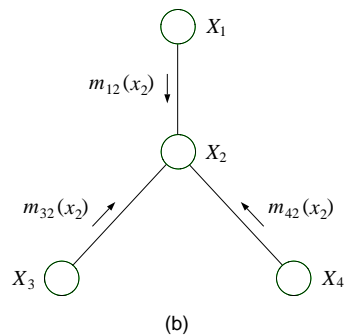
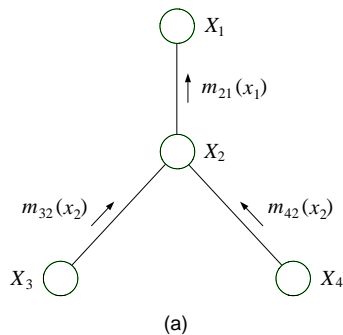
- At the final node x_f , we obtain the answer:

$$p(x_f|\bar{\mathbf{x}}_E) \propto \psi^E(x_f) \prod_{k \in c(f)} m_{kf}(x_f)$$

- If j is an evidence node, $\psi^E(x_j) = \delta(x_j, \bar{x}_j)$, else $\psi^E(x_j) = 1$.
- If j is a leaf node, $c(j)$ is empty, otherwise $c(j)$ are the children of j

MESSAGES ARE REUSED IN MULTIPLE QUERIES

- Consider querying x_1 , x_2 , x_3 and x_4 in the graph below.
- The messages needed for x_1 , x_2 , x_4 individually are shown (a-c).
- Also shown in (d) is the set of messages needed to compute all possible marginals over single query nodes.



- Key insight: even though the naive approach (redo the query from scratch) needs to compute N^2 messages to find marginals for all N query nodes, there are only $2N$ possible messages.
- We can compute all possible messages in only double the amount of work it takes to do one query!
- Then we take the product of relevant messages to get marginals.

COMPUTING ALL POSSIBLE MESSAGES

- How can we compute all possible messages efficiently?
- Idea: respect the following MESSAGE-PASSING-PROTOCOL:
A node can send a message to a neighbour only when it has received messages from all its other neighbours.
- Protocol is realizable: designate one node (arbitrarily) as the root. Collect messages inward to root then distribute back out to leaves.
- Once we have the messages, we can compute marginals using:

$$p(x_i | \bar{\mathbf{x}}_E) \propto \psi^E(x_i) \prod_{k \in c(i)} m_{ki}(x_i)$$

- Remember that the directed tree on which we pass messages might not be same directed tree we started with.
- We can also consider “synchronous” or “asynchronous” message passing nodes that respect the protocol but don’t use the Collect-Distribute schedule above. (Must prove this terminates.)

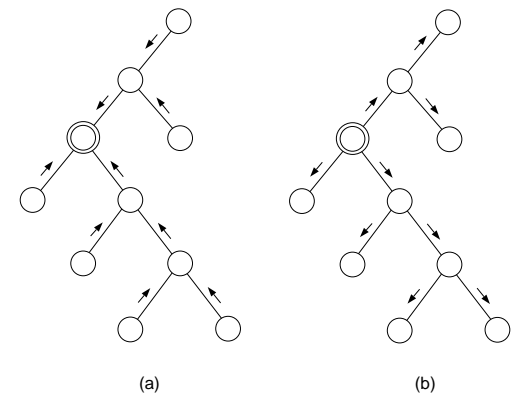
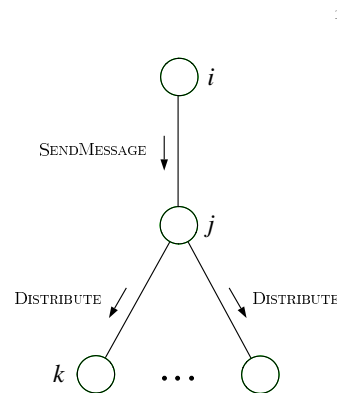
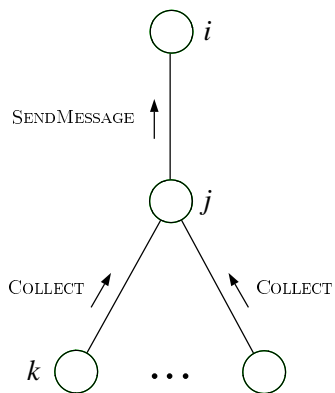
BELIEF PROPAGATION (SUM-PRODUCT) ALGORITHM

- Choose a root node (arbitrarily or as first query node).
- If j is an evidence node, $\psi^E(x_j) = \delta(x_j, \bar{x}_j)$, else $\psi^E(x_j) = 1$.
- Pass messages from leaves up to root and then back down using:

$$m_{ji}(x_i) = \sum_{x_j} \left(\psi^E(x_j) \psi(x_i, x_j) \prod_{k \in c(j)} m_{kj}(x_j) \right)$$

- Given messages, compute marginals using:

$$p(x_i | \bar{\mathbf{x}}_E) \propto \psi^E(x_i) \prod_{k \in c(i)} m_{ki}(x_i)$$



COMPUTING JOINT PAIRWISE POSTERiors

- We can also easily compute the joint pairwise posterior distribution for any pair of connected nodes x_i, x_j .
- To do this, we simply take the product of all messages coming into node i (*except the message from node j*), all the messages coming into node j (*except the message from node i*) and the potentials $\psi_i(x_i), \psi_j(x_j), \psi_{ij}(x_i, x_j)$.
- The posterior is proportional to this product:

$$p(x_i, x_j | \bar{\mathbf{X}}_E) \propto \psi^E(x_i) \psi^E(x_j) \psi(x_i, x_j) \prod_{k \neq j \in c(i)} m_{ki}(x_i) \prod_{\ell \neq i \in c(j)} m_{\ell j}(x_j)$$

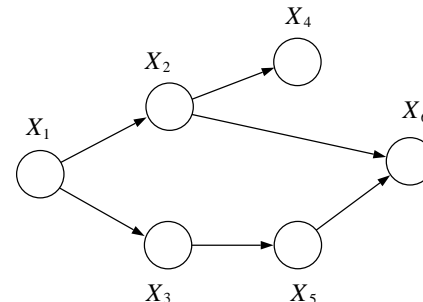
- These joint pairwise posteriors cover all the maximal cliques in the tree, and so those are all we need to do learning.
- Inference of other pairwise or higher order joint posteriors is possible, but more difficult.

MAXIMIZING INSTEAD OF SUMMING

- BELIEF PROPAGATION summed over all possible values of the marginal (non-query, non-evidence) nodes to get a marginal probability.
- What if we wanted to *maximize* over the non-query, non-evidence nodes to find the probability of the single best setting consistent with any query and evidence?

$$\begin{aligned}\max_{\mathbf{x}} p(\mathbf{x}) &= \max_{\mathbf{x}_1} \max_{\mathbf{x}_2} \max_{\mathbf{x}_3} \max_{\mathbf{x}_4} \max_{\mathbf{x}_5} p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_3|\mathbf{x}_1)p(\mathbf{x}_4|\mathbf{x}_2)p(\mathbf{x}_5|\mathbf{x}_3)p(\mathbf{x}_6|\mathbf{x}_2, \mathbf{x}_5) \\ &= \max_{\mathbf{x}_1} p(\mathbf{x}_1) \max_{\mathbf{x}_2} p(\mathbf{x}_2|\mathbf{x}_1) \max_{\mathbf{x}_3} p(\mathbf{x}_3|\mathbf{x}_1) \max_{\mathbf{x}_4} p(\mathbf{x}_4|\mathbf{x}_2) \max_{\mathbf{x}_5} p(\mathbf{x}_5|\mathbf{x}_3)p(\mathbf{x}_6|\mathbf{x}_2, \mathbf{x}_5)\end{aligned}$$

- This is known as the *maximum a-posteriori* or MAP configuration.
- It turns out that (on trees), we can use an algorithm exactly like belief-propagation to solve this problem.



HIDDEN MARKOV MODELS (HMMs)

Add a latent (hidden) variable x_t to a Markov model.

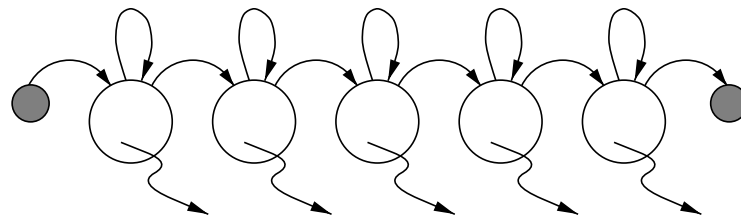
- HMM \equiv “probabilistic function of a Markov chain”:

1. 1st-order Markov chain generates hidden state sequence (path):

$$P(x_{t+1} = j | x_t = i) = S_{ij} \quad P(x_1 = j) = \pi_j$$

2. A set of output probability distributions $A_j(\cdot)$ (one per state) converts state path into sequence of observable symbols/vectors

$$P(\mathbf{y}_t = y | x_t = j) = A_j(y)$$



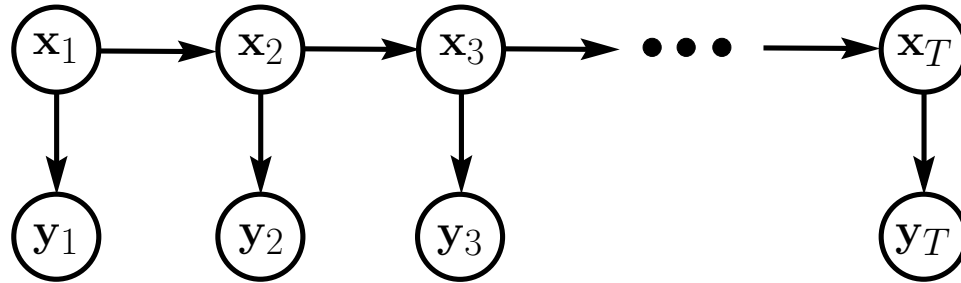
(state transition diagram)

- Even though hidden state seq. is 1st-order Markov, the output process is not Markov of *any* order [ex. 1111121111311121111131...]

APPLICATIONS OF HMMs

- Speech recognition.
- Language modeling.
- Information retrieval.
- Motion video analysis/tracking.
- Protein sequence and genetic sequence alignment and analysis.
- Financial time series prediction.
- ...

HMM GRAPHICAL MODEL



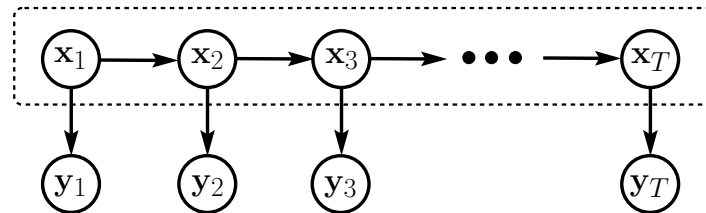
- Hidden states $\{x_t\}$, outputs $\{y_t\}$
Joint probability factorizes:

$$\begin{aligned} P(\{\mathbf{x}\}, \{\mathbf{y}\}) &= \prod_{t=1}^T P(x_t | \mathbf{x}_{t-1}) P(\mathbf{y}_t | x_t) \\ &= \pi_{\mathbf{x}_1} \prod_{t=1}^{T-1} S_{x_t, x_{t+1}} \prod_{t=1}^T A_{x_t}(\mathbf{y}_t) \end{aligned}$$

- NB: Data are *not* i.i.d.
There is no easy way to use plates to show this model. (Why?)

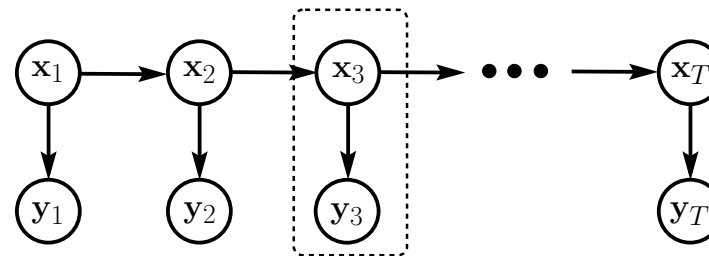
LINKS TO OTHER MODELS

- You can think of an HMM as:
A Markov chain with stochastic measurements.



or

A mixture model with states coupled across time.



- The future is independent of the past given the present.
However, conditioning on all the observations couples hidden states.

EXAMPLE: HMM LIKELIHOOD CALCULATION

- To evaluate the probability $P(\{\mathbf{y}\})$, we want:

$$P(\{\mathbf{y}\}) = \sum_{\{\mathbf{x}\}} P(\{\mathbf{x}\}, \{\mathbf{y}\})$$

$$P(\text{observed sequence}) = \sum_{\text{all paths}} P(\text{observed outputs}, \text{state path})$$

- Looks hard! ($\#\text{paths} = \#\text{states}^T$). But joint probability factorizes:

$$\begin{aligned} P(\{\mathbf{y}\}) &= \sum_{\mathbf{x}_1} \sum_{\mathbf{x}_2} \cdots \sum_{\mathbf{x}_T} \prod_{t=1}^T P(x_t | \mathbf{x}_{t-1}) P(\mathbf{y}_t | x_t) \\ &= \sum_{\mathbf{x}_1} P(\mathbf{x}_1) P(\mathbf{y}_1 | \mathbf{x}_1) \sum_{\mathbf{x}_2} P(\mathbf{x}_2 | \mathbf{x}_1) P(\mathbf{y}_2 | \mathbf{x}_2) \cdots \sum_{\mathbf{x}_T} P(\mathbf{x}_T | \mathbf{x}_{T-1}) P(\mathbf{y}_T | \mathbf{x}_T) \end{aligned}$$

- By moving the summations inside, we can save a lot of work.

THE FORWARD (α) RECURSION

- We want to compute:

$$L = P(\{\mathbf{y}\}) = \sum_{\{\mathbf{x}\}} P(\{\mathbf{x}\}, \{\mathbf{y}\})$$

- There is a clever “forward recursion” to compute the sum efficiently.

$$\alpha_j(t) = P(\mathbf{y}_1^t, x_t = j)$$

$$\alpha_j(1) = \pi_j A_j(\mathbf{y}_1)$$

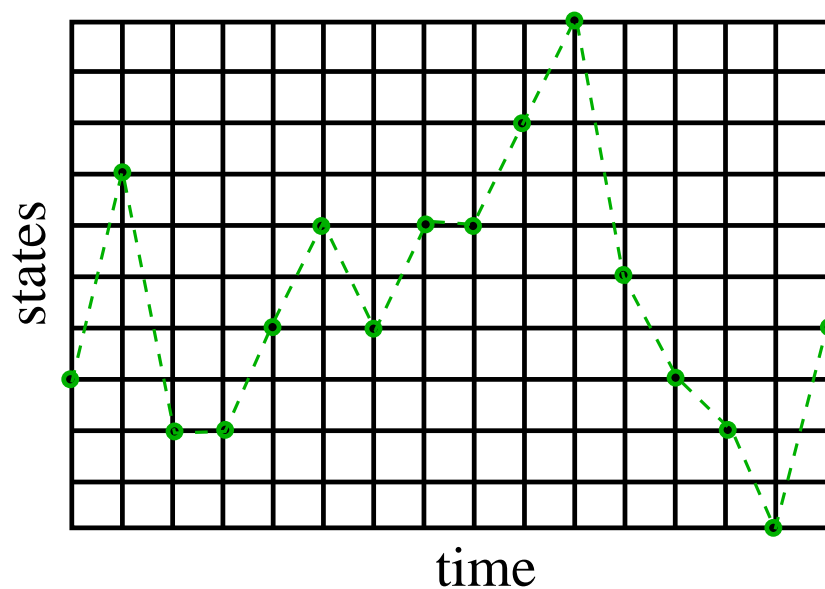
$$\alpha_k(t+1) = \left\{ \sum_j \alpha_j(t) S_{jk} \right\} A_k(\mathbf{y}_{t+1})$$

- Notation: $\mathbf{x}_a^b \equiv \{\mathbf{x}_a, \dots, \mathbf{x}_b\}$; $\mathbf{y}_a^b \equiv \{\mathbf{y}_a, \dots, \mathbf{y}_b\}$
- This enables us to easily (cheaply) compute the desired likelihood L since we know we must end in some possible state:

$$L = \sum_k \alpha_k(T)$$

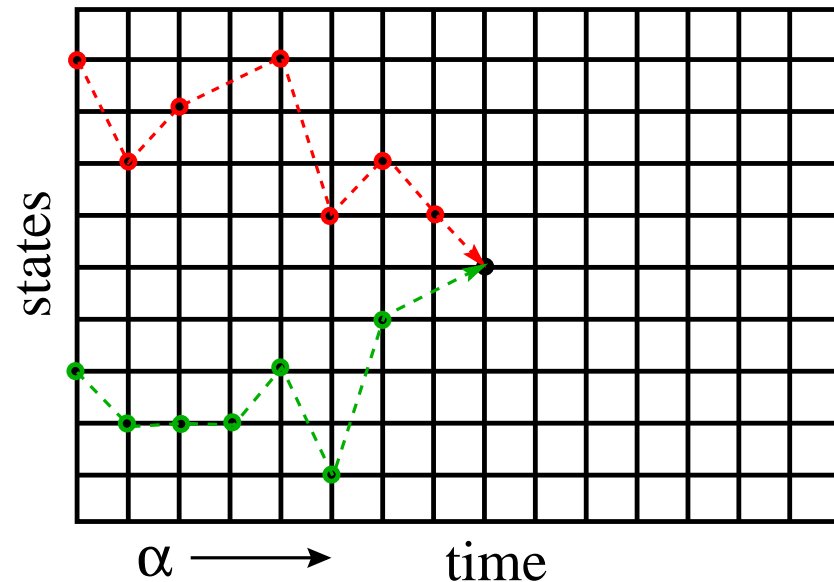
HMM INFERENCE: BUGS ON A GRID

- Naive algorithm:
 1. start bug in each state at $t=1$ holding value 0
 2. move each bug forward in time: make copies & increment the value of each copy by transition prob. + output emission prob.
 3. go to 2 until all bugs have reached time T
 4. sum up values on all bugs



BUGS ON A GRID - TRICK

- Clever recursion:
 - adds a step between 2 and 3 above which says: at each node, replace all the bugs with a single bug carrying the sum of their values



- This trick is called *dynamic programming*, and can be used whenever we have a summation, search, or maximization problem that can be set up as a grid in this way. The axes of the grid don't have to be "time" and "states".

HMMs: INFERENCE OF HIDDEN STATES

- What if we want to estimate the hidden states given observations? To start with, let us estimate a single hidden state:

$$\begin{aligned} p(x_t|\{\mathbf{y}\}) &= \gamma(x_t) = \frac{p(\{\mathbf{y}\}|x_t)p(x_t)}{p(\{\mathbf{y}\})} \\ &= \frac{p(\mathbf{y}_1^t|x_t)p(\mathbf{y}_{t+1}^T|x_t)p(x_t)}{p(\mathbf{y}_1^T)} \\ &= \frac{p(\mathbf{y}_1^t, x_t)p(\mathbf{y}_{t+1}^T|x_t)}{p(\mathbf{y}_1^T)} \\ p(x_t|\{\mathbf{y}\}) &= \gamma(x_t) = \frac{\alpha(x_t)\beta(x_t)}{p(\mathbf{y}_1^T)} \end{aligned}$$

where

$$\begin{aligned} \alpha_j(t) &= p(\mathbf{y}_1^t, x_t = j) \\ \beta_j(t) &= p(\mathbf{y}_{t+1}^T | x_t = j) \\ \gamma_i(t) &= p(x_t = i | \mathbf{y}_1^T) \end{aligned}$$

FORWARD-BACKWARD ALGORITHM

- We compute these quantities efficiently using another recursion. Use total prob. of all paths going through state i at time t to compute the *conditional* prob. of being in state i at time t :

$$\begin{aligned}\gamma_i(t) &= p(x_t = i \mid \mathbf{y}_1^T) \\ &= \alpha_i(t)\beta_i(t)/L\end{aligned}$$

where we defined:

$$\beta_j(t) = p(\mathbf{y}_{t+1}^T \mid x_t = j)$$

- There is also a simple recursion for $\beta_j(t)$:

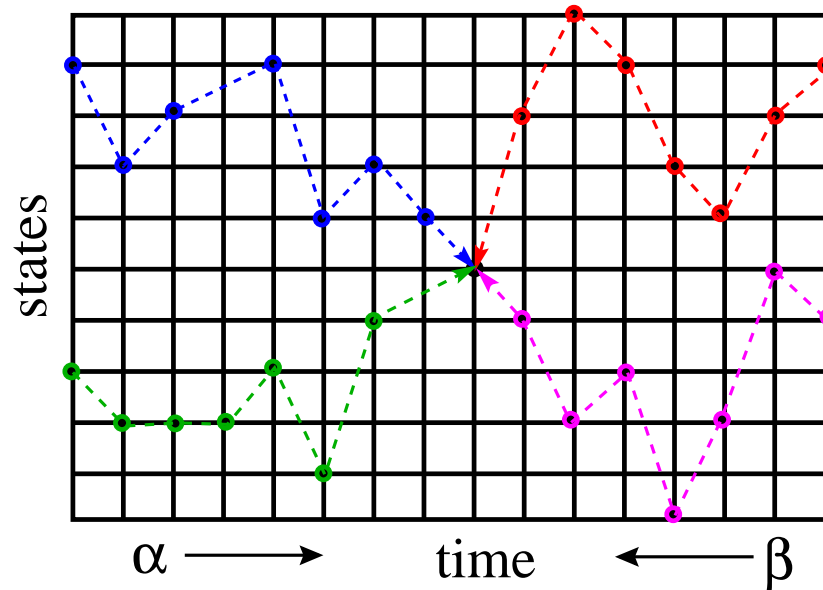
$$\beta_j(t) = \sum_i S_{ji} \beta_i(t+1) A_i(\mathbf{y}_{t+1})$$

$$\beta_j(T) = 1$$

- $\alpha_i(t)$ gives total *inflow* of prob. to node (t, i)
 $\beta_i(t)$ gives total *outflow* of prob.

FORWARD-BACKWARD ALGORITHM

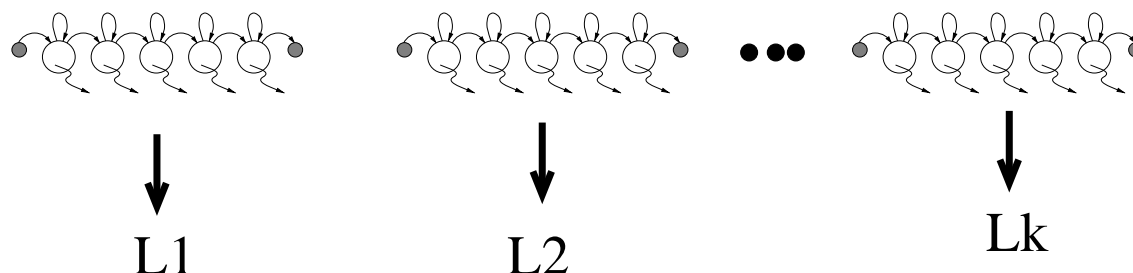
- $\alpha_i(t)$ gives total *inflow* of prob. to node (t, i)
 $\beta_i(t)$ gives total *outflow* of prob.



- Bugs again: we just let the bugs run forward from time 0 to t and backward from time T to t .
- In fact, we can just do one forward pass to compute all the $\alpha_i(t)$ and one backward pass to compute all the $\beta_i(t)$ and then compute any $\gamma_i(t)$ we want. Total cost is $O(K^2T)$.

USING HMMs FOR RECOGNITION

- Use many HMMs for recognition by:
 1. training one HMM for each class (requires *labelled* training data)
 2. evaluating probability of an unknown sequence under each HMM
 3. classifying unknown sequence: HMM with highest likelihood



- This requires the solution of two problems:
 1. Given model, evaluate prob. of a sequence.
(We can do this exactly & efficiently.)
 2. Give some training sequences, estimate model parameters.
(We can find the local maximum of parameter space nearest our starting point using Baum-Welch (EM).)

PARAMETER ESTIMATION USING EM (BAUM-WELCH)

- S_{ij} are transition probs; state j has output distribution $A_j(\mathbf{y})$

$$P(x_{t+1} = j | x_t = i) = S_{ij} \quad P(x_1 = j) = \pi_j$$

$$P(\mathbf{y}_t = y | x_t = j) = A_j(y)$$

- Complete log likelihood:

$$\log p(x, y) = \log \left\{ \pi_{\mathbf{x}_1} \prod_{t=1}^{T-1} S_{x_t, x_{t+1}} \prod_{t=1}^T A_{x_t}(\mathbf{y}_t) \right\}$$

$$= \log \left\{ \prod_i \pi_i^{[\mathbf{x}_1^i]} \prod_{t=1}^{T-1} \prod_{ij} S_{ij}^{[x_t^i, x_{t+1}^j]} \prod_{t=1}^T \prod_k A_k(\mathbf{y}_t)^{[\mathbf{x}_t^k]} \right\}$$

$$= \sum_i [\mathbf{x}_1^i] \log \pi_i + \sum_{t=1}^{T-1} \sum_{ij} [x_t^i, x_{t+1}^j] \log S_{ij} + \sum_{t=1}^T \sum_k [x_t^k] \log A_k(\mathbf{y}_t)$$

where the indicator $[x_t^i] = 1$ if $x_t = i$ and 0 otherwise

- For EM, we need to compute the *expected complete log likelihood*.

STATE EXPECTATIONS REQUIRED FROM THE E-STEP

- The expected complete log likelihood requires

$$\gamma_i(t) = \langle [x_t^i] \rangle \quad \text{and} \quad x_{ij}(t) = \langle [x_t^i, x_{t+1}^j] \rangle$$

- So in the E-step we need to compute both

$$\gamma_i(t) = p(x_t = i | \{\mathbf{y}\}) \quad \text{and} \quad x_{ij}(t) = p(x_t = i, x_{t+1} = j | \{\mathbf{y}\}).$$

- We already know how to compute $\gamma_i(t)$ using α and β recursions.

We can compute $x_{ij}(t)$ the same way (recall BP):

$$\begin{aligned} x_{ij}(t) &= p(x_{it}, x_{jt+1} | \{\mathbf{y}\}) = p(x_{it} | \{\mathbf{y}\}) p(x_{jt+1} | x_{it}, \{\mathbf{y}\}) \\ &= p(x_{it}, y_1^t | y_{t+1}^T) p(x_{jt+1} | x_{it}, y_{t+1}^T) / p(y_1^t | y_{t+1}^T) \\ &= \frac{p(x_{it}, y_1^t) p(y_{t+1}^T | x_{it}, y_1^t) p(y_{t+1}^T | x_{jt+1}, x_{it}) p(x_{jt+1} | x_{it})}{p(y_1^t | y_{t+1}^T) p(y_{t+1}^T)} \\ &= \frac{p(x_{it}, y_1^t) p(y_{t+1}^T | x_{it}) p(y_{t+1} | x_{jt+1}) p(y_{t+2}^T | x_{jt+1}) p(x_{jt+1} | x_{it})}{p(y_1^T) p(y_{t+1}^T | x_i = t)} \\ &= \alpha_i(t) A_j(y_{t+1}) S_{ij} \beta_j(t+1) / L \end{aligned}$$

M-STEP: NEW PARAMETERS ARE JUST RATIOS OF FREQUENCY COUNTS

- Initial state distribution: expected #times in state i at time 1:

$$\hat{\pi}_i = \gamma_i(1)$$

- Expected #transitions from state i to j which begin at time t :

$$x_{ij}(t) = \alpha_i(t) S_{ij} A_j(\mathbf{y}_{t+1}) \beta_j(t+1) / L$$

so the estimated transition probabilities are:

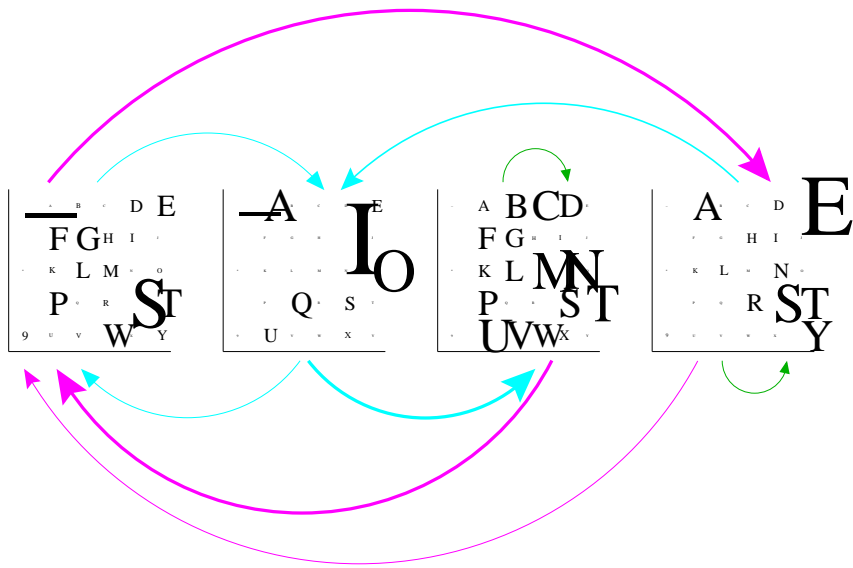
$$\hat{S}_{ij} = \frac{\sum_{t=1}^{T-1} x_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}$$

- The output distributions are the expected number of times we observe a particular symbol in a particular state:

$$\hat{A}_j(y_0) = \frac{\sum_{t|\mathbf{y}_t=y_0} \gamma_j(t)}{\sum_{t=1}^T \gamma_j(t)}$$

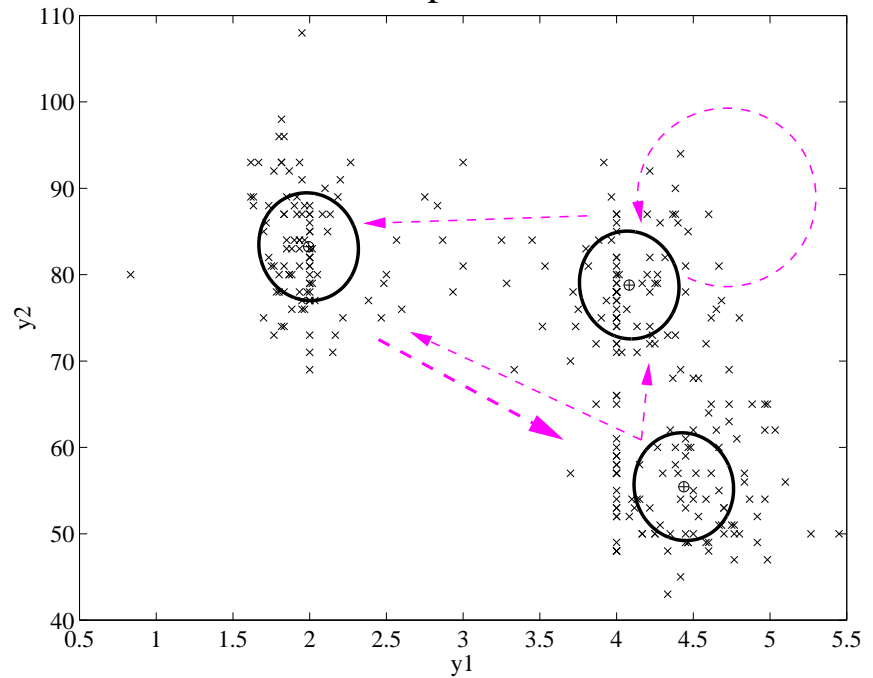
HMM EXAMPLES

English character sequences



Geyser data (continuous)

State output functions



SUMMARY: PROBABILISTIC GRAPHICAL MODELS

- Graphical models provide a *compact factorizations* of large joint probability distributions by exploiting *conditional independencies*.
- Efficient algorithms exist for learning the parameters of a graphical model and for inferring distributions over certain variables in the model given observations of other variables.
- The simplest graphical models have only a single node and represent parametric distributions as in traditional statistics.
- The next most complex models have two nodes and represent classification, regression, clustering and latent factor models.
- Even more complex models have chain and tree structures.
- For fully observed models, maximum likelihood learning decouples across the network and each node can learn its parameters given only observations of itself and its parents.

SUMMARY: EM AND BELIEF PROPAGATION

- In networks with hidden or latent variables, learning is much harder and requires inferring the distribution over the unobserved variables given the observed variables.
- Given the results of such inference, the EM algorithm can be used to update the parameters in a way that never decreases the likelihood of the training data.
- Inference in two-node models is just a simple application of Bayes' rule. This is used at test time in supervised models and at training time in unsupervised models.
- In more complex models such as chains and trees, efficient inference can be performed with the belief propagation (BP) algorithm which is the natural extension of dynamic programming to statistical models.

