Chapter 7

Lecture 7: graphical models and belief propagation

Sept. 26, 2013 MIT EECS course 6.869, Bill Freeman and Antonio Torralba

7.1 Graphical models

Recall the Bayesian inference task: if our observations are the elements of a vector, $\vec{y}$, and we seek to infer the scene parameters, called the vector, $\vec{x}$, then Bayesian inference involves finding

$$P(\vec{x}|\vec{y}) = \frac{P(\vec{y}|\vec{x})P(\vec{x})}{P(\vec{y})} \tag{7.1}$$

The *likelihood term* $P(\vec{y}|\vec{x})$ describes how a rendered scene $\vec{x}$ generates observations $\vec{y}$. The *prior probability*, $P(\vec{x})$, tells the probability of any given scene $\vec{x}$ occuring. We often ignore the denominator, $P(\vec{y})$, called the *evidence*, as it is constant with respect to the variables we seek to estimate, $\vec{x}$.

Such probabilistic formulations are useful for many problems in vision, but often the relationships between the variables and observations are complicated and solving for the best $\vec{x}$ can be very difficult.

Today we study graphical models and belief propagation, which are tools for inferring hidden variables from observations. Probabilistic graphical models describe joint probability distributions in a way that allows us to reason about them and calculate with them even when we're modeling very complicated situations. These things are pervasive in vision, because we often have such complicated situations and we need a framework to reason about the entire system while taking advantage of the modularity of its components. Belief propagation lets us efficiently estimate the states of unobserved variables in the system, for example, given these image observations, what is my estimate of the pose of the person?

Today's topic is the subject of an entire course, 6.438, although, of course, 6.438 covers things in much more depth than we'll treat things in today's lecture. In this lecture, we're just talking about the framework and the algorithm, but we'll use this machinery in various other parts of the course, for tracking, segmentation, and many image interpretation tasks. The goal of this lecture is to expose you to these graphical models, and to teach you the belief propagation algorithm. the problem set after the color one.

A probabilistic graphical model is a graph that describes a class of probability distributions that shares a common structure. The graph has nodes, drawn as circles, indicating the variables of the joint probability. And it has edges, drawn as lines connecting some nodes to others. (For now, we'll restrict

our attention to a type of graphical model called undirected, so the edges are line segments, not arrows). The edges indicate the conditional independence structure of the nodes of the graph. If there is no edge between two nodes, then those variables are independent, conditioning on intervening nodes in any path between them in the graph. If two nodes have a line between them, then we cannot assume they are conditionally independent. Let's go through this with some examples. See the text in the caption of each example figure below.

Figure 7.1: Here's the world's simplest probabilistic graphical model. This "graph" tells us something about the structure of a class of probability functions on x_1. In this case, all it tells us is that they can have any functional form, $P(x_1) = \phi_1(x_1)$.

Figure 7.2: Another very simple graphical model. Here, the lack of a line connecting x_1 with x_2 indicates a lack of statistical dependence between these two variables. Conditioned on knowing the values of all the neighbors of x_1 and x_2–in this case there are none–x_1 and x_2 are statistically independent. So the class of joint probabilities that respects this graphical model looks like this: $P(x_1, x_2) = \phi_1(x_1)\phi_2(x_2)$.

Figure 7.3: Let's add a line between these two variables. By that, we mean that there may be a direct statistical dependency between them. That's all we know about the class of probability distributions depicted here, so all we can write about the joint probability is $P(x_1, x_2) = \phi_{12}(x_1, x_2)$
. This graph offers no simplification from the most general probability function of two variables.

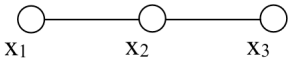

Figure 7.4: Finally, a non-trivial graph with a little structure. This graph structure means that if we condition on the variable x_2, then x_1 and x_3 are independent. So the joint probability of these three variables is restricted to have a special structure. A celebrated theorem, the Hammersly-Clifford theorem, tells us what that structure is: a product of functions of all the "maximal cliques" of the graph. A clique is a group of variables where each is connected to every other variable in the clique. A maximal clique is a clique that can't include more nodes of the graph without losing the clique property. So by drawing this graph, we are refering to a family of joint probabilities which have the form, $P(x_1, x_2, x_3) = \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)$. Later in the lecture, we'll exploit that structure of the joint probability to let us perform inference efficiently.

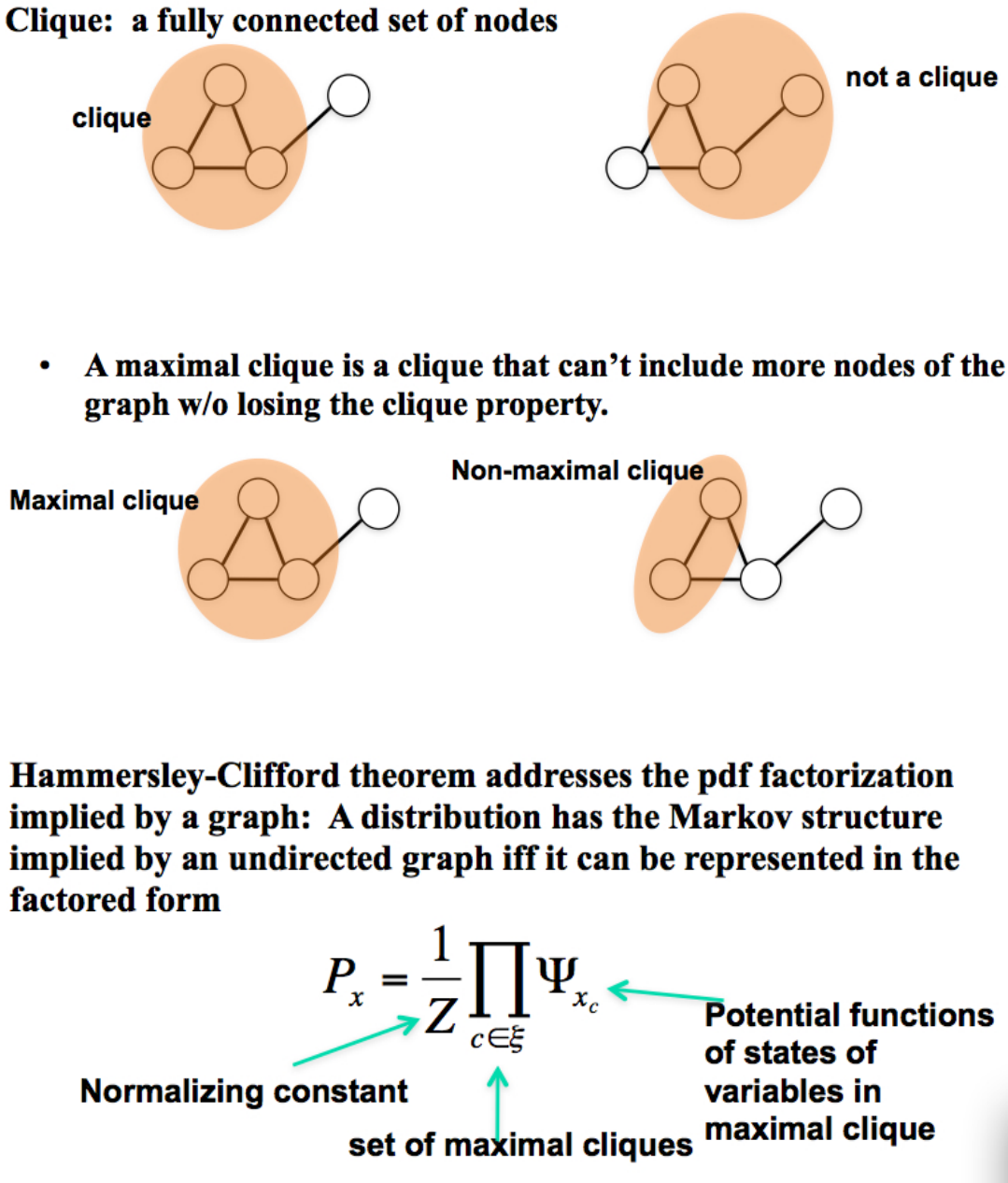

Figure 7.5: The top two rows show examples of cliques and maximal cliques, respectively. The Hammersley-Clifford theorem gives the relationship between an undirected graphical model and the class of joint probability distributions the model represents. Probability distributions within the class share common structure in their inference algorithms.

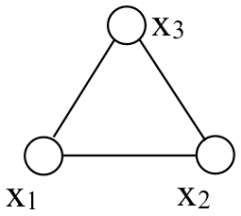

Figure 7.6: Here's a conundrum: what do we mean by this graph? There are no conditional independencies show in this graph, so again we're forced to describe this with the most general joint probability function, $P(x_1, x_2, x_3) = \phi_{123}(x_1, x_2, x_3)$. (The graph has one maximal clique–the entire graph). Suppose you wanted to specify that the structure of the joint probability was $P(x_1, x_2, x_3) = \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\phi_{13}(x_1, x_3)$, how would you do it? The kind of graphs we're working with now, which let us describe conditional independence structure, are called undirected graphical models. There are other types of graphical models that make it easy to display such a factorization.

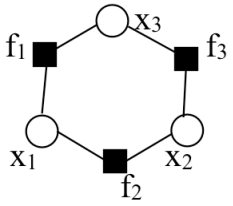

Figure 7.7: A **factor graph** (most often used in coding theory) makes displaying such a factorization of the posterior probability easy. Factor graphcs explicitly draw out the factors in a factorization of the joint probability. The square nodes are function nodes; they connect to variables that are the arguments of those functions. The joint probability described by a factor graph is just the product of all the factor node functions. The inputs to these functions are the variables to which each factor node is connected. So the factor graph shown here depicts the family of joint probabilities over x_1, x_2, and x_3 having the form $P(x_1, x_2, x_3) = f_1(x_1, x_3)f_2(x_1, x_2)f_3(x_2, x_3)$.

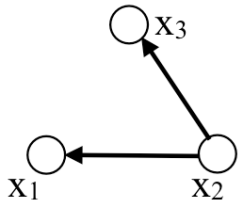

Figure 7.8: While we're describing other graphical model types, there is a 3rd type of graphical model that's commonly used. **Directed graphical models** describe factorizations of the joint probability into products of conditional probability distributions. Each node in a directed graph contributes a well-specified factor in the joint probability: the probability of its variable, conditioned all the variables originating arrows pointing into it. So for this graph, $P(x_1, x_2, x_3) = P(x_2)P(x_1|x_2)P(x_3|x_2)$. Again, that's a more restrictive joint probability for the 3 variables than the most general class of distributions possible.

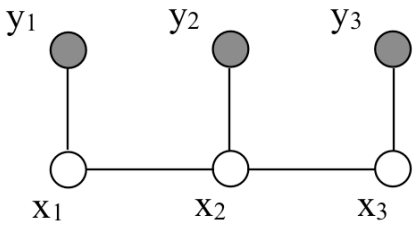

Figure 7.9: Now, we return to undirected graphical models. Typically in a graphical model, we'll have both observed and unobserved variables. Here's a simple "Markov chain" structure that has 3 observed variables, the shaded y variables above, and 3 unobserved variables, the x variables below. It's a chain because the variables form a linear sequence. It's Markov because the hidden variables have the Markov property: conditioned on x_2, variable x_3 is independent of node x_1. The joint probability of all the variables shown here is $P(x_1, x_2, x_3, y_1, y_2, y_3) = \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\psi_1(y_1, x_1)\psi_2(y_2, x_2)\psi_3(y_3, x_3)$. Using $P(a, b) = P(a|b)P(b)$, we can also write this as $P(x_1, x_2, x_3|y_1, y_2, y_3) = \frac{1}{P(\vec{y})}\phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\psi_1(y_1, x_1)\psi_2(y_2, x_2)\psi_3(y_3, x_3)$.

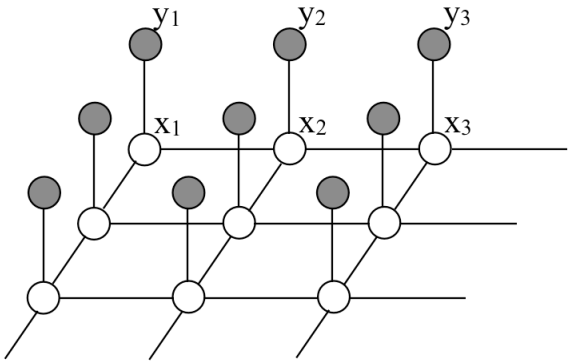

Figure 7.10: For vision applications, we sometimes have 1-d structures, such as the Markov chain shown above, often describing structure over time. To capture relationships over space we often use the 2-d structure of a Markov random field, such as is shown here. Then the joint probability over all the variables factorizes into a form like this, $P(\vec{x}|\vec{y}) = \frac{1}{P(\vec{y})}\prod_{(i,j)}\phi_{ij}(x_i, x_j)\prod_i \psi_i(x_i, y_i)$, where the first product is over spatial neighbors i and j.

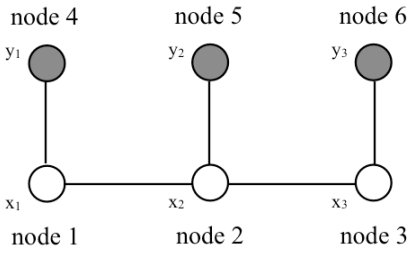

Figure 7.11: Node numbering for this simple belief propagation example.

7.2 Inference in graphical models

Now let's see how these graphical models can be useful for inferring what's in the world, based on what you see. Typically, we make many observations of the variables of some system, and we want to find the the state of some hidden variable, given those observations. The posterior probability, $P(\vec{x}|\vec{y})$, tells us the probability for *any* set of hidden variables, $\vec{x}$, but we typically want some single *best* estimate, also called a point estimate. We may want the value of the hidden variable that maximizes the posterior distribution (MAP estimate), or the mean value (MMSE estimate), or some other point estimate. For the purpose of this derivation, let's assume we want to find posterior mean value of our hidden variables, and therefore we'll compute the marginal probability at each node (from which it is simple to derive the mean). In other words, given $P(\vec{x}|\vec{y})$, compute $P(x_i|\vec{y})$, where i is the hidden node where you want to find the marginal, then from that it is simple to compute the mean.

7.2.1 Simple example

Let's work through this for a simple example, which will give us an intuition for the general cases. Consider the three-node Markov chain of Fig. 7.4. Let's marginalize the joint probability to find the marginal probability at node 1, $P(x_1|\vec{y})$. We have

$$P(x_1|\vec{y}) = \sum_{x_2} \sum_{x_3} P(x_1, x_2, x_3|\vec{y}) \tag{7.2}$$

Here's the important point: If we knew nothing about the structure of the joint probability $P(x_1, x_2, x_3|\vec{y})$, that would require $|x|^3$ computations: a double-sum over all x_2 states must be computed for each possible x_3 state. (We're denoting the number of states of any of the x variables as $|x|$). In the more general case, for a Markov chain of N nodes, we would need $|x|^N$ summations to compute the desired marginal at any node of the chain. Such a computation very quickly becomes intractable.

But watch what happens when we exploit the known structure of the joint probability of the variables in the chain. Substituting the known structure of the joint probability into the marginalization equation, Eq. (7.3), gives

$$P(x_1|\vec{y}) = \frac{1}{P(\vec{y})} \sum_{x_2} \sum_{x_3} \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\psi_1(y_1, x_1)\psi_2(y_2, x_2)\psi_3(y_3, x_3) \tag{7.3}$$

This next step shows the main idea of belief propagation. Because of the modular structure of the joint probability, not every variable is coupled to every other one and we can pass the summations through variables it doesn't sum over. This gives the same marginalization result, but computed much more

efficiently (it's a small difference for this short chain, but will make a huge difference for longer ones). To make subsequent notation for the messages, m, simpler, let's number the nodes as follows: x_1, x_2 and x_3 are nodes 1, 2, 3, respectively. y_1, y_2 and y_3 are nodes 4, 5, and 6. We can write

$$
\begin{aligned}
P(x_1|\vec{y}) &= \frac{1}{P(\vec{y})} \sum_{x_2} \sum_{x_3} \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\psi_1(y_1, x_1)\psi_2(y_2, x_2)\psi_3(y_3, x_3) && (7.4)\\
&= \frac{1}{P(\vec{y})} \psi_1(y_1, x_1) \sum_{x_2} \phi_{12}(x_1, x_2)\psi_2(y_2, x_2) \sum_{x_3} \phi_{23}(x_2, x_3)\psi_3(y_3, x_3) && (7.5)\\
&= \frac{1}{P(\vec{y})} m_{41}(x_1) \sum_{x_2} \phi_{12}(x_1, x_2) m_{52}(x_2) \sum_{x_3} \phi_{23}(x_2, x_3) m_{63}(x_3) && (7.6)\\
&= \frac{1}{P(\vec{y})} m_{41}(x_1) \sum_{x_2} \phi_{12}(x_1, x_2) m_{52}(x_2) m_{32}(x_2) && (7.7)\\
&= \frac{1}{P(\vec{y})} m_{41}(x_1) m_{21}(x_1) && (7.8)
\end{aligned}
$$

where we have defined the following terms as messages: $m_{41}(x_1) = \psi_1(y_1, x_1)$, $m_{52}(x_2) = \psi_2(y_2, x_2)$, $m_{63}(x_3) = \psi_3(y_3, x_3)$, $m_{32}(x_2) = \sum_{x_3} \phi_{23}(x_2, x_3) m_{63}(x_3)$, and $m_{21}(x_1) = \sum_{x_2} \phi_{12}(x_1, x_2) m_{52}(x_2) m_{32}(x_2)$. Factorizing the double-sum as we did in Eq. (7.5) reduces the number of terms summed from order $|x|^3$ to order $3|x|^2$, and in the case of a length N chain, from order $|x|^N$ to order $N|x|^2$, a huge computational savings for large N! Clearly, this type of factorization is an important thing to do.

If you wanted to find the marginal probability at a second node, you can write out the sums over variables needed for that node, pass the sums through factors in the joint probability that they don't operate on, and come up with an efficient series of partial sums analogous to Eq. (7.5) in the marginalization to find $P_1(x_1)$. If you go through that exercise, you'll find that many of the partial sums we just did would need to be recomputed (this is more relevant in a more realistic graphical model with more nodes). For example, to find the marginal probability at node 2, we would compute

$$
\begin{aligned}
P(x_2|\vec{y}) &= \frac{1}{P(\vec{y})} \sum_{x_1} \sum_{x_3} \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\psi_1(y_1, x_1)\psi_2(y_2, x_2)\psi_3(y_3, x_3) && (7.9)\\
&= \frac{1}{P(\vec{y})} \psi_2(y_2, x_2) \sum_{x_1} \psi_1(y_1, x_1)\phi_{12}(x_1, x_2) \sum_{x_3} \phi_{23}(x_2, x_3)\psi_3(y_3, x_3) && (7.10)
\end{aligned}
$$

This again requires the partial sum, $\sum_{x_3} \phi_{23}(x_2, x_3)\psi_3(y_3, x_3)$ that we saw in Eq. (7.5) for $P(x_1|\vec{y})$. It would be nice to have an automatic procedure for identifying those computations that will be needed for other marginalizations to let us cache them efficiently. Belief propagation does that for us by identifying those reusable partial sums as "messages".

Definition In belief propagation for finding the marginal probability at every node, a *message* is a re-usable partial sum for the marginalization calculations.

The sums that can be re-used involve marginalization over the states of one node, leaving unsummed the states of some second node. We call that partial sum a message from the first node to the second. The subsequent lines, Eqs. (7.6) through (7.8), is just a re-naming of the partial sums to define messages and their arguments. (This identification of partial sums as messages is just for bookkeeping, and if you're uncomfortable with these "messages", you can always substitute the phrase "re-usable partial sum"). We re-write every partial sum as a message *from* the node of the variable being summed over

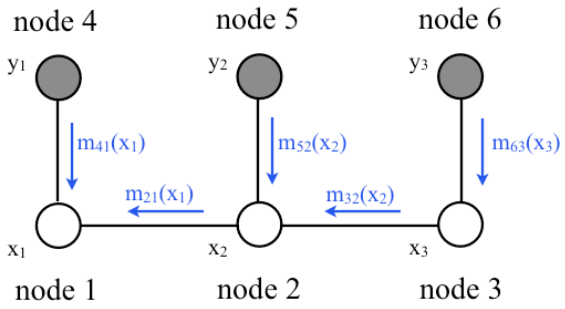

Figure 7.12: Summary of the messages (partial sums) for this simple belief propagation example.

to the node of the variable that's left, not being summed over. For example, in Eq. (7.6), we identify $\psi_3(y_3, x_3) = m_{63}(x_3)$ as a message from node 6 (y_3) to node 3 (x_3), a function of the states of the variable x_3. Similarly, we write

$$m_{32}(x_2) = \sum_{x_3} \psi_{23}(x_2, x_3) m_{63}(x_3) \tag{7.11}$$

and

$$m_{21}(x_1) = \sum_{x_2} \psi_{12}(x_1, x_2) m_{52}(x_2) m_{32}(x_2) \tag{7.12}$$

Then, finishing the renaming of the partial sums as messages, we have, for the desired marginal probability

$$P(x_1|\vec{y}) = \frac{1}{P(\vec{y})} m_{41}(x_1) m_{21}(x_1) \tag{7.13}$$

7.3 Belief propagation (BP)

Now let's present the general case. We won't derive the general case; you can find that in several good sources describing belief propagation []. But we hope the simple example above makes these general rules make sense and seem intuitive. It is straightforward to verify, for any specific graphical model without loops, that these rules lead to the equivalent partial sum calculations needed in computing the marginal probability at any node.

The first step (which may or may not be needed for your graphical model) is to convert the graphical model into one with pairwise potentials. This can be done by augmenting the state of some nodes to encompass several nodes, until the remaining nodes only need pairwise potential functions in their factorization of the joint probability.

7.3.1 Message-passing rule

The arguments of messages are always the state of the node that the message is going to.

To compute the message from node j to node i:

1. Multiply together all messages coming in to node j, except for the message from node i back to node j (we ignore that one in this calculation).

2. Multiply by the compatibility function $\psi_{ij}(x_i, x_j)$.

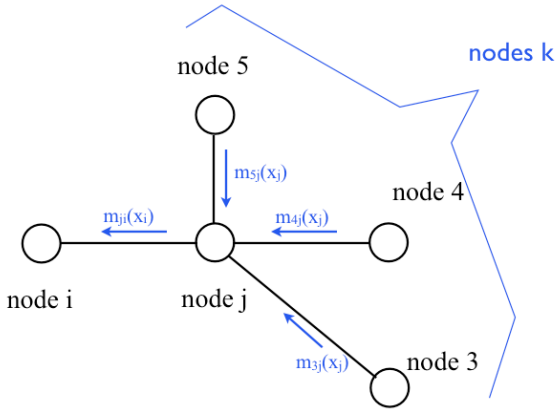

Figure 7.13: To pass a message from node j to node i. (To avoid index overload, we give numerical labels to the depicted neighbor nodes k of node j.)

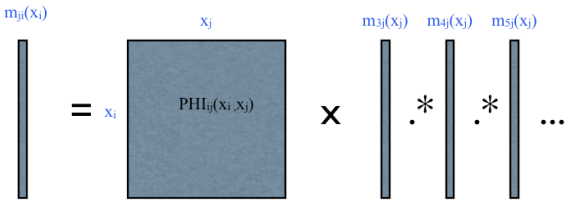

Figure 7.14: Depiction of belief propagation message passing rules, showing vector and matrix shapes.

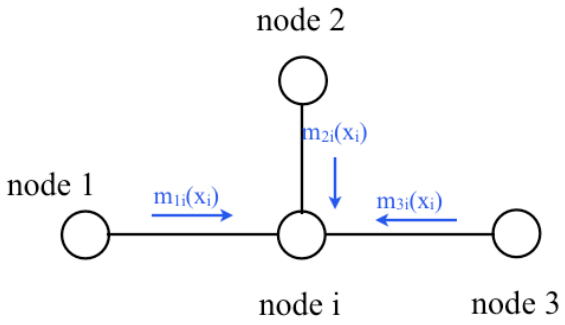

Figure 7.15: To compute the marginal probability at node i, we multiply together all the incoming messages at that node: $P_i(x_i) = \prod_{j \in \eta(i)} m_{ji}(x_i)$. (To avoid index overload, we give numerical labels to the depicted neighbor nodes j of node i.)

3. Marginalize over the variable x_j.

These steps are summarized in this equation to compute the message from node j to node i:

$$m_{ji}(x_i) = \sum_{x_j} \psi_{ij}(x_i, x_j) \prod_{k \in \eta(j) \backslash i} m_{kj}(x_j) \tag{7.14}$$

where $\eta(j) \backslash i$ means "the neighbors of node j except for node i".

For the case of continuous variables, which we won't discuss here, the sum over the states of x_j in Eq. (7.14) is replaced by an integral over the domain of x_j.

7.3.2 Marginal probability

The marginal probability at a node is the product of all incoming messages at that node:

$$P_i(x_i) = \prod_{j \in \eta(i)} m_{ji}(x_i) \tag{7.15}$$

7.3.3 Comments

Some comments

- We're rearranging sums in the marginalization computation to exploit structure in the joint probability distribution. If there is no structure in the joint pdf, you need to be concerned with every state of every other node when you marginalize out any node. If there is structure, you can assume many factors are constant in the summation over the states of a node.
- Belief propagation follows the "nosey neighbor rule" (from Brendan Frey, U. Toronto). Every node is a house in some neighborhood. The nosey neighbor says (as he/she passes the message to you): "given everything I've heard, here's what I think is going on inside your house." That metaphor made much more sense to me after I started having teenaged children.
- The BP algorithm implies a computational structure. There are little computers attached to each node, and they perform these products and sums locally. Of course, you can also run the algorithm with a centralized processor, as well.

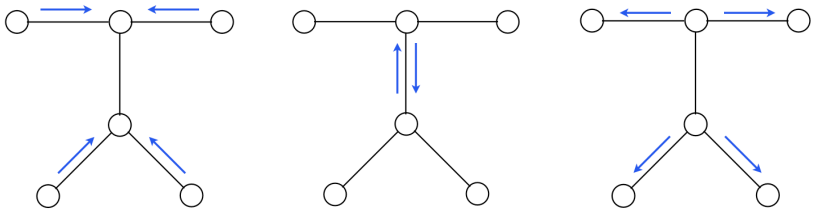

Figure 7.16: Example of **synchronous parallel update schedule** for BP message passing. Whenever any node has the required incoming messages needed to send an outgoing message, it does. (a) At the first iteration, only the leaf nodes have the needed incoming messages to send an outgoing message (by definition, leaf nodes have no links other than the one on which they'll be sending their outgoing message, so they have no incoming messages to wait for). (b) second iteration, (c) third iteration. By the third iteration, every edge has messages computed in both directions, and we can now compute the marginal probability at every node in the graph.

- When do we invoke the BP update rule, Eq. (7.14)? Whenever all the incoming messages to a node are valid. If there are no incoming messages to a node, then its outgoing message is always valid. This lets us start the algorithm.

7.3.4 Message update sequence

A node can send a message whenever all the incoming messages it needs have been computed. So we can start with the outgoing messages from leaf nodes, since they have no incoming messages other than from the node to which they are sending a message, which doesn't enter in the outgoing message computation. Two natural message passing protocols are consistent with that rule: depth-first update, and parallel update. In depth-first update, one node is arbitrarily picked as the root. Messages are then passed from the leaves of the tree (leaves, relative to that root node) up to the root, then back down to the leaves. In parallel update, at each turn, every node sends every outgoing message for which it has received all the necessary incoming messages. Figure 7.16 depicts the flow of messages for the parallel, synchronous update scheme.

We are efficiently re-using messages with the BP algorithm. It takes only twice the number of computations to calculate the marginal probability at every node as it does to calculate the marginal probability at a single node. A single message-passing sweep through all the nodes lets us calculate the marginal at any node (using the depth-first update rules to calculate the marginal at the root node). A second sweep from the root node back to all the leaf nodes calculates all the messages needed to find the marginal probability at every node.

7.3.5 Numerical example

Finally, let's work though a numerical example. To make the arithmetic easy, let's solve for the marginal probabilities in the graphical model of two-state (0 and 1) random variables shown in Fig. 7.17. That graphical model has 3 hidden variables, and one variable observed to be in state 0. The compatibility matrices are given in the arrays below (for which the state indices are 0, then 1, reading from left to right and top to bottom).

$$\psi_{12}(x_1, x_2) = \begin{pmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{pmatrix} \tag{7.16}$$

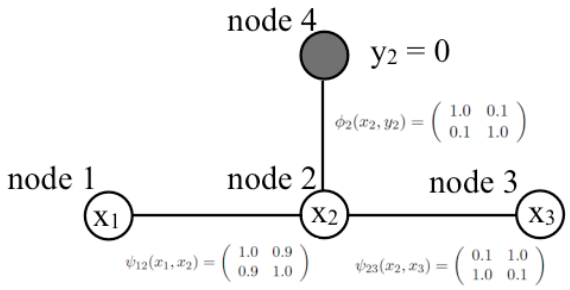

Figure 7.17: Details of numerical example.

$$\psi_{23}(x_2, x_3) = \begin{pmatrix} 0.1 & 1.0 \\ 1.0 & 0.1 \end{pmatrix} \tag{7.17}$$

$$\phi_2(x_2, y_2) = \begin{pmatrix} 1.0 & 0.1 \\ 0.1 & 1.0 \end{pmatrix} \tag{7.18}$$

Note we haven't taken care to normalize the joint probability, so we'll need to normalize each marginal probability at the end. (remember $P(x_1, x_2, x_3, y_2) = \phi_2(x_2, y_2)\psi_{23}(x_2, x_3)\psi_{12}(x_1, x_2)$, which should sum to 1 after summing over all states.)

We can tell what results to expect from looking at the problem (then we can verify that BP is doing the right thing). Node x_2 wants very much to look like $y_2 = 0$, while x_1 has a mild preference to look like x_2. So we expect the marginal probability at node x_2 will be heavily biased toward $x_2 = 0$, and that node x_1 will have a mild preference for state 0. x_3 strongly wants to be the opposite of x_2, so it will be biased toward the state $x_3 = 1$.

Let's see what belief propagation gives us. We'll follow the parallel, synchronous update scheme for calculating all the messages. The leaf nodes can send messages in along their edges without waiting for any messages to be updated. For the message from node 1, we have

$$\begin{aligned} m_{12}(x_2) &= \sum_{x_1} \psi_{12}(x_1, x_2) && (7.19) \\ &= \begin{pmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (7.20) \\ &= \begin{pmatrix} 1.9 \\ 1.9 \end{pmatrix} && (7.21) \\ &= k \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (7.22) \end{aligned}$$

For numerical stability, we typically normalize messages so their entries sum to 1, or so their maximum entry is 1, then remember to renormalize the final marginal probabilities to sum to 1. Here, we've normalized the messages for simplicity, (absorbing the normalization into a constant, k) and because our joint probability is not properly normalized, anyway.

The message from node 3 to node 2 is

$$
\begin{aligned}
m_{32}(x_2) &= \sum_{x_3} \psi_{32}(x_2, x_3) && (7.23)\\
&= \begin{pmatrix} 0.1 & 1.0 \\ 1.0 & 0.1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (7.24)\\
&= \begin{pmatrix} 1.1 \\ 1.1 \end{pmatrix} && (7.25)\\
&= k \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (7.26)
\end{aligned}
$$

We have a non-trivial message from observed node y_2 (node 4) to the hidden variable x_2:

$$
\begin{aligned}
m_{42}(x_2) &= \sum_{x_4} \phi_2(x_2, y_2) && (7.27)\\
&= \begin{pmatrix} 1.0 & 0.1 \\ 0.1 & 1.0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} && (7.28)\\
&= \begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} && (7.29)
\end{aligned}
$$

where y_2 has been fixed to $y_2 = 0$, thus restricting $\phi_2(x_2, y_2)$ to just the first column.

Now we just have two messages left to compute before we have all messages computed (and therefore all node marginals computed from simple combinations of those messages). The message from node 2 to node 1 uses the messages from nodes 4 to 2 and 3 to 2:

$$
\begin{aligned}
m_{21}(x_1) &= \sum_{x_2} \psi_{12}(x_1, x_2) m_{42}(x_2) m_{32}(x_2) && (7.30)\\
&= \begin{pmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{pmatrix} \left[\begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} .* \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right] = \begin{pmatrix} 1.09 \\ 1.0 \end{pmatrix} && (7.31)
\end{aligned}
$$

The final message is that from node 2 to node 3 (since y_2 is observed, we don't need to compute the message from node 2 to node 4). That message is:

$$
\begin{aligned}
m_{23}(x_3) &= \sum_{x_2} \psi_{23}(x_2, x_3) m_{42}(x_2) m_{12}(x_2) && (7.32)\\
&= \begin{pmatrix} 0.1 & 1.0 \\ 1.0 & 0.1 \end{pmatrix} \left[\begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} .* \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right] = \begin{pmatrix} 0.2 \\ 1.01 \end{pmatrix} && (7.33)
\end{aligned}
$$

Now that we've computed all the messages, let's look at the marginals of the three hidden nodes. The product of all the messages arriving at node 1 is just the one message, $m_{21}(x_1)$, so we have (introducing constant k to normalize the product of messages to be a probability distribution)

$$
P_1(x_1) = k m_{21}(x_1) = \frac{1}{2.09} \begin{pmatrix} 1.09 \\ 1.0 \end{pmatrix} \tag{7.34}
$$

As we knew it should, node 1 shows a slight preference for state 0.

The marginal at node 2 is proportional to the product of 3 messages. Two of those are trivial messages, but we'll show them all for completeness:

$$\begin{aligned} P_2(x_2) &= k m_{12}(x_2) m_{42}(x_2) m_{32}(x_2) && (7.35) \\ &= k \begin{pmatrix} 1 \\ 1 \end{pmatrix} .* \begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} .* \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (7.36) \\ &= \frac{1}{1.1} \begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} && (7.37) \end{aligned}$$

As expected, a strong bias for being in state 0.

Finally, for the marginal probability at node 3, we have

$$P_3(x_3) = k m_{23}(x_3) = \frac{1}{1.21} \begin{pmatrix} 0.2 \\ 1.01 \end{pmatrix} \quad (7.38)$$

As predicted, this variable is biased toward being in state 1.

By running belief propagation within this tree, we have computed the exact marginal probabilities at each node, reusing the intermediate sums across different marginalizations, and exploiting the structure of the joint probability to perform the computation efficiently. If nothing were known about the joint probability structure, the marginalization cost would grow exponentially with the number of nodes in the network. But if the graph structure corresponding to the joint probability is known to be a chain or a tree, then the marginalization cost only grows linearly with the number of nodes, and is quadratic in the node state dimensions.

7.4 MAP

Instead of summing over the states of other nodes, we are sometimes interested in finding the $\vec{x}$ that maximizes the joint probability. The argmax operator passes through constant variables just as the summation sign did. This leads to an alternate version of the belief propagation algorithm with the summation (of multiplying the vector message products by the node compatibility matrix) replaced with "argmax". This is called the "max-product" version of belief propagation, and it computes an MAP estimate of the hidden states.

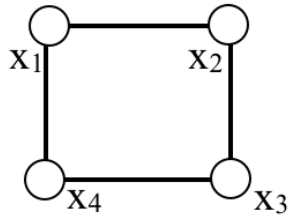

Figure 7.18: A loopy graph

7.5 Loopy belief propagation

The BP message update rules only work to give the exact marginals when the topology of the network is that of a tree or a chain. You can see that, even for a simple 4-node network with one loop (Fig. 7.18), we can't perform the same trick of passing summations over past other variables to get a computational cost that is linear in the number of nodes. If nodes 1 through 4 have pairwise connections, with node 4 connecting back to node 1, the marginal probability at node 1 is given below, after which we have passed through what sums we could and defined some partial sums as messages.

$$P_1(x_1) = \sum_{x_2}\sum_{x_3}\sum_{x_4} \phi_{12}(x_1,x_2)\phi_{23}(x_2,x_3)\phi_{34}(x_3,x_4)\phi_{41}(x_4,x_1) \tag{7.39}$$

$$P_1(x_1) = \sum_{x_2} \phi_{12}(x_1,x_2) \sum_{x_3} \phi_{23}(x_2,x_3) \sum_{x_4} \phi_{34}(x_3,x_4)\phi_{41}(x_4,x_1) \tag{7.40}$$

$$P_1(x_1) = \sum_{x_2} \phi_{12}(x_1,x_2) \sum_{x_3} \phi_{23}(x_2,x_3) m_{4\rightarrow 1,3}(x_3,x_1) \tag{7.41}$$

$$P_1(x_1) = m_{21}(x_1) \tag{7.42}$$

Note that this computation requires partial sums that will be cubic in $|x|$, the cardinality of the state dimension. To see this, consider the partial sum over the states of node x_4, $\sum_{x_4} \phi_{34}(x_3,x_4)\phi_{41}(x_4,x_1)$. Each sum over the states of x_4 must be repeated $|x|^2$ times, once for each possible state configuration of x_3 and x_1. In general, one can show that exact computation of marginal probabilities for graphs with loops depends on a graph-theoretic quantity known as the treewidth of the graph. For many graphical models of interest in vision, such as 2-d Markov Random Fields related to images, these quantities can be intractably large.

But the message update rules are described locally, and one might imagine that it is a useful local operation to perform, even without the global guarantees of ordinary BP. It turns out that is true. Here is the algorithm: **loopy belief propagation algorithm**

1. convert graph to pairwise potentials
2. initialize all messages to all ones, or to random values between 0 and 1.
3. run BP update rules until convergence

One can show that fixed points of the belief propagation algorithm (message configurations where the messages don't change with a message update) correspond to minima of a well-known approximation from the statistical physics community known as the Bethe free energy. In practice, the solutions found by the loopy belief propagation algorithm are often quite good.

Dampening

Since we have guarantees for BP fixed points, we are free to modify the BP update rules provided they give us the same fixed points. One such modification is that of "dampening".

The damped message at belief propagation message update iteration k, $\bar{m}_{ji}^{k}(x_i)$, is a weighted combination of the message that would be sent from the BP algorithm at iteration k, and the damped message that was sent at iteration $k-1$:

$$\bar{m}_{ji}^{k}(x_i) = \alpha m_{ji}^{k}(x_i) + (1-\alpha) m_{ji}^{k-1}(x_i) \tag{7.43}$$

7.5.1 Other algorithms for approximate inference in loopy graphical models

Approximate inference (marginalization) algorithms for Markov networks with loops is a thriving research field. A few of the other methods that are used, in addition to Loopy Belief Propagation:

- graph cuts
- tree-reweighted belief propagation
- iterated conditional modes
- Gibbs sampling

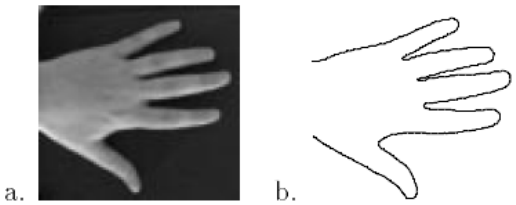

Figure 2: **a.** the first frame of a sequence. The hand is translated to the left. **b.** contour extracted using standard methods

Figure 7.19: Image data for hand contour.

ground. A local cue for DOF is convexity - given three neighboring points along the contour we prefer the DOF that makes the angle defined by those points acute rather than obtuse. Figure 4a shows the results of using this local cue on the hand contour. The local cue is not sufficient.

We can overcome the local ambiguity by minimizing a cost functional that takes into account the DOF at neighboring points in addition to the local convexity. Denote by $L_k(m)$ the DOF at point k along the contour and define

$$J(L) = \sum_m \sum_k V_k(m) L_k(m) - \lambda \sum_m \sum_k L_k(m) L_{k+1}(m) \qquad (14)$$

with $V_k(m)$ determined by the acuteness of the angle at location k.

Figure 4b shows the performance of four propagation algorithms on this task: three traditional relaxation labeling algorithms (MF Hopfield, Rosenfeld et al, constrained gradient descent) and BBP. All three traditional algorithms converge to a local minimum, while the BBP converges to the global minimum. Figure 4c shows the local minimum reached by the Hopfield network and figure 4d shows the correct solution reached by the BBP algorithm. Recall (section 1.1) that BBP is guaranteed to converge to the correct posterior given all the data.

Figure 7.20: Function to be optimized by belief propagation.

7.6 Vision applications of 1-d inference

The figures from this section are from a nice NIPS paper by Yair Weiss,

```
Interpreting images by propagating Bayesian beliefs, by Yair Weiss
in: M.C. Mozer, M.I. Jordan and T. Petsche, editors,
Advances in Neural Information Processing Systems 9 908-915 (1997).
http://www.cs.huji.ac.il/~yweiss/nips96.pdf
```

They show how to use belief propagation to integrate the local evidence for figure/ground over a 1-d Markov chain. The local evidence is very weak, but the integrated figure/ground signal is quite robust.

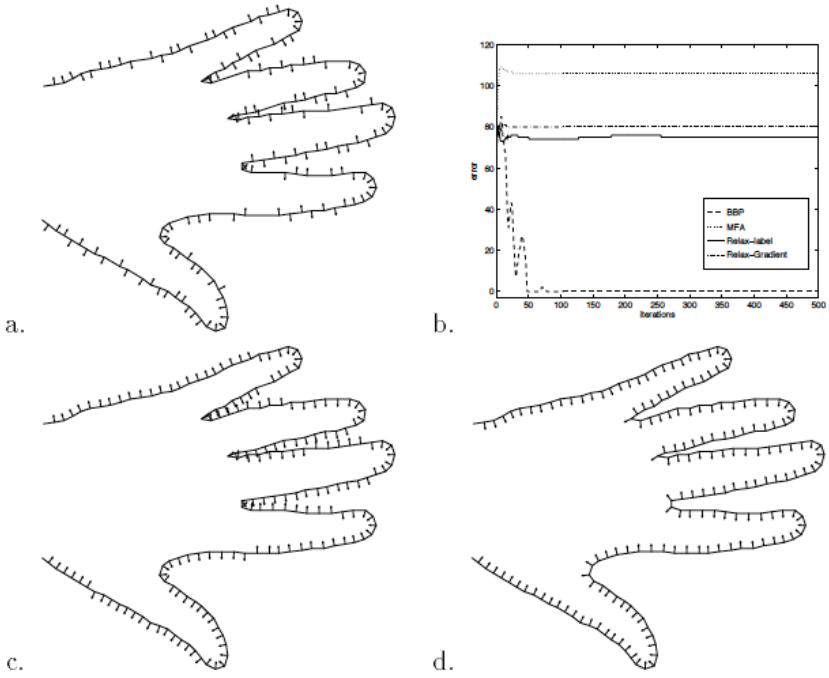

Figure 4: **a.** Local estimate of DOF along the contour. **b.** Performance of Hopfield,gradient descent, relaxation labeling and BBP as a function of time. BBP is the only method that converges to the global minimum. **c.** DOF estimate of Hopfield net after convergence. **d.** DOF estimate of BBP after convergence.

xs

Figure 7.21: BP results for figure/ground computation (and comparison with Hopfield network in (c).)