Contents

1 Probabilistic graphical models

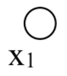
In this chapter we study probabilistic models and efficient computations with those models. *Probabilistic graphical models* describe joint probability distributions in a modular way that allows us to reason about the visual world even when we're modeling very complicated situations. These models are pervasive in vision, where we often need to exploit modularity to make computations tractable. We will describe belief propagation, which uses the structure of the graphical models to efficiently estimate the states of unobserved variables in a system. For example, given image observations, we want to efficiently estimate the pose of the person, and belief propagation can let us do that efficiently.

A probabilistic graphical model is a graph that describes a class of probability distributions sharing a common structure. The graph has nodes, drawn as circles, indicating the variables of the joint probability. It has edges, drawn as lines connecting nodes to other nodes. (At first, we'll restrict our attention to a type of graphical model called undirected, so the edges are line segments without arrowheads).

The edges indicate the conditional independence structure of the nodes of the graph. If there is no edge between two nodes, then the variables described by those nodes are independent, if you condition on the values of intervening nodes in any path between them in the graph. If two nodes do have a line between them, then we cannot assume they are independent. We introduce this visual vocabulary through a set of examples.

1.1 Simple Examples

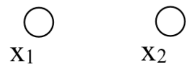
Here is the simplest probabilistic graphical model:



(1.1)

This "graph" is just a single node for the variable x_1. There is no restriction imposed by the graph on the functional form of its probability function. In keeping with notation we'll use later, we write that the probability distribution over x_1, $P(x_1) = \phi_1(x_1)$.

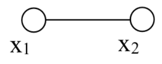
Another very simple graphical model is this:



(1.2)

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 any connecting neighbors of x_1 and x_2–in this case there are none–x_1 and x_2 are statistically independent. Because of that independence, the class of joint probabilities depicted by this graphical model must have this form: $P(x_1, x_2) = \phi_1(x_1)\phi_2(x_2)$.

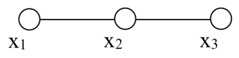
Let's add a line between these two variables:



(1.3)

By that graph, we mean that there may be a statistical dependency between the two variables. The class of probability distributions depicted here is now more general than the one above, and we can only write that the joint probability as some unknown function of x_1 and x_2, $P(x_1, x_2) = \phi_{12}(x_1, x_2)$. This graph offers no simplification from the most general probability function of two variables.

Here is a graph with some structure:

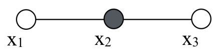


(1.4)

This graph means that if we condition on the variable x_2, then x_1 and x_3 are independent. A general form for $P(x_1, x_2, x_3)$ that guarantees such conditional independence is

$$P(x_1, x_2, x_3) = \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3). \tag{1.5}$$

Note the conditional independence implicit in Eq. (1.5): if the value of x_2 is given (indicated by the solid circle in the graph below), then the joint probability of Eq. (1.5) becomes a product $f(x_1)$ times $g(x_3)$, revealing the independence of x_1 and x_3.

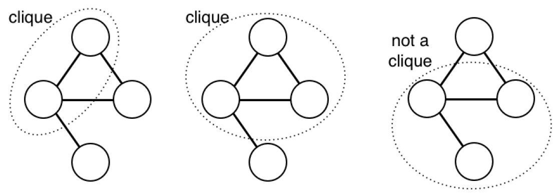


(1.6)

Later in this chapter, we'll exploit that structure of the joint probability to perform inference efficiently using belief propagation.

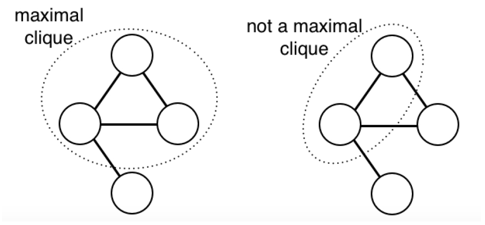
A celebrated theorem, the *Hammersley-Clifford theorem*, tells the form the joint probability must have for any given graphical model. The joint probability consistent with a probabilistic graphical model must be a product of functions of each the "maximal cliques" of the graph. Here we define the terms

A *clique* is any set of nodes where each node is connected to every other node in the clique. These graphs illustrate the clique property:



(1.7)

A maximal clique is a clique that can't include more nodes of the graph without losing the clique property. The sets of nodes below form maximal cliques (left), or do not (right):



(1.8)

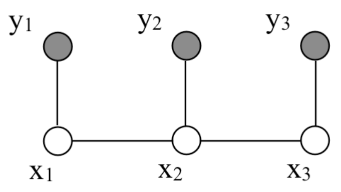
The *Hammersley-Clifford theorem*: A positive probability distribution has the independence structure described by a graphical model if and only if it can be written as a product of functions over the variables of each maximal clique:

$$P(x_1, x_2, \dots, x_N) = \prod_{x_c \in x_i} \Psi_{x_c}, \tag{1.9}$$

where the product is over all maximal cliques x_c in the graph, x_i.

Because the maximal cliques of the graph 1.4 are (x1, x2) and (x2, x3), the Hammersley-Clifford theorem says that any joint probability respecting the conditional independence structure of graph 1.4 must have the form of Eq. (1.5). This modular framework allows us to construct complicated joint probabilities from components, and to compute marginal probabilities efficiently.

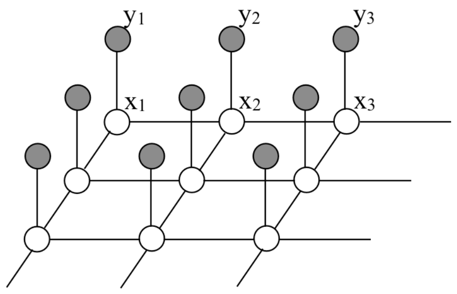
Now we examine some graphical models that are especially useful in vision. In perception problems, we typically have both observed and unobserved variables. The graph below shows a simple "Markov chain" structure with 3 observed variables, the shaded y variables above, and 3 unobserved variables, the x variables, below:



(1.10)

This is 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 variable 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 $P(x_1, x_2, x_3|y_1, y_2, y_3) = \frac{1}{P(\vec{y})}\phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\psi_1(y_1, x_1)\psi_2(y_2, x_2)\psi_3(y_3, x_3)$. (For brevity, we write $P(\vec{y})$ for $P(y_1, y_2, y_3)$.) Thus, to form the conditional distribution, within a normalization factor, we simply include the observed variable values into the joint probability. For vision applications, we sometimes have 1-d structures, such as the Markov chain shown above, often describing events over time.

To capture relationships over space we often use a 2-d structure such as that shown here:



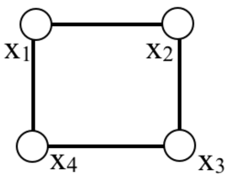
(1.11)

Then the joint probability over all the variables factorizes into a form like: $P(\vec{x}|\vec{y}) = \frac{1}{P(\vec{y})}\prod_{(i,j)}\phi_{ij}(x_i, x_j)\prod_i \psi_i(x_i, y_i)$, where the first product is over all spatial neighbors i and j, and the second product is over all nodes i.

1.2 Other types of graphical models

So far, we have discussed what are called **undirected graphical models**. Different types of graphical models are useful for describing different classes of probability distributions. There are different, but closely related, inference rules for each type of graphical model. Before describing how to perform inference in graphical models, we present two other classes of graphical models that can be applied to computer vision problems: factor graphs and directed models.

To motivate factor graphs, consider this graphical model,



(1.12)

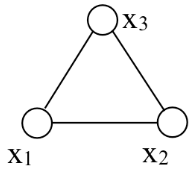
with the corresponding form of the joint probability being the product of the clique potentials of the four cliques of the graph:

$$P(x_1, x_2, x_3, x_4) = \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\phi_{34}(x_3, x_4)\phi_{41}(x_4, x_1) \tag{1.13}$$

Suppose we had a loop of three nodes? How would you construct a graphical model to specify a joint probability with this structure:

$$P(x_1, x_2, x_3) = \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\phi_{13}(x_1, x_3)? \tag{1.14}$$

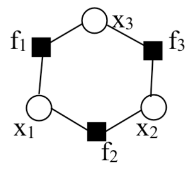
It would be tempting to consider the graph,



(1.15)

But that graph has a single maximal clique, the entire graph, and we can only describe this with the most general joint probability function, $P(x_1, x_2, x_3) = \phi_{123}(x_1, x_2, x_3)$.

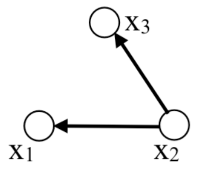
There are other types of graphical models, beyond the undirected models we have used so far, that make it easy to represent graphically the factorization of Eq. (1.14). A **factor graph**, most often used in coding theory, makes displaying such a factorization of the posterior probability easy. Factor graphs explicitly denote the factors in the joint probability:



(1.16)

A factor graph is a bipartite graph, with square nodes denoting functions, and round nodes variables that are input to the functions. The square nodes connect to variables that are the arguments of those functions. The joint probability described by a factor graph is 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 $P(x_1, x_2, x_3) = f_1(x_1, x_3) f_2(x_1, x_2) f_3(x_2, x_3)$, as desired.

In addition to undirected graphical models, and factor graphs, a third type of graphical model is 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 this graph:



(1.17)

denotes the joint probability,

$$P(x_1, x_2, x_3) = P(x_2) P(x_1|x_2) P(x_3|x_2) \tag{1.18}$$

The general rule for writing the joint probability described by a directed graph is this: Each node, x_n, contributes the factor $P(x_n|x_\Xi)$, where Ξ is the set of nodes with arrows pointing in to node x_n. You can verify that Eq. 1.18 follows this rule. Directed graphical models are often used to describe causal processes.

The table below summarizes the joint probability representation rules for each class of graphical model we discussed.

Graphical model	Probability depicted by graph
Undirected	$P_x(\vec{x}) = \frac{1}{Z} \prod_c \Psi_{x_c}(x_c)$
Factor graph	$P_x(\vec{x}) = \prod_a f_a(x_a)$
Directed	$P_x(\vec{x}) = \prod_\text{n} P_n(x_n \vert x_{\text{parents of n}})$

1.3 Some vision problems described using graphical models

1.3.1 1-d Markov chain for stereo matching

Suppose we have two stereo images, from calibrated cameras, and we want to compute the depth everywhere. As has been discussed earlier, knowing the calibration of a pair of cameras, the task of triangulating the depth of matching camera points reduces to computing the best-matching image points along a line in one of the two images. Let's assume we have a rectified image pair and we seek to find the displacement at each x position that yields the best matching image intensities.

Let the state that we seek to estimate be the unknown disparity at each pixel along a line in one image of the stereo pair. We assert that the probability of a given configuration of disparities along the line depends on two things: (1) the observed image intensities, I, and (2) the disparity of the neighboring pixels along the line. We can interpret this as a prior probability for the shapes of surfaces in the world.

Let the unknown state of each pixel in the line be its offset from the corresponding line in the other image. Let the disparity offset of pixel i be an integer d_i, one of M possible values.

For the line of N pixels, we can model the probability of some disparity configuration $\vec{d} = [d_1, d_2, \ldots, d_N]$ as

$$P(\vec{d})|I) = k \prod_i \phi_i(d_i) \prod_{(i,j)} \psi_{ij}(d_i, d_j) \tag{1.19}$$

where k is a normalization constant to make $P(\vec{d})|I)$ be a valid probability. The "local evidence" term, $\phi_i(d_i)$, might be written

$$\phi_i(d_i) = \exp \frac{(I_i - I_{i+d_i})^2}{2\sigma^2} \tag{1.20}$$

where σ is a parameter determined by the noise level.

One "pair-wise compatibility" term that favors adjascent pixels having the same disparity is the Potts model,

$$\psi_{ij}(d_i, d_j) = \begin{cases} \alpha, & \text{if } d_i = d_j \\ \beta, & \text{otherwise} \end{cases} \tag{1.21}$$

where α and β are constants and $\alpha > \beta$.

1.3.2 Detecting a person using a directed graph

Suppose we seek to detect a person in an image, and their pose. Let's say our person model has 4 sub-components, with individual detectors for each: Head (h), body (b), upper arm (u) and forearm (f). Let the observed image be I. We seek $P(h, b, u, f|I)$. By Bayes rule, we have $P(h, b, u, f|I) = P(I|h, b, u, f)P(h, b, u, f)$

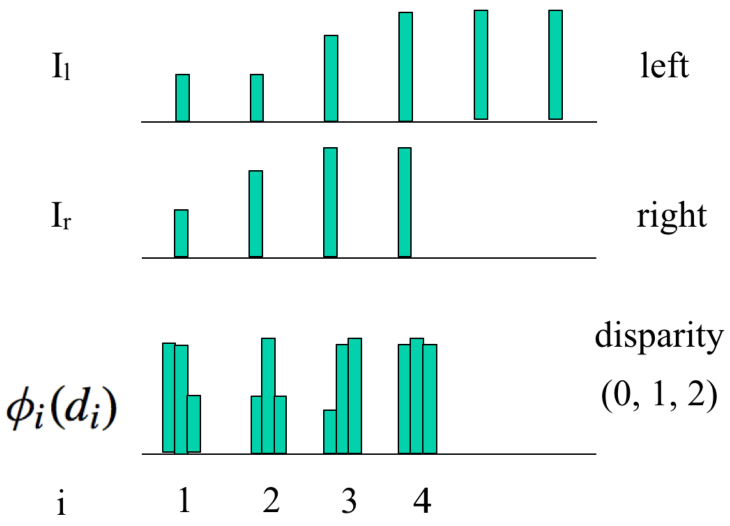


Figure 1.1
Probabilistic interpretation of the stereo problem. Based on local evidence alone, the optimal disparity states could be [0, 1, 2, 1]. But if the pairwise Potts model term uses $\alpha = 1$ and $\beta = 0.5$ then including the pairwise term penalizing local disparity state changes, makes the state [1,1,1,1] be about 8 times more probable under the model of Eq. (1.19).

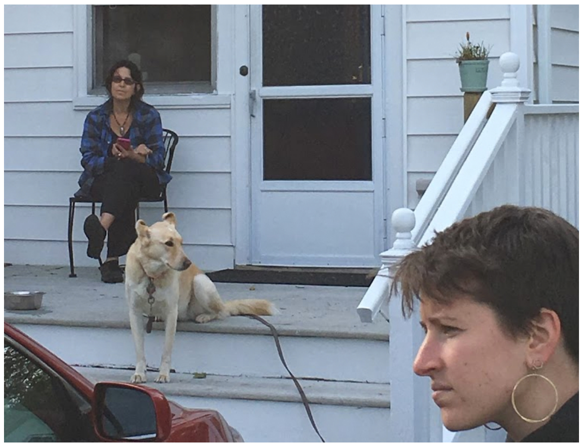


Figure 1.2
Image for which a prior model on body connectivity will be useful for accurate body pose estimation.

Because of the chain-like structure of the body, we can write the prior probability as

$$\begin{aligned} P(h,b,u,f) &= P(h|b,u,f)P(b,u,f) & (1.22)\\ &= P(h|b)P(b,|u)P(u,f) & (1.23)\\ &= P(h|b)P(b,|u)P(u|f)P(f) & (1.24) \end{aligned}$$

where we have repeatedly assumed that conditioning on a nearer body part superceeds conditioning on one further away. These pairwise probabilities are much easier to work with than the full joint probability.

1.4 Inference in graphical models

Now we explain how these graphical models are useful for inferring scenes from images.

Recall the Bayesian inference task: our observations are the elements of a vector, $\vec{y}$, and we seek to infer the probability $P(\vec{x}|\vec{y})$ of some scene parameters, $\vec{x}$, given the observations. By Bayes' rule, we have

$$P(\vec{x}|\vec{y}) = \frac{P(\vec{y}|\vec{x})P(\vec{x})}{P(\vec{y})} \qquad (1.25)$$

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. For inference, we often ignore the denominator, $P(\vec{y})$, called the *evidence*, as it is constant with respect to the variables we seek to estimate, $\vec{x}$.

Typically, we make many observations, $\vec{y}$, of the variables of some system, and we want to find the the state of some hidden variable, $\vec{x}$, given those observations. The posterior probability, $P(\vec{x}|\vec{y})$, tells us the probability for any value of the hidden variables, $\vec{x}$. From this posterior probability, we often want some single *best* estimate for $\vec{x}$, denoted $\hat{\vec{x}}$ and called a point estimate.

Selecting the best estimate $\hat{\vec{x}}$ requires specifying a penalty for making the wrong guess. If we penalize all wrong answers equally, the best strategy is to guess the value of $\vec{x}$ that maximizes the posterior probability, $P(\vec{x}|\vec{y})$ (because any other explanation $\hat{\vec{x}}$ for the observations $\vec{y}$ would be less probable). That is called the MAP estimate, for *maximum a posteriori*.

But we may want to penalize wrong answers as a function of how far away they are from the correct answer. If that penalty function is the squared distance in $\vec{x}$, then the point estimate that minimizes the average value of that error is called the minimum mean squared error estimate, or MMSE. To find this estimate, we seek the $\hat{\vec{x}}$ that minimizes the squared error, averaged over all outcomes:

$$\hat{\vec{x}}_{MMSE} = \text{argmin}_{\tilde{\vec{x}}} \int_{\vec{x}} P(\vec{x}|\vec{y})(\vec{x} - \tilde{\vec{x}})'(\vec{x} - \tilde{\vec{x}})d\vec{x} \tag{1.26}$$

Differentiating with respect to $\vec{x}$ to solve for the stationary point, the global minimum for this convex function, we find

$$\hat{\vec{x}}_{MMSE} = \int_{\vec{x}} \vec{x} P(\vec{x}|\vec{y}) d\vec{x}. \tag{1.27}$$

Thus, the minimum mean square error estimate, $\hat{\vec{x}}_{MMSE}$, is the mean of the posterior distribution. If $\vec{x}$ represents a discretized space, then the marginalization integrals over $d\vec{x}$ become sums over the corresponding states of $\vec{x}$.

For now, we'll assume we seek the MMSE estimate. By the properties of the multivariate mean, to find the mean at each variable, or node, in a network, we can first find the marginal probability at each node, then compute the mean of each marginal probability. In other words, given $P(\vec{x}|\vec{y})$, we will compute $P(x_i|\vec{y})$, where i is the ith hidden node. From $P(x_i|\vec{y})$ it is simple to compute the posterior mean at node i.

1.4.1 Simple example

To gain intuition, let's calculate the marginal probability for a simple example. Consider the three-node Markov chain of Line 1.10. This can correspond to the problem of inferring a label for a linear structure in an image, for example, identifying an object boundary. The observations $\vec{y}$ can be local evidence for the presence or absence of the occlusion boundary, and the variables $\vec{x}$ would be the inferred state.

We seek to marginalize the joint probability to find the marginal probability at node 1, $P(x_1|\vec{y})$. We have

$$P(x_1|\vec{y}) = \sum_{x_2} \sum_{x_3} P(x_1, x_2, x_3|\vec{y}) \tag{1.28}$$

Here's the important point: If we knew nothing about the structure of the joint probability $P(x_1, x_2, x_3|\vec{y})$, the computation would require $|x|^3$ computations: a double-sum over all x_2 and x_3 states must be computed for each possible x_1 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 as N grows.

But we can exploit the known structure of the joint probability of the variables in the chain, specified by the graphical model, to avoid the exponential growth of the computation with N. Substituting the joint probability, from the graphical model, into the marginalization equation, Eq. (1.29), gives

$$P(x_1|\vec{y}) = \frac{1}{P(\vec{y})} \sum_{x_2} \sum_{x_3} \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\psi_1(y_1, x_1)\psi_2(y_2, x_2)\psi_3(y_3, x_3) \quad (1.29)$$

Because of the form 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 lets us compute the marginalization much more efficiently. This will make only a small difference for this short chain, but it makes a huge difference for longer ones. We can write

$$\begin{aligned}
P(x_1|\vec{y}) &= \frac{1}{P(\vec{y})} \sum_{x_2} \sum_{x_3} \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\psi_1(y_1, x_1)\psi_2(y_2, x_2)\psi_3(y_3, x_3) && (1.30)\\
&= \frac{1}{P(\vec{y})}\psi_1(y_1, x_1) \sum_{x_2} \phi_{12}(x_1, x_2)\psi_2(y_2, x_2) \sum_{x_3} \phi_{23}(x_2, x_3)\psi_3(y_3, x_3) && (1.31)\\
&= \frac{1}{P(\vec{y})}\psi_1(y_1, x_1) \sum_{x_2} \phi_{12}(x_1, x_2)\psi_2(y_2, x_2)m_{32}(x_2) && (1.32)\\
&= \frac{1}{P(\vec{y})}\psi_1(y_1, x_1)m_{21}(x_1) && (1.33)
\end{aligned}$$

where we have defined the following terms as *messages*, partial sums that we will re-use later in computing the marginal probabilities at other nodes. m_{ij} is read as "the message from node i to node j". Note that the messages are always messages about the states of the node that the message is being sent *to*–the arguments of the message m_{ij} are the states x_j of node j. $m_{41}(x_1) = \psi_1(y_1, x_1)$, $m_{52}(x_2) = \psi_2(y_2, x_2)$, $m_{63}(x_3) = \psi_3(y_3, x_3)$, $m_{32}(x_2) = \sum_{x_3} \phi_{23}(x_2, x_3)m_{63}(x_3)$, 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. (1.33) reduces the number of terms summed from order $|x|^3$ to order $2|x|^2$, and in the case of a length N chain, from order $|x|^N$ to order $(N-1)|x|^2$, a huge computational savings for large N.

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. (1.33) in the marginalization to find $P_1(x_1)$. You would find that many of

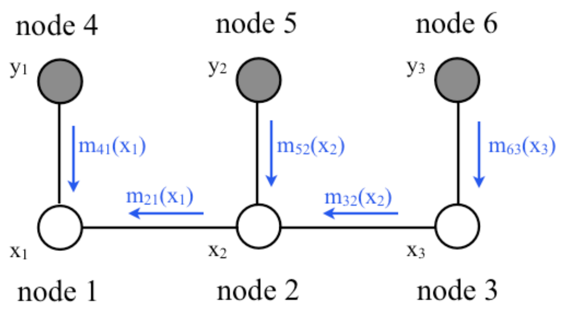


Figure 1.3
Summary of the messages (partial sums) for a simple belief propagation example.

the partial sums from marginalization at the first node would need to be recomputed for the marginalization at the second node, if they weren't stored as messages and to be re-used.

1.5 Belief propagation (BP)

To replace the manual marginalization that we did in the example above, it is preferable to have an automatic procedure for identifying the computations needed for marginalizations and to cache them efficiently. Belief propagation does that by identifying those reusable partial sums, which we call "messages". In finding the marginal probability at every node, a *message* is a re-usable partial sum from the marginalization calculations.

1.5.1 Message-passing rule

We'll describe belief propagation (BP) only for the special case of graphical models with pairwise potentials. Extensions to higher-order potentials is straightforward. (One way to proceed is to convert the graphical model into one with only 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.) You can find formal derivations of belief propagation in Jordan (1998); Koller and Friedman (2009).

Consider Fig. 1.4. There is a network of $N+1$ nodes, numbered 0 through N and we seek to marginalize over nodes $x_1 \dots x_N$. (Node x_0 and the connections to it, are not shown in the figure). Fig. 1.4 (a) shows the marginalization equation. If we distribute the marginalization sum past nodes for which the sum is a constant value, we obtain the sums depicted in Fig. 1.4 (b).

We assume the network of nodes to be a tree, and we define a *message*:

A message m_{ij} at node i is the sum over all states of all nodes in the subtree leaving node i at node j.

Referring to Fig. 1.4, (a) shows the desired marginalization sum over a network of pairwise cliques. (Again, this approach can be extended to more general cliques). Applying the factorization that took us from Eq. (1.30) to Eq. (1.31), we can pass the summations over node states through nodes that are constant over those summations, arriving at the equivalent marginalization depicted in (b). In (c), we have identified the appropriate partial sums as messages. From (c), we can read-off the belief propagation message update rule:

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 don't use that one in this calculation).
2. Multiply by the pairwise compatibility function $\psi_{ij}(x_i, x_j)$.
3. Marginalize over the variable x_j.

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

$$m_{ji}(x_i) = \sum_{x_j} \psi_{ij}(x_i, x_j) \prod_{k \in \eta(j) \backslash i} m_{kj}(x_j) \tag{1.34}$$

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

For the case of continuous variables, the sum over the states of x_j in Eq. (1.34) is replaced by an integral over the domain of x_j.

As mentioned above, BP messages are partial sums in the marginalization calculation. The arguments of messages are always the state of the node that the message is going to.

1.5.2 Marginal probability

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

$$P_i(x_i) = \prod_{j \in \eta(i)} m_{ji}(x_i) \tag{1.35}$$

You can verify that these rules lead to the same computations as for the marginalization of the example in Sect. 1.4.1.

1.5.3 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

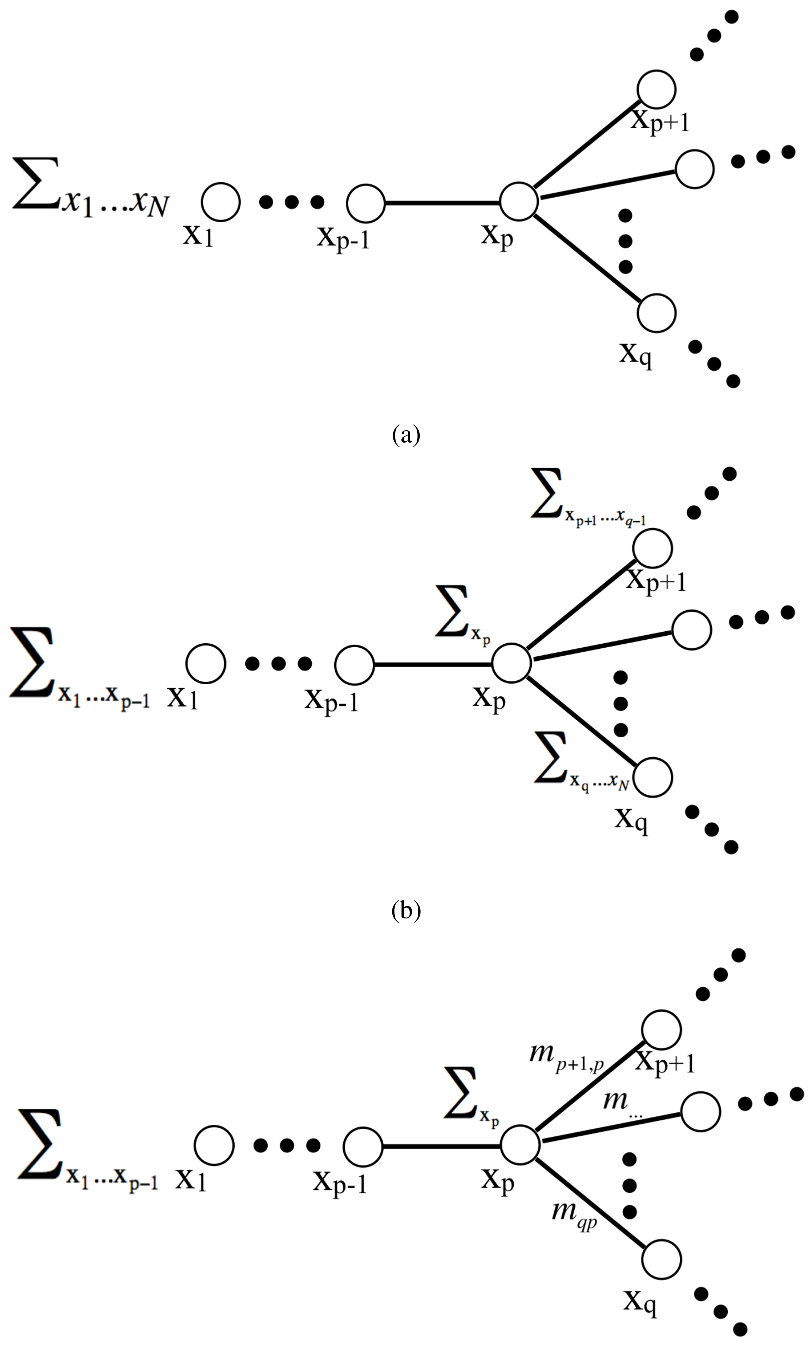


Figure 1.4
Example motivating the general formula for belief propagation.

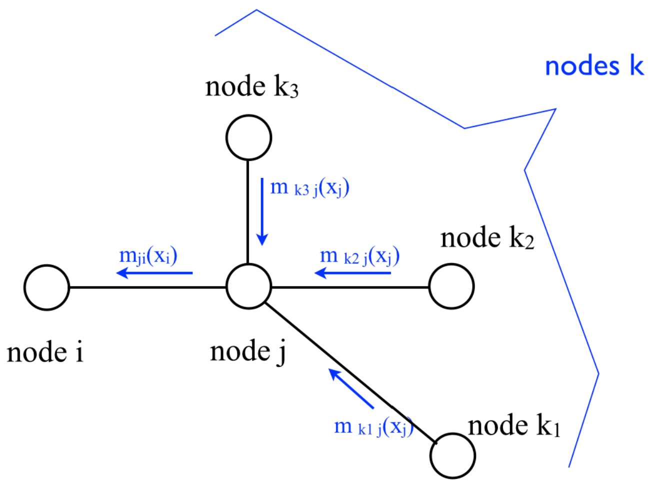


Figure 1.5
To pass a message from node j to node i.

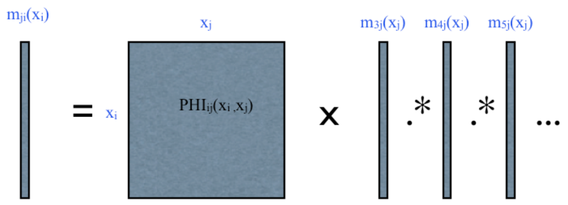


Figure 1.6
Pictorial depiction of belief propagation message passing rules, showing vector and matrix shapes.

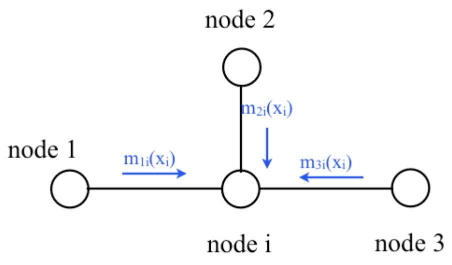


Figure 1.7
To compute the marginal probability at node i, we multiply together all the incoming messages at that node: $P_i(x_i) = \prod_{j \in \eta(i)} m_{ji}(x_i)$.

the message to you): "given everything I've heard, here's what I think is going on inside your house." That metaphor made sense to me after I started having teenaged children.

- The BP algorithm implies a local computational structure. If there are little computers attached to each node, they can perform these products and sums locally. Of course, you can also run the algorithm with a centralized processor, as well.

1.5.4 Message update sequence

When do we invoke the BP update rule, Eq. (1.34)? 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.

A node can send a message whenever all the incoming messages it needs have been computed. 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 1.8 depicts the flow of messages for the parallel, synchronous update scheme.

Note that, when computing marginals at many nodes, we re-use messages with the BP algorithm. 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. 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.

1.5.5 Numerical example

Finally, let's work though a numerical example. To make the arithmetic easy, we'll solve for the marginal probabilities in the graphical model of two-state (0 and 1) random variables shown in Fig. 1.10. 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).

$$\psi_{12}(x_1, x_2) = \begin{pmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{pmatrix} \tag{1.36}$$

$$\psi_{23}(x_2, x_3) = \begin{pmatrix} 0.1 & 1.0 \\ 1.0 & 0.1 \end{pmatrix} \tag{1.37}$$

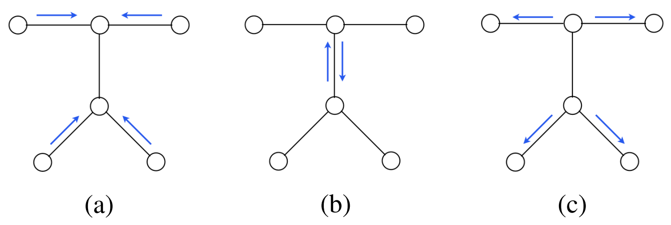


Figure 1.8
Example of a **synchronous parallel update schedule** for BP message passing. Whenever any node has the required incoming messages needed to send an outgoing message, it does. (a) At the first iteration, only the leaf nodes have the needed incoming messages to send an outgoing message (by 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.

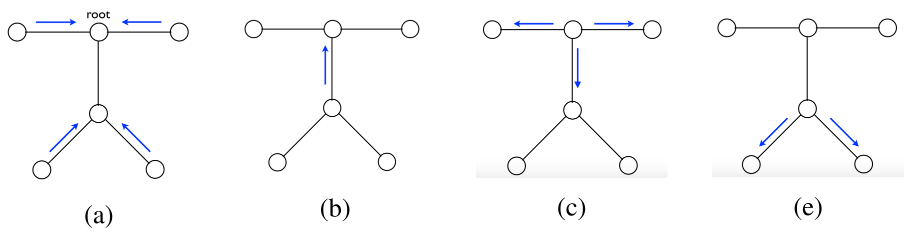


Figure 1.9
Example of a **depth-first update schedule** for BP message passing. We arbitrarily select one node as the root. (a) Messages begin at the leaves and (b) proceed to the root. Once all the messages reach the designated root node, (c) an outgoing sweep computes the remaining messages, (d) ending at the leaf nodes.

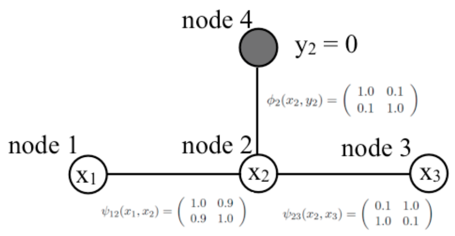

node 4
$y_2 = 0$
$\phi_2(x_2, y_2) = \begin{pmatrix} 1.0 & 0.1 \\ 0.1 & 1.0 \end{pmatrix}$
node 1
node 2
node 3
x_1
x_2
x_3
$\psi_{12}(x_1, x_2) = \begin{pmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{pmatrix}$
$\psi_{23}(x_2, x_3) = \begin{pmatrix} 0.1 & 1.0 \\ 1.0 & 0.1 \end{pmatrix}$

Figure 1.10
Details of numerical example.

$$\psi_{42}(x_2, y_2) = \begin{pmatrix} 1.0 & 0.1 \\ 0.1 & 1.0 \end{pmatrix} \tag{1.38}$$

Note that in defining these potential functions, we haven't taken care to normalize the joint probability, so we'll need to normalize each marginal probability at the end. (remember $P(x_1, x_2, x_3, y_2) = \psi_{42}(x_2, y_2)\psi_{23}(x_2, x_3)\psi_{12}(x_1, x_2)$, which should sum to 1 after summing over all states.)

For this simple toy example, 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 $y_2 = 0$, because $\psi_{42}(x_2, y_2)$ contributes a large valued to the posterior probability when $x_2 = y_2 = 1$ or when $x_2 = y_2 = 0$. From $\psi_{12}(x_1, x_2)$ we see that x_1 has a very 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. $\psi_{23}(x_2, x_3)$ indicates that 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

$$m_{12}(x_2) = \sum_{x_1} \psi_{12}(x_1, x_2) \tag{1.39}$$

$$= \begin{pmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{1.40}$$

$$= \begin{pmatrix} 1.9 \\ 1.9 \end{pmatrix} \tag{1.41}$$

$$= k \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{1.42}$$

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).

The message from node 3 to node 2 is

$$\begin{aligned} m_{32}(x_2) &= \sum_{x_3} \psi_{32}(x_2, x_3) && (1.43)\\ &= \begin{pmatrix} 0.1 & 1.0 \\ 1.0 & 0.1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (1.44)\\ &= \begin{pmatrix} 1.1 \\ 1.1 \end{pmatrix} && (1.45)\\ &= k \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (1.46) \end{aligned}$$

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

$$\begin{aligned} m_{42}(x_2) &= \sum_{x_4} \psi_{42}(x_2, y_2) && (1.47)\\ &= \begin{pmatrix} 1.0 & 0.1 \\ 0.1 & 1.0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} && (1.48)\\ &= \begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} && (1.49) \end{aligned}$$

where y_2 has been fixed to $y_2 = 0$, thus restricting $\psi_{42}(x_2, y_2)$ 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:

$$\begin{aligned} m_{21}(x_1) &= \sum_{x_2} \psi_{12}(x_1, x_2) m_{42}(x_2) m_{32}(x_2) && (1.50)\\ &= \begin{pmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{pmatrix} \left[\begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} .* \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right] = \begin{pmatrix} 1.09 \\ 1.0 \end{pmatrix} && (1.51) \end{aligned}$$

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:

$$\begin{aligned} m_{23}(x_3) &= \sum_{x_2} \psi_{23}(x_2, x_3) m_{42}(x_2) m_{12}(x_2) && (1.52)\\ &= \begin{pmatrix} 0.1 & 1.0 \\ 1.0 & 0.1 \end{pmatrix} \left[\begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} .* \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right] = \begin{pmatrix} 0.2 \\ 1.01 \end{pmatrix} && (1.53) \end{aligned}$$

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)

$$P_1(x_1) = km_{21}(x_1) = \frac{1}{2.09}\begin{pmatrix} 1.09 \\ 1.0 \end{pmatrix} \tag{1.54}$$

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:

$$\begin{aligned} P_2(x_2) &= km_{12}(x_2)m_{42}(x_2)m_{32}(x_2) && (1.55) \\ &= k\begin{pmatrix} 1 \\ 1 \end{pmatrix} .* \begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} .* \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (1.56) \\ &= \frac{1}{1.1}\begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} && (1.57) \end{aligned}$$

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

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

$$P_3(x_3) = km_{23}(x_3) = \frac{1}{1.21}\begin{pmatrix} 0.2 \\ 1.01 \end{pmatrix} \tag{1.58}$$

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.

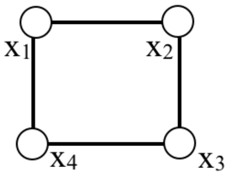


Figure 1.11
A probabilistic graphical model with a single loop.

1.6 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. 1.11), 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.

$$P_1(x_1) = \sum_{x_2}\sum_{x_3}\sum_{x_4} \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\phi_{34}(x_3, x_4)\phi_{41}(x_4, x_1) \tag{1.59}$$

$$P_1(x_1) = \sum_{x_2} \phi_{12}(x_1, x_2) \sum_{x_3} \phi_{23}(x_2, x_3) \sum_{x_4} \phi_{34}(x_3, x_4)\phi_{41}(x_4, x_1) \tag{1.60}$$

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 the belief propagation update rules of Sect. 1.5.1 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 Yedidia et al. (2001). In practice, the solutions found by the loopy belief propagation algorithm are often quite good.

1.6.0.1 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". Occasionally, the belief propagation updates can oscillate in a loopy graph. Dampening can be very useful to remove the oscillations and facilitate convergence to a fixed point.

The damped message at belief propagation message update iteration k, $\bar{m}^k_{ji}(x_i)$, 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 $k-1$:

$$\bar{m}^k_{ji}(x_i) = \alpha m^k_{ji}(x_i) + (1-\alpha) m^{k-1}_{ji}(x_i) \tag{1.61}$$

1.7 MAP estimation and energy models

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.

Bibliography

Jordan, M. I., ed. 1998. *Learning in graphical models*. MIT Press.

Koller, D., and N. Friedman, eds. 2009. *Probabilistic graphical models*. MIT Press.

Yedidia, J. S., W. T. Freeman, and Y. Weiss. 2001. Generalized belief propagation, Vol. 13, 689–695. MIT Press.