The Tiny Book of Computer Vision

Antonio Torralba, Bill Freeman, Phillip Isola

October 7, 2019

Contents

Chapter 2

Probabilistic graphical 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 useful in vision, where we often need to exploit modularity to make computations tractable.

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 through a set of examples.

2.1 Simple Examples

Here is the simplest probabilistic graphical model:

(2.1)

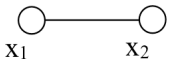
This "graph" is just a single node for the variable x_1. There is no restriction imposed by the graph on the functional form of its probability function. In keeping with notation we'll use later, we write that the probability distribution over x_1, $P(x_1) = \phi_1(x_1)$.

Another trivial graphical model is this:

(2.2)

Here, the lack of a line connecting x_1 with x_2 indicates a lack of statistical dependence between these two variables. In detail, we condition on knowing the values of any connecting neighbors of x_1 and x_2–in this case there are none–and thus x_1 and x_2 are statistically independent. Because of that independence, the joint probability depicted by this graphical model must be a product of each variable's marginal probability and thus must have this form: $P(x_1, x_2) = \phi_1(x_1)\phi_2(x_2)$.

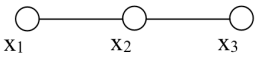
Let's add a line between these two variables:



(2.3)

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:

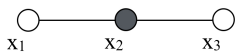


(2.4)

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

$$P(x_1, x_2, x_3) = \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3). \tag{2.5}$$

Note the conditional independence implicit in Eq. (2.5): if the value of x_2 is given (indicated by the solid circle in the graph below), then the joint probability of Eq. (2.5) becomes a product of some function $f(x_1)$ times some other function $g(x_3)$, revealing the independence of x_1 and x_3.



(2.6)

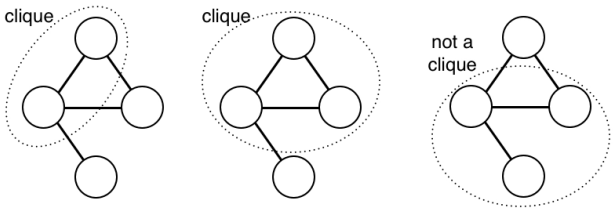
$P(x_1, x_2, x_3)$ then becomes

$$P(x_1, x_2, x_3) = \phi_{12}(x_1, \bar{x}_2)\phi_{23}(\bar{x}_2, x_3) = f(x_1)g(x_3) \tag{2.7}$$

where $\bar{x}_2$ is the observed value of x_2. In the next chapter, we'll exploit that structure of the joint probability to perform inference efficiently using belief propagation.

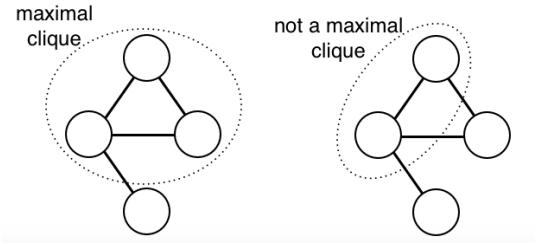
A celebrated theorem, the *Hammersley-Clifford theorem*, tells the form the joint probability must have for any given graphical model. The joint probability for a probabilistic graphical model must be a product of functions of each the "maximal cliques" of the graph. Here we define the terms.

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:



(2.8)

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):



(2.9)

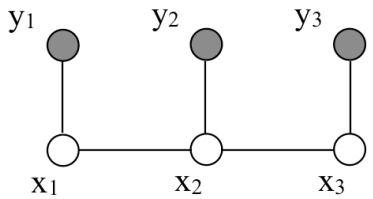
The *Hammersley-Clifford theorem*: A positive probability distribution has the independence structure described by a graphical model if and only if it can be written as a product of functions over the variables of each maximal clique:

$$P(x_1, x_2, \ldots, x_N) = \prod_{x_c \in x_i} \Psi_{x_c}, \tag{2.10}$$

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

In the example of the graph 2.4, the maximal cliques are (x1, x2) and (x2, x3), so the Hammersley-Clifford theorem says that the corresponding joint probability must be of the form of Eq. (2.5).

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, shaded, labeled y_i, and 3 unobserved variables, labeled x_i:



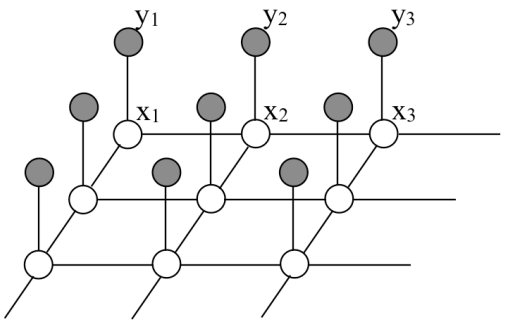
(2.11)

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 the probability of the x variables conditioned on the observations y,

$$P(x_1, x_2, x_3|y_1, y_2, y_3) = \frac{1}{P(\vec{y})}\phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\psi_1(y_1, x_1)\psi_2(y_2, x_2)\psi_3(y_3, x_3). \tag{2.12}$$

(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 often use such Markov chain structures to describe events over time.

To capture relationships over space, a 2-d structure is useful:



(2.13)

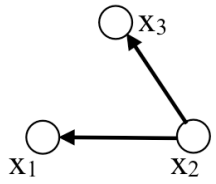
Then the joint probability over all the variables factorizes into this product:

$$P(\vec{x}|\vec{y}) = \frac{1}{P(\vec{y})} \prod_{(i,j)} \phi_{ij}(x_i, x_j) \prod_i \psi_i(x_i, y_i), \tag{2.14}$$

where the first product is over all spatial neighbors i and j, and the second product is over all nodes i.

2.2 Directed graphical models

In addition to undirected graphical models, another 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 on all the variables originating arrows pointing into it. So this graph:



(2.15)

denotes the joint probability,

$$P(x_1, x_2, x_3) = P(x_2)P(x_1|x_2)P(x_3|x_2) \tag{2.16}$$

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. 2.16 follows this rule. Node x_2 has no arrows going into it, so contributes the factor $P(x_2)$, with no conditioning. Directed graphical models are often used to describe causal processes.

Chapter 3

Inference in graphical models

Given a probabilistic graphical model and observations, we want to estimate the states of the unobserved variables. For example, given image observations, we want to efficiently estimate the pose of the person. The belief propagation algorithm lets us do that efficiently.

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

$$P(\vec{x}|\vec{y}) = \frac{P(\vec{y}|\vec{x})P(\vec{x})}{P(\vec{y})} \tag{3.1}$$

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 $\vec{x}$ occurring. 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 a 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 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:

$$\hat{\vec{x}}_{MMSE} = \text{argmin}_{\tilde{\vec{x}}} \int_{\vec{x}} P(\vec{x}|\vec{y})(\vec{x} - \tilde{\vec{x}})'(\vec{x} - \tilde{\vec{x}})d\vec{x} \tag{3.2}$$

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

$$\hat{\vec{x}}_{MMSE} = \int_{\vec{x}} \vec{x} P(\vec{x}|\vec{y})d\vec{x}. \tag{3.3}$$

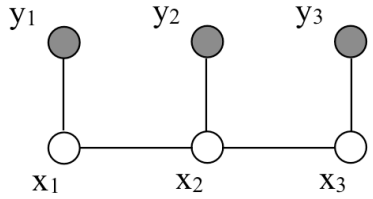
Thus, the minimum mean square error estimate, $\hat{\vec{x}}_{MMSE}$, is the mean of the posterior distribution. If $\vec{x}$ represents a discretized space, then the marginalization integrals over $d\vec{x}$ become sums over the corresponding discrete states of $\vec{x}$.

For now, we'll assume we seek the MMSE estimate. By the properties of the multi-variate 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.

3.1 Simple example of inference in a graphical model

To gain intuition, let's calculate the marginal probability for a simple example. Consider the three-node Markov chain of 3.4, reproduced here.



(3.4)

This can model the probability of a pixel belonging to an object boundary, given evidence, observations $\vec{y}$, at a point and two neighboring location. The inferred states $\vec{x}$ could be a label indicating an edge.

We seek to marginalize the joint probability to find the marginal probability at node 1, $P(x_1|\vec{y})$, given obervations y_1, y_2, and y_3, which we write as $\vec{y}$. We have

$$P(x_1|\vec{y}) = \sum_{x_2}\sum_{x_3} P(x_1, x_2, x_3|\vec{y}) \tag{3.5}$$

Here's the main 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 quickly becomes intractable as N grows.

But we can exploit the model's structure to avoid the exponential growth of the computation with N. Substituting the joint probability, from the graphical model, into the marginalization equation, Eq. (3.6), gives

$$P(x_1|\vec{y}) = \frac{1}{P(\vec{y})}\sum_{x_2}\sum_{x_3} \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\psi_1(y_1, x_1)\psi_2(y_2, x_2)\psi_3(y_3, x_3) \tag{3.6}$$

This form for the joint probability says that not every variable is coupled to every other one. We can pass summations through variables they doesn't sum over, letting 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. So we write

$$\begin{aligned} P(x_1|\vec{y}) &= \frac{1}{P(\vec{y})}\sum_{x_2}\sum_{x_3} \phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\psi_1(y_1, x_1)\psi_2(y_2, x_2)\psi_3(y_3, x_3) &(3.7)\\ &= \frac{1}{P(\vec{y})}\psi_1(y_1, x_1)\sum_{x_2} \phi_{12}(x_1, x_2)\psi_2(y_2, x_2)\sum_{x_3} \phi_{23}(x_2, x_3)\psi_3(y_3, x_3) &(3.8)\end{aligned}$$

That factorization of Eq. (3.8) is the key step in "belief propagation" It reduces the number of terms summed from order $|x|^3$ to order $2|x|^2$, and for a chain of length N chain, from order $|x|^N$ to order $(N-1)|x|^2$, a huge computational savings for large N.

Because we may re-use them, we name the partial sums of Eq. (3.8) "messages". We call the message from node 3 to node 2 $m_{32}(x_2) = \sum_{x_3} \phi_{23}(x_2, x_3)m_{63}(x_3)$. The other partial sum is the message from node 2 to node 1, $m_{21}(x_1) = \sum_{x_2} \phi_{12}(x_1, x_2)m_{52}(x_2)m_{32}(x_2)$.

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.

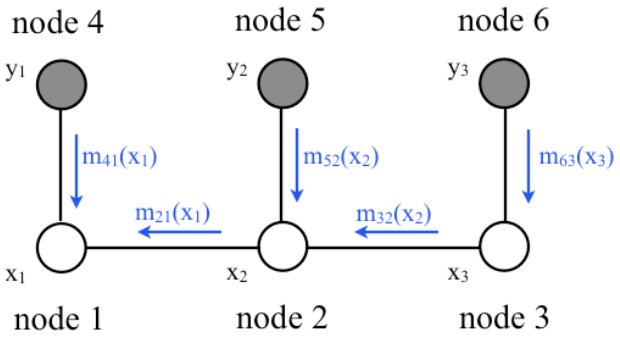


Figure 3.1: Summary of the messages (partial sums) for a simple belief propagation example.

Eq. (3.8) giave us the marginal probability at node 1. To find the marginal probability at another node, you can write out the sums over variables needed for that node, and pass the sums through factors in the joint probability that they don't operate on, to come up with an efficient reorganization of summations analogous to Eq. (3.8). You would find that many of the summations from marginalization at the node x_1 would need to be recomputed for the marginalization at node x_2. That motivates storing and reusing the messages.

3.2 Belief propagation (BP)

For more complicated graphical models, we want to replace the manual factorization with an automatic procedure for identifying the computations needed for marginalizations and to cache them efficiently. Belief propagation does that by identifying those reusable sums, the "messages".

3.2.1 Derivation of message-passing rule

We'll describe belief propagation (BP) only for the special case of graphical models with pairwise potentials. The clique potentials between neighboring nodes are $\psi_{ij}(x_j, x_i)$. Extensions to higher-order potentials is straightforward. (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 [**?**, **?**].

Consider Fig. 3.2. There is a network of $N + 1$ nodes, numbered 0 through N and we will marginalize over nodes $x_1 \dots x_N$. Fig. 3.2 (a) shows the marginalization equation. If we assume the nodes form a tree, we can distribute the marginalization sum past nodes for which the sum is a constant value to obtain the sums depicted in Fig. 3.2 (b).

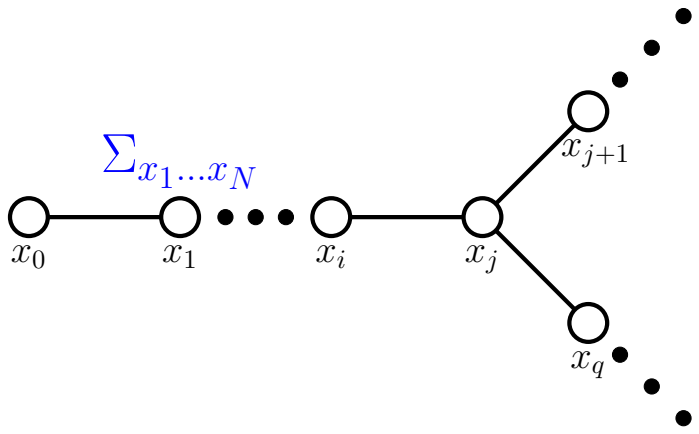
We define a *belief propagation message*:

A message m_{ij}, from node i to node j, is the sum of the probability over all states of all nodes in the subtree that leaves node i and does not include node j.

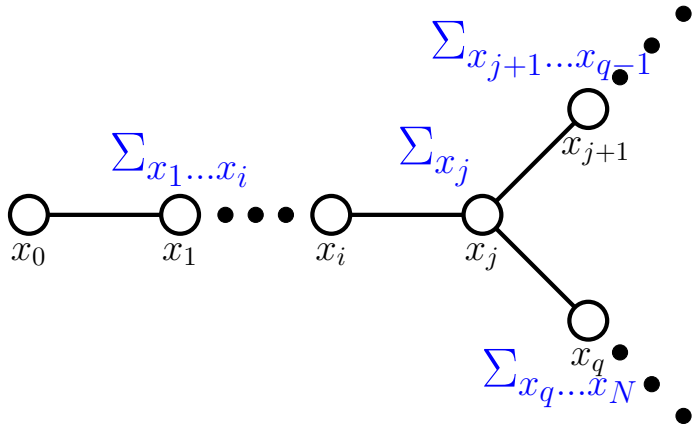
Referring to Fig. 3.2, (a) shows the desired marginalization sum over a network of pairwise cliques. (This approach can be extended to more general cliques). We can pass the summations over node states through nodes that are constant over those summations, arriving at the factorization shown in (b). Remembering that the message from node i to node j is the sum over the tree leaving node i, we can read-off the recursive belief propagation message update rule from Fig. 3.2:

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,



(a)



(b)

Figure 3.2: Example motivating the formula for belief propagation.(a) shows the marginalization for a general graph, focussing on the partial sums at note x_j. (b) Shows how the partial sums distribute over the nodes

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:

$$m_{ji}(x_i) = \sum_{x_j} \psi_{ij}(x_i, x_j) \prod_{k \in \eta(j) \backslash i} m_{kj}(x_j) \tag{3.9}$$

where $\eta(j) \backslash i$ means "the neighbors of node j except for node i". Figure 3.3 shows this equation in a graphical form. Any local potential functions $\phi_j(x_j)$ are treated as an additional message into node j, $m_{0j}(x_j) = \phi_j(x_j)$. For the case of continuous variables, the sum over the states of x_j in Eq. (3.9) 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.

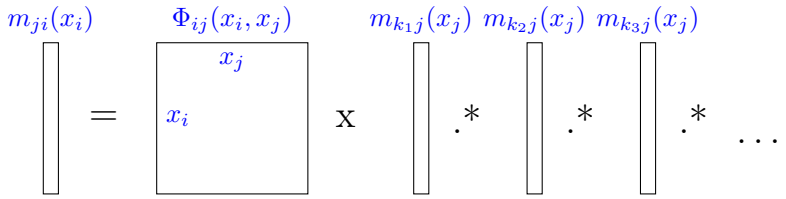


Figure 3.3: Pictorial depiction of belief propagation message passing rules of Eq. (3.9), showing vector and matrix shapes. To send a message from node j to node i: We term-by-term multiply (shown by .*) the messages (column vectors) coming in to node j, then matrix multiply (shown by x) the resulting column vector by the compatibility matrix $\Phi_{ij}(x_i, x_j)$ to obtain $m_{ji}(x_i)$.

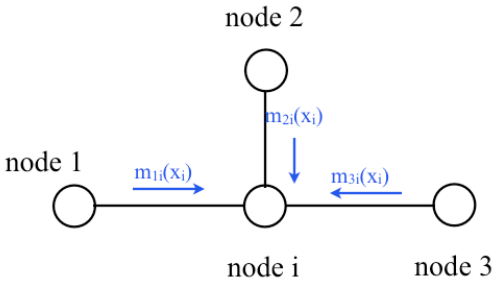


Figure 3.4: To compute the marginal probability at node i, we multiply together all the incoming messages at that node: $P_i(x_i) = \prod_{j \in \eta(i)} m_{ji}(x_i)$, remembering to include any local potential terms $\phi_i(x_i)$ as another message.

3.2.2 Marginal probability

The marginal probability at a node i is the sum over all states except those of x_i. That is the product (because of the conditional independence structure) of the sum of all states of all nodes in each tree leaving node i. Thus the marginal probability at node i is the product of all the incoming messages:

$$P_i(x_i) = \prod_{j \in \eta(i)} m_{ji}(x_i) \tag{3.10}$$

(We include the local clique potential at node i, $\psi_i(x_i)$, as one of the messages in the product of all messages into node i).

3.2.3 Message update sequence

So we need to find the messages. When do we invoke the recursive BP update rule, Eq. (3.9)? Whenever all the incoming messages in the BP update rule are defined. If there are no incoming messages to a node, then its outgoing message is well-defined in the update rule. This lets us start the recursive algorithm.

A node can send a message whenever all the incoming messages it needs have been computed. We can start by computing the outgoing messages from leaf nodes in the graphical model tree, 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 3.5 depicts the flow of messages for the parallel, synchronous update scheme.

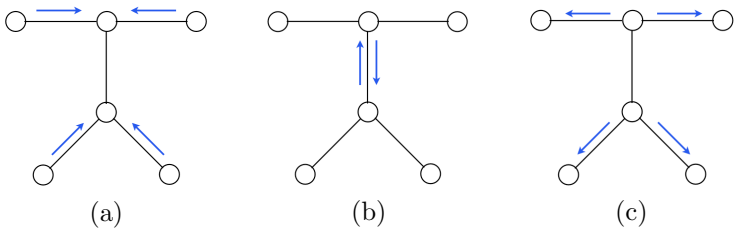


Figure 3.5: Example of a **synchronous parallel update schedule** for BP message passing. Whenever any node has the required incoming messages needed to send an outgoing message, it does. (a) At the first iteration, only the leaf nodes have the needed incoming messages to send an outgoing message (by 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.

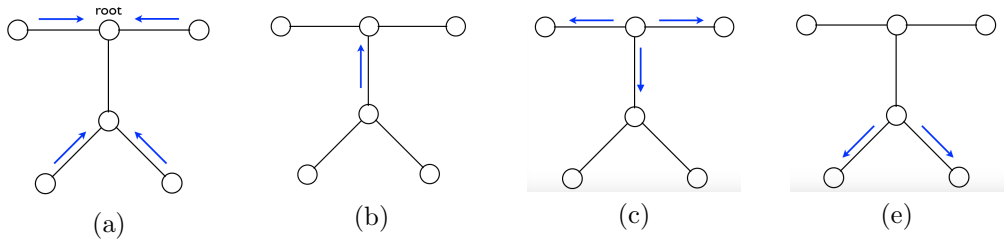


Figure 3.6: Example of a **depth-first update schedule** for BP message passing. We arbitrarily select one node as the root. (a) Messages begin at the leaves and (b) proceed to the root. Once all the messages reach the designated root node, (c) an outgoing sweep computes the remaining messages, (d) ending at the leaf nodes.

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 incoming messages to every node as it does to calculate the incoming messages at a single node.

3.3 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. 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. 3.2.1 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 [**?**]. In practice, the solutions found by the loopy belief propagation algorithm are often quite good.

3.4 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.

Improvements have been developed over loopy belief propagation for the case of MAP estimation, see, for example, tree-reweighted belief propagation,

```
https://pub.ist.ac.at/~vnk/papers/TRW-S-PAMI.pdf
Published in: IEEE Transactions on Pattern Analysis and Machine
Intelligence ( Volume: 28 , Issue: 10 , Oct. 2006 )
V. Kolmogorov
```

and graph cuts,

```
http://www.cs.cornell.edu/rdz/Papers/KZ-ECCV02-graphcuts.pdf
What Energy Functions Can Be Minimizedvia Graph Cuts?
February 2004, pp. 147-159, vol. 26
Authors
Ramin Zabih, IEEE
Vladimir Kolmogorov, IEEE
```

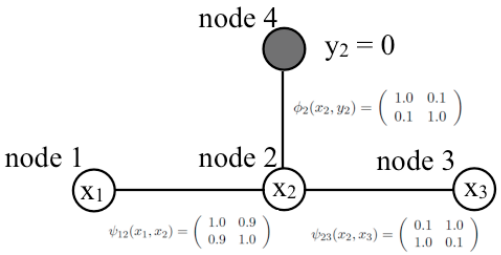


Figure 3.7: Details of numerical example.

3.4.1 Appendix: Numerical example of Belief Propagation

Here we work though a numerical example of belief propagation. 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. 3.7. 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).

$$\psi_{12}(x_1, x_2) = \begin{pmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{pmatrix} \tag{3.11}$$

$$\psi_{23}(x_2, x_3) = \begin{pmatrix} 0.1 & 1.0 \\ 1.0 & 0.1 \end{pmatrix} \tag{3.12}$$

$$\psi_{42}(x_2, y_2) = \begin{pmatrix} 1.0 & 0.1 \\ 0.1 & 1.0 \end{pmatrix} \tag{3.13}$$

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

$$\begin{aligned} m_{12}(x_2) &= \sum_{x_1} \psi_{12}(x_1, x_2) && (3.14) \\ &= \begin{pmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (3.15) \\ &= \begin{pmatrix} 1.9 \\ 1.9 \end{pmatrix} && (3.16) \\ &= k \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (3.17) \end{aligned}$$

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).

The message from node 3 to node 2 is

$$\begin{aligned} m_{32}(x_2) &= \sum_{x_3} \psi_{32}(x_2, x_3) && (3.18)\\ &= \begin{pmatrix} 0.1 & 1.0 \\ 1.0 & 0.1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (3.19)\\ &= \begin{pmatrix} 1.1 \\ 1.1 \end{pmatrix} && (3.20)\\ &= k \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (3.21) \end{aligned}$$

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

$$\begin{aligned} m_{42}(x_2) &= \sum_{x_4} \psi_{42}(x_2, y_2) && (3.22)\\ &= \begin{pmatrix} 1.0 & 0.1 \\ 0.1 & 1.0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} && (3.23)\\ &= \begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} && (3.24) \end{aligned}$$

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:

$$\begin{aligned} m_{21}(x_1) &= \sum_{x_2} \psi_{12}(x_1, x_2) m_{42}(x_2) m_{32}(x_2) && (3.25)\\ &= \begin{pmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{pmatrix} \left[\begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} .* \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right] = \begin{pmatrix} 1.09 \\ 1.0 \end{pmatrix} && (3.26) \end{aligned}$$

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:

$$\begin{aligned} m_{23}(x_3) &= \sum_{x_2} \psi_{23}(x_2, x_3) m_{42}(x_2) m_{12}(x_2) && (3.27)\\ &= \begin{pmatrix} 0.1 & 1.0 \\ 1.0 & 0.1 \end{pmatrix} \left[\begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} .* \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right] = \begin{pmatrix} 0.2 \\ 1.01 \end{pmatrix} && (3.28) \end{aligned}$$

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)

$$P_1(x_1) = k m_{21}(x_1) = \frac{1}{2.09} \begin{pmatrix} 1.09 \\ 1.0 \end{pmatrix} \quad (3.29)$$

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:

$$\begin{aligned} P_2(x_2) &= k m_{12}(x_2) m_{42}(x_2) m_{32}(x_2) && (3.30)\\ &= k \begin{pmatrix} 1 \\ 1 \end{pmatrix} .* \begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} .* \begin{pmatrix} 1 \\ 1 \end{pmatrix} && (3.31)\\ &= \frac{1}{1.1} \begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} && (3.32) \end{aligned}$$

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

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

$$P_3(x_3) = km_{23}(x_3) = \frac{1}{1.21}\begin{pmatrix} 0.2 \\ 1.01 \end{pmatrix} \tag{3.33}$$

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.