CS281A/Stat241A: Statistical Learning Theory

Lecture 6: Factor Graphs

Lecturer: Michael I. Jordan *Scribes: L.Subramanian and D.Mandelin*

Overview of the Lecture:
a) Sum-Product algorithm for directed trees b) Factor graph representations c) Sum-product algorithm for factor trees d) Tree-like graphs and polytrees

1 Sum-Product Algorithm for Directed Trees

Let V and $\mathcal{E}$ denote the nodes and edges of a graphical model. Let X_E denote the set of evidence nodes in this graphical model. Let $N(j)$ represent the set of neighbors of node j. In the case of an undirected tree, the sum product algorithm provides a way of estimating $p(x|x_E)$ to be:

$$p(x|x_E) = \frac{1}{Z}\Pi_{v\in V}\Psi^E(x_v)\Pi_{(i,j)\in\mathcal{E}}\Psi(x_i,x_j) \tag{1}$$

Applying the message-passing protocol of the sum-product algorithm (from previous class), we also obtain:

$$m_{ji}(x_i) = \Sigma_{x_j}\Psi^E(x_j)\Psi(x_i,x_j)\Pi_{k\in N(j)-\{i\}}m_{kj}(x_j) \tag{2}$$

We note that $p(x_i)$ is proportional to $\Pi_{j\in N(i)}m_j(x_i)$. For an undirected tree, the direction of message passing respects the structure of the tree. To infer $p(x_f|X_E)$, we consider the tree rooted at f. Given this orientation property, the algorithm specified for undirected trees also works for directed trees.

The ordering induced is equivalent to the elimination ordering ending at the root. The ordering also eliminates the parent only after eliminating all its children. Hence based on the elimination algorithm, we can infer that $p(root)$ is correct. In the sum-product algorithm, the message $m_{ji}(x_i)$ is produced by eliminating all sub-trees of the node x_i. By using induction on the size of the tree and these two observations, we can show that the sum-product algorithm also works for directed trees.

Re-using messages: The concept that "messages" can be reused is important. A message computed for determining $p(x_i|x_E)$ can be reused for computing $p(x_j|x_E)$. We will illustrate this using an example. Consider a directed version of the tree shown in Figure 1 rooted at X_1. We first calculate $P(x_1|x_E)$. Now, we proceed to X_2, X_3 and X_4 in that order. While calculating $P(x_2|x_E)$, we can re-use the results $m_{32}(x_2)$ and $m_{42}(x_2)$ used for computing for X_1. These messages could be cached and the computation of $p(x|x_E)$ for multiple nodes x in the graph can be much simplified. The number of messages is very small compared to the number of elimination orderings. Figure 1 illustrates the list of messages that may be exchanged in a simple four-node graph. The total number of messages is equal to twice the number of edges in the graphical model.

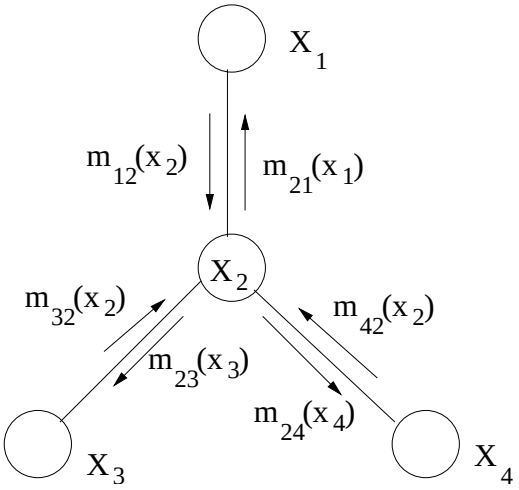


Figure 1: All messages that can be potentially exchanged in a simple four node graph.

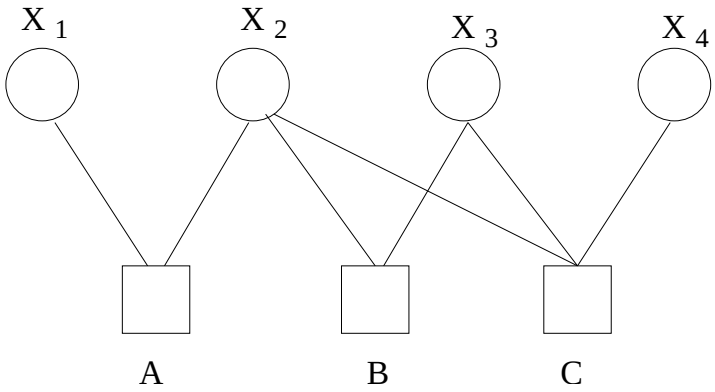


Figure 2: A simple factor graph

2 Factor graphs

Factor graphs are alternative graphical representations of probability distributions that capture factorizations. A factor graph is a bipartite graph with two types of nodes:variable nodes and factor nodes. Figure 2 contains the factor graph representation of the following factorized expression:

$$f(x_1, x_2, x_3, x_4) = f_A(x_1, x_2) f_B(x_2, x_3) f_C(x_2, x_3, x_4) \tag{3}$$

A factor graph has an explicit node for every factor in the expression (as opposed to just the normal nodes). Every square node in the figure, namely A, B, C, represent one of the factor terms in the expression. It is possible to construct a factor graph representation of both directed and undirected graphical models. While directed graphical models capture conditional dependencies, factor graphs capture factorizations.

Factor graphs provide a more fine-grained representation of probability distributions than directed or undirected graphical models. We will illustrate this by considering three factorized representations of probability distributions on three variables.

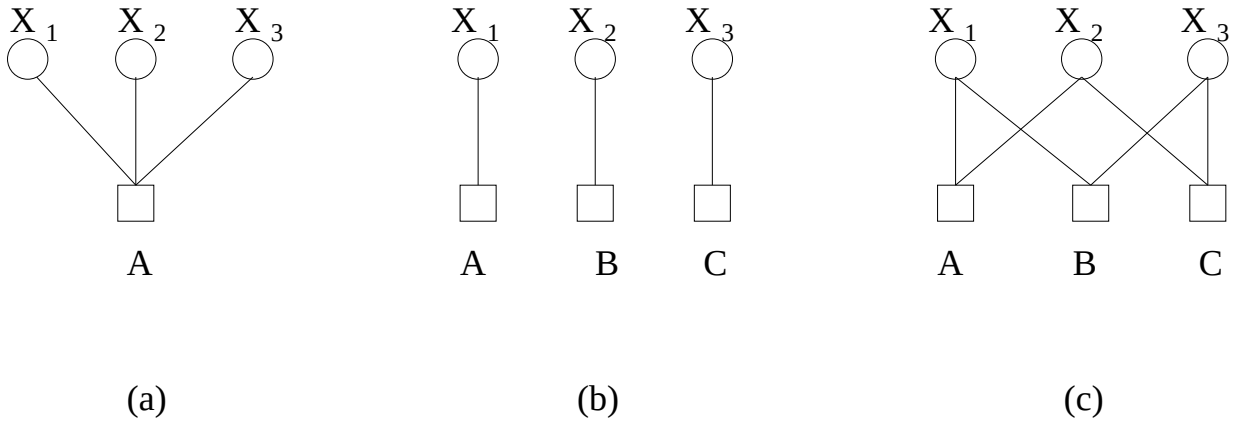


Figure 3: Factor graph representation for the 3 expressions. (a) Expression E1 (b) Expression E2 (c) Expression E3

$$E1 : f(x_1, x_2, x_3) = \Psi(x_1, x_2, x_3) \tag{4}$$
$$E2 : f(x_1, x_2, x_3) = \Psi(x_1)\Psi(x_2)\Psi(x_3) \tag{5}$$
$$E3 : f(x_1, x_2, x_3) = \Psi(x_1, x_2)\Psi(x_1, x_3)\Psi(x_2, x_3) \tag{6}$$

Figure 3 shows the factor representations of these three expressions. While we have simple undirected graph representations of expressions $E1$ and $E2$, the simplest graphical model that captures $E3$ is simply a clique on 3 nodes. An undirected graphical model provides no information about possible factorizations of the potential function associated with a clique. $E3$ is an example of one such expression which cannot be captured by a simple clique.

However, it is possible to capture fine grain representation of factor graphs within the directed and undirected formalisms. So factor graphs, as such provide no additional power. Figure 4(a) shows an undirected graph that represents the factorization of $E3$. While this graph resembles the factor graph representation of $E3$, the random variables introduced should be viewed as simply indicator variables. For example, consider the variable Z_1. For binary X_1 and X_2, Z_1 would take four possible values and the potential function $\Psi(z_1)$ would be set to the corresponding value, $f_A(x_1, x_2)$.

Similarly, Figure 4(b) represents the directed graphical model which captures the factorization of $E3$. In this model, the new variables introduced W_1, W_2 and W_3 are always set to 1. The probability $p(W_1 = 1|x_1, x_2)$ is set to the corresponding value $f_A(x_1, x_2)$.

Hence, factor graph representations do not have any additional power over directed or undirected graphical representations.

Applications of Each Graphical Model: Undirected graphical models are used extensively in Physics. (e.g. lattice graphs). Directed graphs are used in genetics and expert systems. Factor graph representations are useful in error-correction codes. The factors in these graphs represent the parity bits.

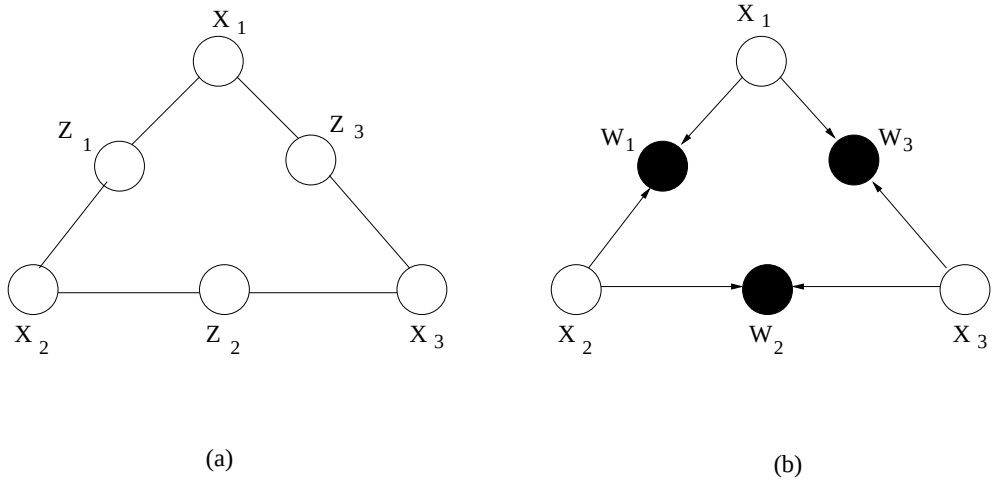
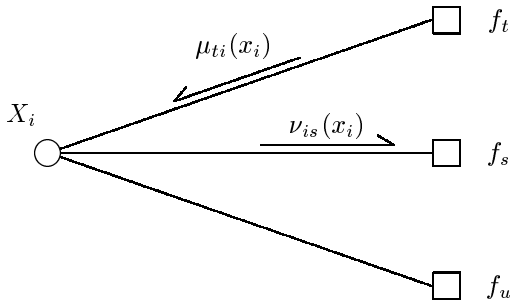


Figure 4: Equivalent directed and undirected graph representations for Expression E3. (a) Undirected graph representation (b) Directed Graph representation

3 Sum-Product Algorithm for Factor Trees

A *factor tree* is a factor graph that is a tree. The sum-product algorithm may be applied to a factor tree. When the algorithm is expressed in terms of the factors, it takes the form given below. In this form, there are two kinds of messages: Factor nodes send μ-messages to variable nodes, and variable nodes send ν-messages to factor nodes.



$$\nu_{is}(x_i) = \prod_{t \in \eta(i) - \{s\}} \mu_{ti}(x_i) \tag{7}$$

$$\mu_{si}(x_i) = \sum_{x_{\eta(s)-\{i\}}} \Big(f_s(x_{\eta(s)}) \prod_{j \in \eta(s) - \{i\}} \nu_{js}(x_j) \Big) \tag{8}$$

The underlying message-protocol is the same as the sum-product algorithm for directed or undirected trees: *Every node can send a message to its neighbor only after it has received messages from all its other neighbors.* This rule applies to both factor nodes and the variable nodes. Once a node has received messages from all its neighbors, the marginal probability of a node is calculated as:

$$p(x_i) \propto \prod_{s \in \eta(i)} \mu_{si}(x_i) \tag{9}$$

4 Tree-like graphs and Poly-trees

If the original graphical model is a tree(undirected or directed), we have little gain in translating the graph to a factor graph. The Sum-product algorithm for factor trees is particularly useful if the underlying graphical model is not a tree while its factor graph representation is a tree.

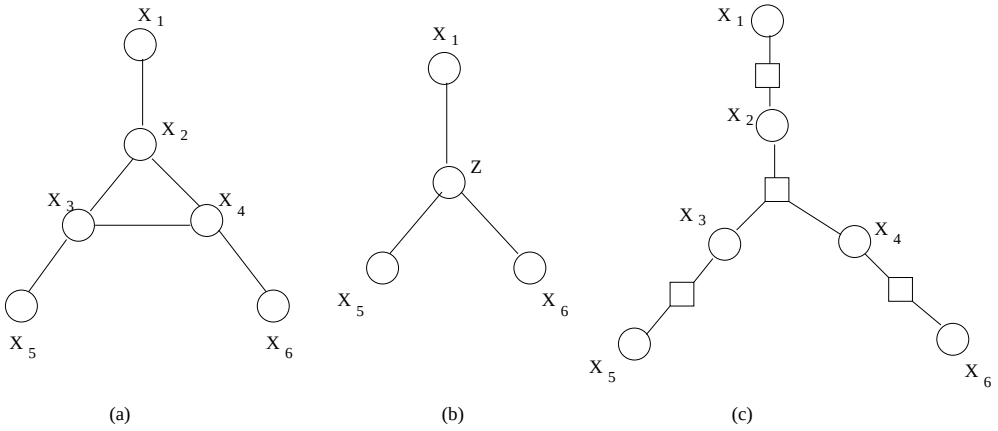


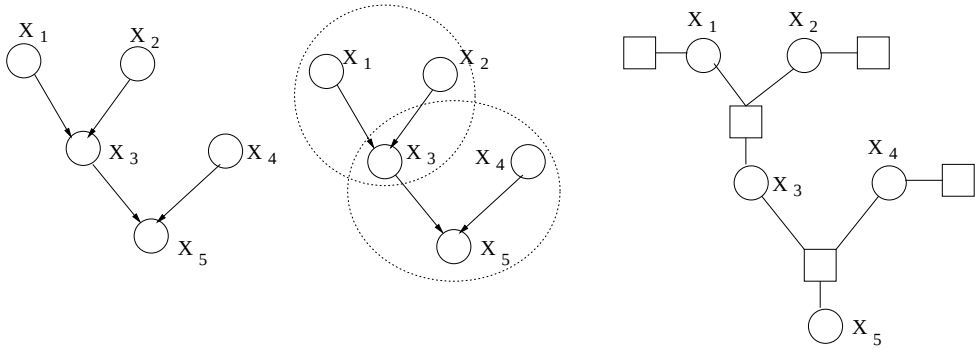
Figure 5: (a) A simple tree-like graph (b) Reducing the graph to a tree using a super-variable (c) Reducing to a tree using a factor graph representation

Some graphs can be simplified into tree structures by introducing super variables. Figure 5 gives an example of a graph where the dependencies between a clique of 3 nodes X_2, X_3, X_4 can be replaced by a single *super-variable* Z. The range of Z is the Cartesian product of the ranges of the three variables X_2, X_3, X_4. By creating new potentials $\Psi(x_1, z)$, $\Psi(x_5, z)$, $\Psi(x_6, z)$ and $\Psi(z)$ we can mimic the factorization of the probability distribution of the original graph.

Such a graph also has a factor-graph representation which is indicated in Figure 5. Since this factor graph is a factor tree, the sum-product algorithm can also be applied to this graph.

Super-variables vs factor-graphs: By introducing super-variables, we need to calculate new potential functions for these variables. But in a factor graph representation, we need no additional potentials to be calculated. Since both these methods produce tree structures, it will be better to use factor graph representations since it reduces computational overhead.

A directed graphical model worth mentioning is a *poly-tree*. A poly-tree is a directed graph such that the undirected version of the graph has no cycles. In a poly-tree, we can create super-variables for a collection of node and its parents. After this translation, we could convert the graph into an undirected tree. However, as mentioned earlier, it would be more advantageous to convert a poly-tree into a factor-graph rather than



(a) (b) (c)

Figure 6: (a) A simple poly-tree (b) Super-variable construction for poly-trees (c) Reducing to a tree using a factor graph representation

introducing new variables. Figure 6 illustrates a poly-tree and the corresponding factor graph representation. The figure also shows how super-variables are constructed for poly-trees.