Probabilistic Machine Learning:
7. Probabilistic Graphical Models - part 1

Tomasz Kajdanowicz, Piotr Bielak, Maciej Falkiewicz, Kacper Kania, Piotr Zieliński

Department of Computational Intelligence
Wroclaw University of Science and Technology

The presentation has been inspired and in some parts totally based on

1. Pattern Recognition and Machine Learning, Christopher M. Bishop [1] Chapter 8. Graphical Models

Table of Contents

Pre-reading

There was nothing to read.

Table of Contents

Table of Contents

Chain rule – recap

By the *chain rule* of probability, we can always represent a *joint distribution* as follows, using any ordering of the variables:

$$P(x_{1:V}) = P(x_1)P(x_2|x_1)P(x_3|x_2,x_1)P(x_4|x_1,x_2,x_3)\dots P(x_V|x_{1:V-1}) \quad (1)$$

, where V is the number of variables, and where we have dropped the conditioning on the fixed parameters θ for brevity. The problem with this expression is that it becomes more and more complicated to represent the conditional distributions $P(x_t|x_{1:t-1})$ as t gets large.

Example I

Bishop p.361-362 [1]

Let us consider a situation when some variables are independent of each other and the join distribution factorizes into the following form

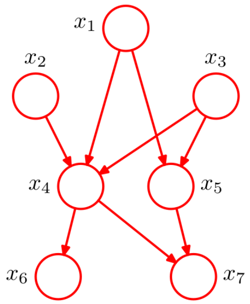
$$P(x_{1:7}) = P(x_1)P(x_2)P(x_3)P(x_4|x_1, x_2, x_3)P(x_5|x_1, x_3)P(x_6|x_4)P(x_7|x_4, x_5) \tag{2}$$

Example I

Bishop p.361-362 [1]

Let us consider a situation when some variables are independent of each other and the join distribution factorizes into the following form

$$P(x_{1:7}) = P(x_1)P(x_2)P(x_3)P(x_4|x_1, x_2, x_3)P(x_5|x_1, x_3)P(x_6|x_4)P(x_7|x_4, x_5) \tag{2}$$



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 362*

Directed Graphical Model

The joint distribution defined by a graph is given by the *product, over all of the nodes of the graph, of a conditional distribution for each node conditioned on the variables corresponding to the parents of that node* in the graph. Thus, for a graph with K nodes, the joint distribution is given by

$$\mathrm{P}(\boldsymbol{x}) = \prod_{k=1}^{K} \mathrm{P}(x_k|\mathrm{pa}_k), \tag{3}$$

where pa_k denotes the set of parents of x_k.

Directed Graphical Model

The joint distribution defined by a graph is given by the *product, over all of the nodes of the graph, of a conditional distribution for each node conditioned on the variables corresponding to the parents of that node* in the graph. Thus, for a graph with K nodes, the joint distribution is given by

$$\mathrm{P}(\boldsymbol{x}) = \prod_{k=1}^{K} \mathrm{P}(x_k | \mathrm{pa}_k), \tag{3}$$

where pa_k denotes the set of parents of x_k.
The directed graphs are subject to an important restriction – there must be *no directed cycles*, in other words there are no closed paths within the graph such that we can move from node to node along links following the direction of the arrows and end up back at the starting node. Such graphs are called *directed acyclic graphs*, or *DAG*s.

Directed Graphical Model

The joint distribution defined by a graph is given by the *product, over all of the nodes of the graph, of a conditional distribution for each node conditioned on the variables corresponding to the parents of that node* in the graph. Thus, for a graph with K nodes, the joint distribution is given by

$$P(\mathbf{x}) = \prod_{k=1}^{K} P(x_k | \mathrm{pa}_k), \tag{3}$$

where pa_k denotes the set of parents of x_k.
The directed graphs are subject to an important restriction – there must be *no directed cycles*, in other words there are no closed paths within the graph such that we can move from node to node along links following the direction of the arrows and end up back at the starting node. Such graphs are called *directed acyclic graphs*, or *DAG*s.
This is equivalent to the statement that there exists an ordering of the nodes such that there are no links that go from any node to any lower numbered node.

Example – Bayesian Regression

Let us consider the Bayesian Regression model that consist of coefficients vector β and obervations vector $\boldsymbol{y} = (y_1, \ldots, y_N)^\top$ and the noise variance σ^2 which we all consider *random variables*.

Example – Bayesian Regression

Let us consider the Bayesian Regression model that consist of coefficients vector β and obervations vector $\boldsymbol{y} = (y_1, \dots, y_N)^\top$ and the noise variance σ^2 which we all consider *random variables*. The joint distribution can be written as

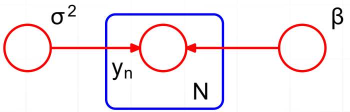
$$\mathrm{P}(\boldsymbol{y}, \boldsymbol{\beta}, \sigma^2) = \mathrm{P}(\sigma^2)\mathrm{P}(\boldsymbol{\beta}) \prod_{n=1}^{N} \mathrm{P}(y_n|\boldsymbol{\beta}, \sigma^2) \tag{4}$$

Example – Bayesian Regression

Let us consider the Bayesian Regression model that consist of coefficients vector β and obervations vector $\boldsymbol{y} = (y_1, \dots, y_N)^\top$ and the noise variance σ^2 which we all consider *random variables*. The joint distribution can be written as

$$\mathrm{P}(\boldsymbol{y}, \beta, \sigma^2) = \mathrm{P}(\sigma^2)\mathrm{P}(\beta) \prod_{n=1}^{N} \mathrm{P}(y_n | \beta, \sigma^2) \tag{4}$$

What results with the following plate representation:



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 363*

Example – Bayesian Regression

The model additionally has non-random variables which are the input data $\boldsymbol{x} = (x_1, \ldots, x_N)^\mathsf{T}$, σ^2's priors φ and β's priors α.

Example – Bayesian Regression

The model additionally has non-random variables which are the input data $\boldsymbol{x} = (x_1, \ldots, x_N)^\mathsf{T}$, σ^2's priors φ and β's priors α. The joint turns into

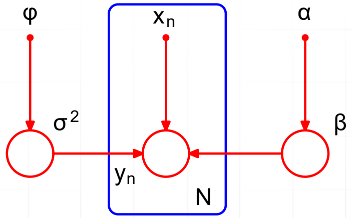
$$P(\boldsymbol{y}, \boldsymbol{\beta}, \sigma^2 | \boldsymbol{x}, \alpha, \varphi) = P(\sigma^2|\varphi)P(\boldsymbol{\beta}|\alpha)\prod_{n=1}^{N} P(y_n|x_n, \boldsymbol{\beta}, \sigma^2) \tag{5}$$

Example – Bayesian Regression

The model additionally has non-random variables which are the input data $\boldsymbol{x} = (x_1, \dots, x_N)^T$, σ^2's priors φ and β's priors α. The joint turns into

$$P(\boldsymbol{y}, \boldsymbol{\beta}, \sigma^2 | \boldsymbol{x}, \alpha, \varphi) = P(\sigma^2 | \varphi) P(\boldsymbol{\beta} | \alpha) \prod_{n=1}^{N} P(y_n | x_n, \boldsymbol{\beta}, \sigma^2) \tag{5}$$

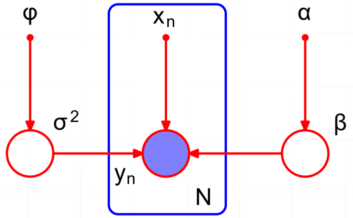
What results with the following plate representation:



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 364*

Example – Bayesian Regression

If we additionally mark the observed variables we get



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 364*

Example – Bayesian Regression

In general, model parameters such as β are of little direct interest in themselves, because our ultimate goal is to make predictions for new input values.

Example – Bayesian Regression

In general, model parameters such as β are of little direct interest in themselves, because our ultimate goal is to make predictions for new input values.
Suppose we are given a new input value $\hat{x}$ and we wish to find the corresponding probability distribution for $\hat{y}$ conditioned on the observed data.

Example – Bayesian Regression

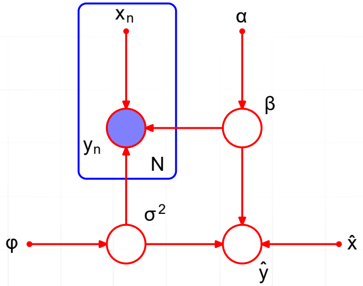
In general, model parameters such as β are of little direct interest in themselves, because our ultimate goal is to make predictions for new input values.
Suppose we are given a new input value $\hat{x}$ and we wish to find the corresponding probability distribution for $\hat{y}$ conditioned on the observed data. The corresponding joint distribution of all of the random variables in this model, conditioned on the deterministic parameters, is then given by

$$P(\hat{y}, \mathbf{y}, \beta, \sigma^2 | \hat{x}, \mathbf{x}, \alpha, \varphi) = P(\hat{y} | \hat{x}, \beta, \sigma^2) P(\sigma^2 | \varphi) P(\beta | \alpha) \prod_{n=1}^{N} P(y_n | x_n, \beta, \sigma^2) \tag{6}$$

Example – Bayesian Regression

$$P(\hat{y}, \mathbf{y}, \beta | \hat{x}, \mathbf{x}, \alpha, \varphi) = P(\hat{y} | \hat{x}, \beta, \sigma^2) P(\sigma^2 | \varphi) P(\beta | \alpha) \prod_{n=1}^{N} P(y_n | x_n, \beta, \sigma^2) \quad (7)$$

which is equivalent to



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 365*

Example – Mixture Model

Do you remember the simplest Latent Variable Model – Mixture Model?

Example – Mixture Model

Do you remember the simplest Latent Variable Model – Mixture Model?

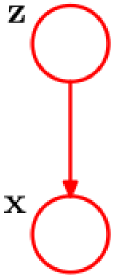
$$\mathrm{P}(\boldsymbol{x}, \boldsymbol{z}) = \mathrm{P}(\boldsymbol{z})\mathrm{P}(\boldsymbol{x}|\boldsymbol{z}) = \prod_{k=1}^{K} \pi_k^{z_k} \prod_{k=1}^{K} \mathrm{P}(x_i|\boldsymbol{\theta}_k)^{z_k} \tag{8}$$

Example – Mixture Model

Do you remember the simplest Latent Variable Model – Mixture Model?

$$\mathrm{P}(\boldsymbol{x}, \boldsymbol{z}) = \mathrm{P}(\boldsymbol{z})\mathrm{P}(\boldsymbol{x}|\boldsymbol{z}) = \prod_{k=1}^{K} \pi_k^{z_k} \prod_{k=1}^{K} \mathrm{P}(x_i|\boldsymbol{\theta}_k)^{z_k} \tag{8}$$

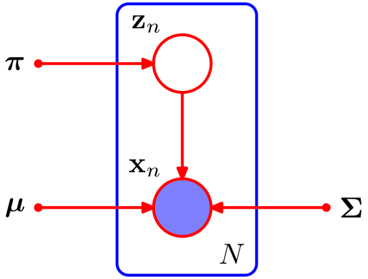
This is how it looks like as directed graphical model:



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 431*

Example – Mixture Model

And here with all the non-random parameters:



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 433*

Plate notation – summary

1. Are there any other drawing convetions? Yes, you can read about them at Wikipedia

Plate notation – summary

1. Are there any other drawing convetions? Yes, you can read about them at Wikipedia
2. How to draw PGMs in a convenient way?

Plate notation – summary

1. Are there any other drawing convetions? Yes, you can read about them at Wikipedia
2. How to draw PGMs in a convenient way?
    1. tikz-bayesnet package in LaTeX

Plate notation – summary

1. Are there any other drawing convetions? Yes, you can read about them at Wikipedia
2. How to draw PGMs in a convenient way?
   1. tikz-bayesnet package in LaTeX
   2. LaTeXDraw wich is a general purpose graphical drawing editor for LaTeX

Plate notation – summary

1. Are there any other drawing convetions? Yes, you can read about them at Wikipedia
2. How to draw PGMs in a convenient way?
    1. tikz-bayesnet package in LaTeX
    2. LaTeXDraw wich is a general purpose graphical drawing editor for LaTeX
    3. Daft package in Python

Plate notation – summary

1. Are there any other drawing convetions? Yes, you can read about them at Wikipedia
2. How to draw PGMs in a convenient way?
    1. tikz-bayesnet package in LaTeX
    2. LaTeXDraw wich is a general purpose graphical drawing editor for LaTeX
    3. Daft package in Python
    4. Dia which is a general purpose open source diagram creation program

Plate notation – summary

1. Are there any other drawing convetions? Yes, you can read about them at Wikipedia
2. How to draw PGMs in a convenient way?
    1. tikz-bayesnet package in LaTeX
    2. LaTeXDraw wich is a general purpose graphical drawing editor for LaTeX
    3. Daft package in Python
    4. Dia which is a general purpose open source diagram creation program
    5. Online tools such as Lucidchart, draw.io, Google Docs Draw, etc.

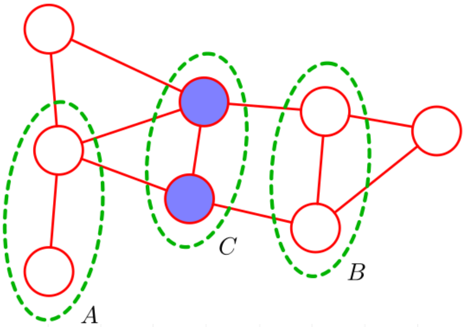
Table of Contents

Markov Random Fields

A *Markov random field*, also known as a *Markov network* or an *undirected graphical model*, has a set of nodes each of which corresponds to a variable or group of variables, as well as a set of links each of which connects a pair of nodes. The *links are undirected*, that is they do not carry arrows.

Markov Random Fields

A *Markov random field*, also known as a *Markov network* or an *undirected graphical model*, has a set of nodes each of which corresponds to a variable or group of variables, as well as a set of links each of which connects a pair of nodes. The *links are undirected*, that is they do not carry arrows.



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 384*

Markov Random Fields

Conditional independence

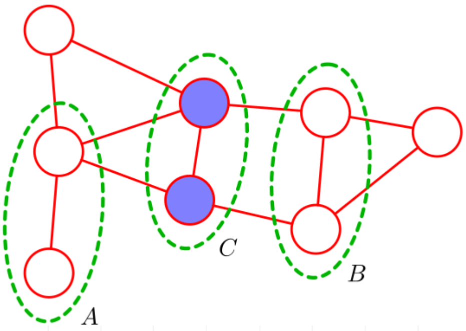
But how about conditional independence? How to handle the “problematic” head-to-head relation in D-separation?

Markov Random Fields

Conditional independence

But how about conditional independence? How to handle the "problematic" head-to-head relation in D-separation?

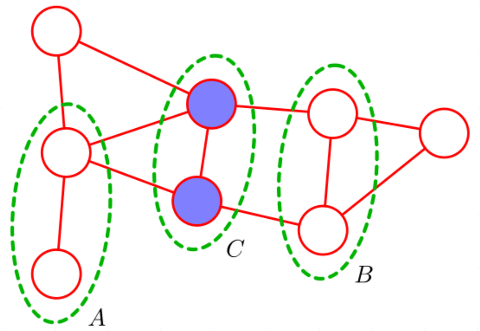
It turns out the situation here is easier. Consider the following undirected model:



Credit: *Pattern Recognition and Machine Learning*, *Christopher M. Bishop p. 384*

Markov Random Fields

Conditional independence



Credit: *Pattern Recognition and Machine Learning*, *Christopher M. Bishop p. 384*

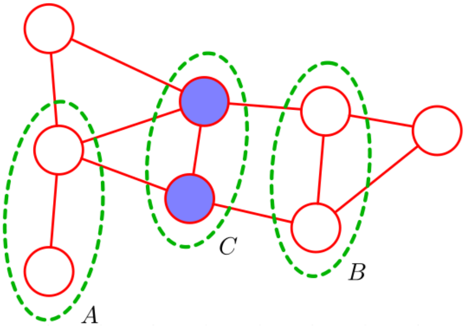
Three sets of nodes are identified – A, B and C. Mind that as a special case, they could consist of single elements. We want to test if

$$A \perp\!\!\!\perp B | C \tag{9}$$

holds.

Markov Random Fields

Conditional independence

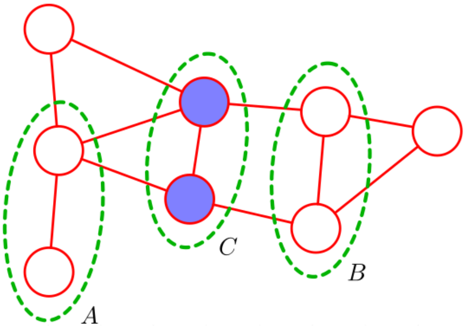


Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 384*

To test it we consider *all possible paths* that connect nodes in set *A* to nodes in set *B*. If all such paths *pass through one or more nodes in set* C, then all such paths are *blocked* and so the conditional independence property holds.

Markov Random Fields

Conditional independence



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 384*

An alternative way to view the conditional independence test is to imagine *removing all nodes in set C* from the graph together with any links that connect to those nodes. We then ask *if there exists a path that connects any node in A to any node in B*. If there are no such paths, then the conditional independence property must hold.

Markov blanket – recap

Directed models

Consider a joint distribution $\mathrm{P}(x_1, \ldots, x_D)$ of a model consisting of D nodes, and consider the conditional distribution of a particular node x_i conditioned on all of the remaining variables $\boldsymbol{x}_{j \neq i}$.

Markov blanket – recap

Directed models

Consider a joint distribution $\mathrm{P}(x_1, \ldots, x_D)$ of a model consisting of D nodes, and consider the conditional distribution of a particular node x_i conditioned on all of the remaining variables $\boldsymbol{x}_{j \neq i}$.

$$\mathrm{P}(x_i | \boldsymbol{x}_{j \neq i}) = \frac{\mathrm{P}(x_1, \ldots, x_D)}{\int \mathrm{P}(x_1, \ldots, x_D) \mathrm{d}x_i} = \frac{\prod_k \mathrm{P}(x_k | \mathrm{pa}_k)}{\int \prod_k \mathrm{P}(x_k | \mathrm{pa}_k) \mathrm{d}x_i} \tag{10}$$

Eq. 3 was applied here.

Markov blanket – recap

Directed models

Consider a joint distribution $\mathrm{P}(x_1, \ldots, x_D)$ of a model consisting of D nodes, and consider the conditional distribution of a particular node x_i conditioned on all of the remaining variables $\boldsymbol{x}_{j \neq i}$.

$$\mathrm{P}(x_i | \boldsymbol{x}_{j \neq i}) = \frac{\mathrm{P}(x_1, \ldots, x_D)}{\int \mathrm{P}(x_1, \ldots, x_D) \mathrm{d}x_i} = \frac{\prod_k \mathrm{P}(x_k | \mathrm{pa}_k)}{\int \prod_k \mathrm{P}(x_k | \mathrm{pa}_k) \mathrm{d}x_i} \tag{10}$$

Eq. 3 was applied here. We now observe that any factor $\mathrm{P}(x_k | \mathrm{pa}_k)$ in the denominator that does not have any functional dependence on x_i can be taken outside the integral over x_i, and will therefore cancel between numerator and denominator.

Markov blanket – recap

Directed models

Consider a joint distribution $P(x_1, \ldots, x_D)$ of a model consisting of D nodes, and consider the conditional distribution of a particular node x_i conditioned on all of the remaining variables $\boldsymbol{x}_{j \neq i}$.

$$P(x_i | \boldsymbol{x}_{j \neq i}) = \frac{P(x_1, \ldots, x_D)}{\int P(x_1, \ldots, x_D) \mathrm{d}x_i} = \frac{\prod_k P(x_k | \mathrm{pa}_k)}{\int \prod_k P(x_k | \mathrm{pa}_k) \mathrm{d}x_i} \quad (10)$$

Eq. 3 was applied here. We now observe that any factor $P(x_k | \mathrm{pa}_k)$ in the denominator that does not have any functional dependence on x_i can be taken outside the integral over x_i, and will therefore cancel between numerator and denominator.
The only factors that remain will be the conditional distribution $P(x_i | \mathrm{pa}_i)$ for node x_i itself, together with the conditional distributions for any nodes x_k such that node x_i is in the conditioning set of $P(x_k | \mathrm{pa}_k)$.

Markov blanket – recap

Directed models

Consider a joint distribution $\mathrm{P}(x_1, \ldots, x_D)$ of a model consisting of D nodes, and consider the conditional distribution of a particular node x_i conditioned on all of the remaining variables $\boldsymbol{x}_{j\neq i}$.

$$\mathrm{P}(x_i|\boldsymbol{x}_{j\neq i}) = \frac{\mathrm{P}(x_1, \ldots, x_D)}{\int \mathrm{P}(x_1, \ldots, x_D)\mathrm{d}x_i} = \frac{\prod_k \mathrm{P}(x_k|\mathrm{pa}_k)}{\int \prod_k \mathrm{P}(x_k|\mathrm{pa}_k)\mathrm{d}x_i} \tag{10}$$

Eq. 3 was applied here. We now observe that any factor $\mathrm{P}(x_k|\mathrm{pa}_k)$ in the denominator that does not have any functional dependence on x_i can be taken outside the integral over x_i, and will therefore cancel between numerator and denominator.
The only factors that remain will be the conditional distribution $\mathrm{P}(x_i|\mathrm{pa}_i)$ for node x_i itself, together with the conditional distributions for any nodes x_k such that node x_i is in the conditioning set of $\mathrm{P}(x_k|\mathrm{pa}_k)$.
The conditional $\mathrm{P}(x_i|\mathrm{pa}_i)$ will depend on the parents of node x_i.

Markov blanket – recap

Directed models

Consider a joint distribution $\mathrm{P}(x_1, \ldots, x_D)$ of a model consisting of D nodes, and consider the conditional distribution of a particular node x_i conditioned on all of the remaining variables $\boldsymbol{x}_{j \neq i}$.

$$\mathrm{P}(x_i|\boldsymbol{x}_{j\neq i}) = \frac{\mathrm{P}(x_1, \ldots, x_D)}{\int \mathrm{P}(x_1, \ldots, x_D)\mathrm{d}x_i} = \frac{\prod_k \mathrm{P}(x_k|\mathrm{pa}_k)}{\int \prod_k \mathrm{P}(x_k|\mathrm{pa}_k)\mathrm{d}x_i} \tag{10}$$

Eq. 3 was applied here. We now observe that any factor $\mathrm{P}(x_k|\mathrm{pa}_k)$ in the denominator that does not have any functional dependence on x_i can be taken outside the integral over x_i, and will therefore cancel between numerator and denominator.

The only factors that remain will be the conditional distribution $\mathrm{P}(x_i|\mathrm{pa}_i)$ for node x_i itself, together with the conditional distributions for any nodes x_k such that node x_i is in the conditioning set of $\mathrm{P}(x_k|\mathrm{pa}_k)$.

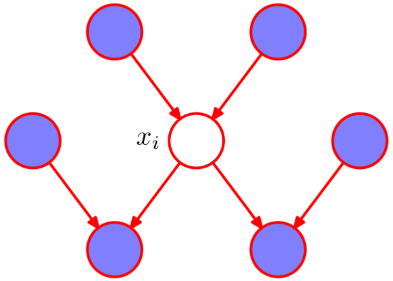
The conditional $\mathrm{P}(x_i|\mathrm{pa}_i)$ will depend on the parents of node x_i.

The conditionals $\mathrm{P}(x_k|\mathrm{pa}_k)$ will depend on the children of x_i as well as on the *co-parents*, in other words variables corresponding to parents of node x_k other than node x_i due to head-to-head.

Markov blanket – recap

Directed models

The set of nodes comprising the parents, the children and the co-parents is called the *Markov blanket*.

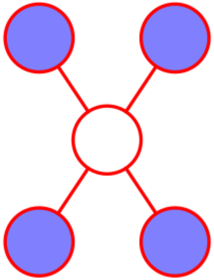


Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 383*

Markov blanket

Undirected models

For an undirected graph, the Markov blanket of a node x_i consists of the set of neighbouring nodes.



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 385*

Joint distribution factorization

We now seek a factorization rule for undirected graphs that will correspond to the conditional independence test. Again, this will involve expressing the joint distribution $P(\boldsymbol{x})$ as a product of functions defined over sets of variables that are local to the graph.

Joint distribution factorization

We now seek a factorization rule for undirected graphs that will correspond to the conditional independence test. Again, this will involve expressing the joint distribution $P(\boldsymbol{x})$ as a product of functions defined over sets of variables that are local to the graph. Consider two nodes x_i and x_j that are not connected by a link. Then these variables must be conditionaly independent given all other nodes in the graph.

$$P(x_i, x_j|\boldsymbol{x}_{\setminus\{i,j\}}) = P(x_i|\boldsymbol{x}_{\setminus\{i,j\}})P(x_j|\boldsymbol{x}_{\setminus\{i,j\}}) \tag{11}$$

where $\boldsymbol{x}_{\setminus\{i,j\}}$ denotes denotes the set $\boldsymbol{x}$ of all variables with x_i and x_j removed.

Joint distribution factorization

We now seek a factorization rule for undirected graphs that will correspond to the conditional independence test. Again, this will involve expressing the joint distribution $P(\boldsymbol{x})$ as a product of functions defined over sets of variables that are local to the graph. Consider two nodes x_i and x_j that are not connected by a link. Then these variables must be conditionaly independent given all other nodes in the graph.

$$P(x_i, x_j|\boldsymbol{x}_{\setminus\{i,j\}}) = P(x_i|\boldsymbol{x}_{\setminus\{i,j\}})P(x_j|\boldsymbol{x}_{\setminus\{i,j\}}) \tag{11}$$

where $\boldsymbol{x}_{\setminus\{i,j\}}$ denotes denotes the set $\boldsymbol{x}$ of all variables with x_i and x_j removed.
The factorization of the joint distribution must therefore be such that x_i and x_j do not appear in the same *factor*.

Joint distribution factorization

We now seek a factorization rule for undirected graphs that will correspond to the conditional independence test. Again, this will involve expressing the joint distribution $P(\boldsymbol{x})$ as a product of functions defined over sets of variables that are local to the graph. Consider two nodes x_i and x_j that are not connected by a link. Then these variables must be conditionaly independent given all other nodes in the graph.

$$P(x_i, x_j|\boldsymbol{x}_{\setminus\{i,j\}}) = P(x_i|\boldsymbol{x}_{\setminus\{i,j\}})P(x_j|\boldsymbol{x}_{\setminus\{i,j\}}) \tag{11}$$

where $\boldsymbol{x}_{\setminus\{i,j\}}$ denotes denotes the set $\boldsymbol{x}$ of all variables with x_i and x_j removed.
The factorization of the joint distribution must therefore be such that x_i and x_j do not appear in the same *factor*.
We define the factors in the decomposition of the joint distribution to be functions of the variables in the *cliques*. In fact, we can consider functions of the *maximal cliques*, without loss of generality, because other cliques must be subsets of maximal cliques.

Joint distribution factorization

Let us denote clique by C and the set of variables in the clique by $\mathbf{x}_C$.

Joint distribution factorization

Let us denote clique by C and the set of variables in the clique by $\boldsymbol{x}_C$.
Then the join distribution is written as a product of *potential functions* $\psi_C(\boldsymbol{x}_C)$ over the maximal cliques in the graph

$$P(\boldsymbol{x}) = \frac{1}{Z}\prod_C \psi_C(\boldsymbol{x}_C), \tag{12}$$

where

$$Z = \sum_{\boldsymbol{x}}\prod_C \psi_C(\boldsymbol{x}_C) \tag{13}$$

acts as a normalizing factor. It is called the *partition function*.

Joint distribution factorization

Let us denote clique by C and the set of variables in the clique by $\mathbf{x}_C$.
Then the join distribution is written as a product of *potential functions* $\psi_C(\mathbf{x}_C)$ over the maximal cliques in the graph

$$P(\mathbf{x}) = \frac{1}{Z} \prod_C \psi_C(\mathbf{x}_C), \qquad (12)$$

where

$$Z = \sum_{\mathbf{x}} \prod_C \psi_C(\mathbf{x}_C) \qquad (13)$$

acts as a normalizing factor. It is called the *partition function*.
In eq. 13 we have assumed that $\mathbf{x}$ comprises discrete variables, but the framework is equally applicable to continuous variables, or a combination of the two, in which the summation is replaced by the appropriate combination of summation and integration.

Potential functions – remarks

1. Note that we do not restrict the choice of potential functions to those that have a specific probabilistic interpretation as marginal or conditional distributions.

Potential functions – remarks

1. Note that we do not restrict the choice of potential functions to those that have a specific probabilistic interpretation as marginal or conditional distributions.
2. One consequence of the generality of the potential functions $\psi_C(\boldsymbol{x}_C)$ is that their product will in general *not be correctly normalized*.

Potential functions – remarks

1. Note that we do not restrict the choice of potential functions to those that have a specific probabilistic interpretation as marginal or conditional distributions.
2. One consequence of the generality of the potential functions $\psi_C(\mathbf{x}_C)$ is that their product will in general *not be correctly normalized*. Recall that for directed graphs, the joint distribution was *automatically normalized* as a consequence of the normalization of each of the conditional distributions in the factorization.

Potential functions – remarks

1. Note that we do not restrict the choice of potential functions to those that have a specific probabilistic interpretation as marginal or conditional distributions.
2. One consequence of the generality of the potential functions $\psi_C(\mathbf{x}_C)$ is that their product will in general *not be correctly normalized*. Recall that for directed graphs, the joint distribution was *automatically normalized* as a consequence of the normalization of each of the conditional distributions in the factorization. The presence of this normalization constant is one of the major *limitations of undirected graphs*. If we have a model with M discrete nodes each having K states, then the evaluation of the *normalization term involves summing over K^M states* and so (in the worst case) is exponential in the size of the model.

Potential functions – remarks

1. Note that we do not restrict the choice of potential functions to those that have a specific probabilistic interpretation as marginal or conditional distributions.
2. One consequence of the generality of the potential functions $\psi_C(\mathbf{x}_C)$ is that their product will in general *not be correctly normalized*. Recall that for directed graphs, the joint distribution was *automatically normalized* as a consequence of the normalization of each of the conditional distributions in the factorization. The presence of this normalization constant is one of the major *limitations of undirected graphs*. If we have a model with M discrete nodes each having K states, then the evaluation of the *normalization term involves summing over K^M states* and so (in the worst case) is exponential in the size of the model.
3. For evaluation of local conditional distributions, the partition function is not needed because a conditional is the *ratio of two marginals*, and the partition function cancels between numerator and denominator when evaluating this ratio.

Potential functions – remarks

1. Note that we do not restrict the choice of potential functions to those that have a specific probabilistic interpretation as marginal or conditional distributions.
2. One consequence of the generality of the potential functions $\psi_C(\mathbf{x}_C)$ is that their product will in general *not be correctly normalized*. Recall that for directed graphs, the joint distribution was *automatically normalized* as a consequence of the normalization of each of the conditional distributions in the factorization. The presence of this normalization constant is one of the major *limitations of undirected graphs*. If we have a model with M discrete nodes each having K states, then the evaluation of the *normalization term involves summing over K^M states* and so (in the worst case) is exponential in the size of the model.
3. For evaluation of local conditional distributions, the partition function is not needed because a conditional is the *ratio of two marginals*, and the partition function cancels between numerator and denominator when evaluating this ratio.
4. For evaluating local marginal probabilities we can work with the unnormalized joint distribution and then *normalize the marginals explicitly at the end*. Provided the marginals only involves a small number of variables, the evaluation of their normalization coefficient will be feasible.

Joint distribution factorization – formally

The construction shown above gives intution, but lacks formal structure.

Joint distribution factorization – formally

The construction shown above gives intution, but lacks formal structure.
We need to restrict to potential functions that are strictly positive, that isbn

$$\forall C : \ \psi_C(\boldsymbol{x}_C) > 0 \tag{14}$$

Joint distribution factorization – formally

The construction shown above gives intution, but lacks formal structure.
We need to restrict to potential functions that are strictly positive, that isbn

$$\forall C : \ \psi_C(\mathbf{x}_C) > 0 \tag{14}$$

In such a case it is convenient to express then as exponentials, so that

$$\psi_C(\mathbf{x}_C) = \exp(-E(\mathbf{x}_C)), \tag{15}$$

where $E(\mathbf{x}_C)$ is called *energy function* and the exponential representation is called the *Boltzman distribution*.

Joint distribution factorization – formally

The construction shown above gives intution, but lacks formal structure.
We need to restrict to potential functions that are strictly positive, that isbn

$$\forall C : \ \psi_C(\mathbf{x}_C) > 0 \tag{14}$$

In such a case it is convenient to express then as exponentials, so that

$$\psi_C(\mathbf{x}_C) = \exp(-E(\mathbf{x}_C)), \tag{15}$$

where $E(\mathbf{x}_C)$ is called *energy function* and the exponential representation is called the *Boltzman distribution*.
The joint distribution is defined as the product of potentials, and so the *total energy* is obtained by adding the energies of each of the maximal cliques.

Joint distribution factorization – formally

The construction shown above gives intution, but lacks formal structure.
We need to restrict to potential functions that are strictly positive, that isbn

$$\forall C : \; \psi_C(\mathbf{x}_C) > 0 \tag{14}$$

In such a case it is convenient to express then as exponentials, so that

$$\psi_C(\mathbf{x}_C) = \exp(-E(\mathbf{x}_C)), \tag{15}$$

where $E(\mathbf{x}_C)$ is called *energy function* and the exponential representation is called the *Boltzman distribution*.
The joint distribution is defined as the product of potentials, and so the *total energy* is obtained by adding the energies of each of the maximal cliques.

The *Hammersley-Clifford theorem* [2] states that such formulation of the factorization of the joint probability distribution is correct.

Relation of directed and undirected models

Graphical models should be seen from the perspective of *conditional independence*, that is *the lack of links* rather then the links itself.

Relation of directed and undirected models

Graphical models should be seen from the perspective of *conditional independence*, that is *the lack of links* rather then the links itself.

D map A graph is said to be a *D map* (for *dependency map*) of a distribution if every conditional independence statement satisfied by the distribution is reflected in the graph. Thus a completely disconnected graph (no links) will be a trivial D map for any distribution.

I map If every conditional independence statement implied by a graph is satisfied by a specific distribution, then the graph is said to be an *I map* (for *independence map*) of that distribution. Clearly a fully connected graph will be a trivial I map for any distribution.

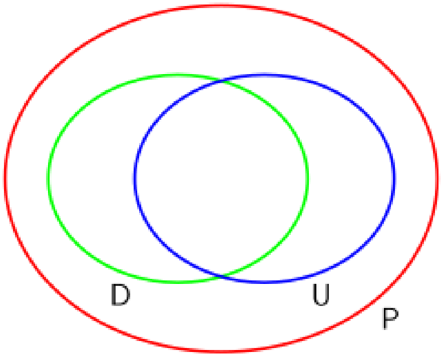
Relation of directed and undirected models

Graphical models should be seen from the perspective of *conditional independence*, that is *the lack of links* rather then the links itself.

D map	A graph is said to be a *D map* (for *dependency map*) of a distribution if every conditional independence statement satisfied by the distribution is reflected in the graph. Thus a completely disconnected graph (no links) will be a trivial D map for any distribution.
I map	If every conditional independence statement implied by a graph is satisfied by a specific distribution, then the graph is said to be an *I map* (for *independence map*) of that distribution. Clearly a fully connected graph will be a trivial I map for any distribution.

If it is the case that every conditional independence property of the distribution is reflected in the graph, and vice versa, then the graph is said to be a *perfect map* for that distribution. A perfect map is therefore both an I map and a D map.

Relation of directed and undirected models



Credit: *Pattern Recognition and Machine Learning*, *Christopher M. Bishop p. 392*

Figure: Venn diagram illustrating the set of all distributions *P* over a given set of variables, together with the set of distributions *D* that can be represented as a *perfect map* using a directed graph, and the set *U* that can be represented as a perfect map using an undirected graph.

Relation of directed and undirected models

These means that *some* directed models can be translated into undirected ones *and vice versa*.

Relation of directed and undirected models

These means that *some* directed models can be translated into undirected ones *and vice versa*.

If you are interested in this subject please check [1] Chapter 8.3.4 Relation to directed graphs p.390-393.

Relation of directed and undirected models

These means that *some* directed models can be translated into undirected ones *and vice versa*.
If you are interested in this subject please check [1] Chapter 8.3.4 Relation to directed graphs p.390-393.

The graphical framework can be extended in a consistent way to graphs that include *both directed and undirected links*. These are called *chain graphs* [3], and contain the directed and undirected graphs considered so far as special cases. Although such graphs can represent a broader class of distributions than either directed or undirected alone, *there remain distributions for which even a chain graph cannot provide a perfect map*. Chain graphs are not discussed further in this course.

Table of Contents

Inference in Graphical Models

Consider a situation when for a given graphical model we have *observed data* for part of the nodes (possibly all).

Inference in Graphical Models

Consider a situation when for a given graphical model we have *observed data* for part of the nodes (possibly all).
We want to compute the *posterior distributions* of one or more subsets of other nodes.

Inference in Graphical Models

Consider a situation when for a given graphical model we have *observed data* for part of the nodes (possibly all).
We want to compute the *posterior distributions* of one or more subsets of other nodes.

We can exploit the graphical structure both to find efficient algorithms for inference, and to make the structure of those algorithms transparent.

Inference in Graphical Models

Consider a situation when for a given graphical model we have *observed data* for part of the nodes (possibly all).
We want to compute the *posterior distributions* of one or more subsets of other nodes.

We can exploit the graphical structure both to find efficient algorithms for inference, and to make the structure of those algorithms transparent.
Specifically, we shall see that many algorithms can be expressed in terms of the propagation of *local messages* around the graph.

Inference in Graphical Models

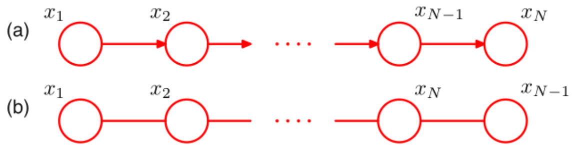
Consider a situation when for a given graphical model we have *observed data* for part of the nodes (possibly all).
We want to compute the *posterior distributions* of one or more subsets of other nodes.

We can exploit the graphical structure both to find efficient algorithms for inference, and to make the structure of those algorithms transparent.
Specifically, we shall see that many algorithms can be expressed in terms of the propagation of *local messages* around the graph.

In this part of the lecture we will focus primarily on techniques for *exact inference*, and later we shall consider a number of *approximate inference* algorithms.

Inference on a chain

Consider the undirected graph in *Figure (b)* consisting of N nodes. The directed chain can be transformed into an equivalent undirected chain. Because the directed graph does not have any nodes with more than one parent, this does not require the addition of any extra links, and the directed and undirected versions of this graph express exactly the same set of conditional independence statements.

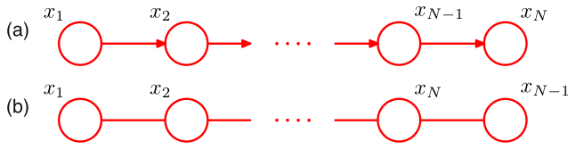


Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 390*

Figure: a) Example of directed graph. b) The equivalent undirected graph.

Inference on a chain

Consider the undirected graph in *Figure (b)* consisting of N nodes. The directed chain can be transformed into an equivalent undirected chain. Because the directed graph does not have any nodes with more than one parent, this does not require the addition of any extra links, and the directed and undirected versions of this graph express exactly the same set of conditional independence statements.



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 390*

Figure: a) Example of directed graph. b) The equivalent undirected graph.

The joint distribution takes the form

$$P(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\cdots\psi_{N-1,N}(x_{N-1}, x_N) \tag{16}$$

Inference on a chain

$$P(\boldsymbol{x}) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\cdots\psi_{N-1,N}(x_{N-1}, x_N)$$

Let us restrict to a specific case where the N nodes represent discrete variables each having K states, in which case each potential function $\psi_{n-1,n}(x_{n-1}, x_n)$ comprises an $K \times K$ table, and so the joint distribution has $(N-1)K^2$ parameters.

Inference on a chain

$$P(\boldsymbol{x}) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\cdots\psi_{N-1,N}(x_{N-1}, x_N)$$

Let us restrict to a specific case where the N nodes represent discrete variables each having K states, in which case each potential function $\psi_{n-1,n}(x_{n-1}, x_n)$ comprises an $K \times K$ table, and so the joint distribution has $(N-1)K^2$ parameters.

We are interested in finding the marginal distribution $P(x_n)$ for node x_n somewhere in the chain.

Inference on a chain

$$P(\boldsymbol{x}) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\cdots\psi_{N-1,N}(x_{N-1}, x_N)$$

Let us restrict to a specific case where the N nodes represent discrete variables each having K states, in which case each potential function $\psi_{n-1,n}(x_{n-1}, x_n)$ comprises an $K \times K$ table, and so the joint distribution has $(N-1)K^2$ parameters.

We are interested in finding the marginal distribution $P(x_n)$ for node x_n somewhere in the chain.
For a moment imagine there are no observed nodes and simply by definition write down the marginal by integrating out all variables except for x_n.

Inference on a chain

$$P(\boldsymbol{x}) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\cdots\psi_{N-1,N}(x_{N-1}, x_N)$$

Let us restrict to a specific case where the N nodes represent discrete variables each having K states, in which case each potential function $\psi_{n-1,n}(x_{n-1}, x_n)$ comprises an $K \times K$ table, and so the joint distribution has $(N-1)K^2$ parameters.

We are interested in finding the marginal distribution $P(x_n)$ for node x_n somewhere in the chain.
For a moment imagine there are no observed nodes and simply by definition write down the marginal by integrating out all variables except for x_n.

$$P(x_n) = \sum_{x_1}\cdots\sum_{n-1}\sum_{n+1}\cdots\sum_{x_N} P(x) \tag{17}$$

Inference on a chain

$$P(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\cdots\psi_{N-1,N}(x_{N-1}, x_N)$$

Let us restrict to a specific case where the N nodes represent discrete variables each having K states, in which case each potential function $\psi_{n-1,n}(x_{n-1}, x_n)$ comprises an $K \times K$ table, and so the joint distribution has $(N-1)K^2$ parameters.

We are interested in finding the marginal distribution $P(x_n)$ for node x_n somewhere in the chain.
For a moment imagine there are no observed nodes and simply by definition write down the marginal by integrating out all variables except for x_n.

$$P(x_n) = \sum_{x_1}\cdots\sum_{n-1}\sum_{n+1}\cdots\sum_{x_N} P(x) \tag{17}$$

Now recall the chain factorization in eq. 16 from the previous slide and put it into eq. 17.

Inference on a chain

$$P(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\cdots\psi_{N-1,N}(x_{N-1}, x_N)$$

Let us restrict to a specific case where the N nodes represent discrete variables each having K states, in which case each potential function $\psi_{n-1,n}(x_{n-1}, x_n)$ comprises an $K \times K$ table, and so the joint distribution has $(N-1)K^2$ parameters.

We are interested in finding the marginal distribution $P(x_n)$ for node x_n somewhere in the chain.
For a moment imagine there are no observed nodes and simply by definition write down the marginal by integrating out all variables except for x_n.

$$P(x_n) = \sum_{x_1}\cdots\sum_{n-1}\sum_{n+1}\cdots\sum_{x_N} P(x) \qquad (17)$$

Now recall the chain factorization in eq. 16 from the previous slide and put it into eq. 17.

Inference on a chain

$$P(x_n) = \sum_{x_1} \cdots \sum_{n-1} \sum_{n+1} \cdots \sum_{x_N} \frac{1}{Z} \psi_{1,2}(x_1, x_2) \psi_{2,3}(x_2, x_3) \cdots \psi_{N-1,N}(x_{N-1}, x_N) \qquad (18)$$

Inference on a chain

$$P(x_n) = \sum_{x_1} \cdots \sum_{n-1} \sum_{n+1} \cdots \sum_{x_N} \frac{1}{Z} \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3) \cdots \psi_{N-1,N}(x_{N-1}, x_N) \qquad (18)$$

In a naive implementation, we would first evaluate the joint distribution and then perform the summations explicitly. The joint distribution can be represented as a set of numbers, one for each possible value for **x**.

Inference on a chain

$$P(x_n) = \sum_{x_1} \cdots \sum_{n-1} \sum_{n+1} \cdots \sum_{x_N} \frac{1}{Z} \psi_{1,2}(x_1, x_2) \psi_{2,3}(x_2, x_3) \cdots \psi_{N-1,N}(x_{N-1}, x_N) \tag{18}$$

In a naive implementation, we would first evaluate the joint distribution and then perform the summations explicitly. The joint distribution can be represented as a set of numbers, one for each possible value for $\mathbf{x}$.
Because there are N variables each with K states, there are K^N values for $\boldsymbol{x}$ so evaluation and storage of the joint distribution, as well as marginalization to obtain $P(x_n)$, all involve storage and computation that scale exponentially with the length N of the chain.

Inference on a chain

$$P(x_n) = \sum_{x_1} \cdots \sum_{n-1} \sum_{n+1} \cdots \sum_{x_N} \frac{1}{Z} \psi_{1,2}(x_1, x_2) \psi_{2,3}(x_2, x_3) \cdots \psi_{N-1,N}(x_{N-1}, x_N) \tag{18}$$

In a naive implementation, we would first evaluate the joint distribution and then perform the summations explicitly. The joint distribution can be represented as a set of numbers, one for each possible value for **x**.
Because there are N variables each with K states, there are K^N values for $\boldsymbol{x}$ so evaluation and storage of the joint distribution, as well as marginalization to obtain $P(x_n)$, all involve storage and computation that scale exponentially with the length N of the chain.

But by by exploiting the factorization we can rearrange the order of the summations and the multiplications to allow the required marginal to be evaluated much more efficiently.

Inference on a chain

$$P(x_n) = \sum_{x_1} \cdots \sum_{n-1} \sum_{n+1} \cdots \sum_{x_N} \frac{1}{Z} \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3) \cdots \psi_{N-1,N}(x_{N-1}, x_N)$$

Consider the summation over x_N. The only factor that depends on x_N is $\psi_{N-1,N}(x_{N-1}, x_N)$.

Inference on a chain

$$P(x_n) = \sum_{x_1} \cdots \sum_{n-1} \sum_{n+1} \cdots \sum_{x_N} \frac{1}{Z} \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3) \cdots \psi_{N-1,N}(x_{N-1}, x_N)$$

Consider the summation over x_N. The only factor that depends on x_N is $\psi_{N-1,N}(x_{N-1}, x_N)$. We can perform the summation

$$\sum_{N} \psi_{N-1,N}(x_{N-1}, x_N) \tag{19}$$

at first to obtain a function of x_{N-1} which can be integrated out by a summation that additionally involves only $\psi_{N-2,N-1}(x_{N-2}, x_{N-1})$.

Inference on a chain

$$P(x_n) = \sum_{x_1} \cdots \sum_{n-1} \sum_{n+1} \cdots \sum_{x_N} \frac{1}{Z} \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3) \cdots \psi_{N-1,N}(x_{N-1}, x_N)$$

Consider the summation over x_N. The only factor that depends on x_N is $\psi_{N-1,N}(x_{N-1}, x_N)$.
We can perform the summation

$$\sum_{N} \psi_{N-1,N}(x_{N-1}, x_N) \tag{19}$$

at first to obtain a function of x_{N-1} which can be integrated out by a summation that additionally involves only $\psi_{N-2,N-1}(x_{N-2}, x_{N-1})$.
Similarly we can proceed from “the left side”.

Inference on a chain

Finally we end up with the following form:

$$p(x_n) = \frac{1}{Z} \underbrace{\left[\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \cdots \left[\sum_{x_2} \psi_{2,3}(x_2, x_3) \left[\sum_{x_1} \psi_{1,2}(x_1, x_2) \right] \right] \cdots \right]}_{\mu_\alpha(x_n)} \underbrace{\left[\sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \cdots \left[\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N) \right] \cdots \right]}_{\mu_\beta(x_n)} . \qquad (8.52)$$

Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 396*

Inference on a chain

Finally we end up with the following form:

$$p(x_n) = \frac{1}{Z}$$

$$\underbrace{\left[\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1}, x_n) \cdots \left[\sum_{x_2} \psi_{2,3}(x_2, x_3) \left[\sum_{x_1} \psi_{1,2}(x_1, x_2)\right]\right] \cdots \right]}_{\mu_\alpha(x_n)}$$

$$\underbrace{\left[\sum_{x_{n+1}} \psi_{n,n+1}(x_n, x_{n+1}) \cdots \left[\sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)\right] \cdots \right]}_{\mu_\beta(x_n)}. \qquad (8.52)$$

Credit: *Pattern Recognition and Machine Learning*, *Christopher M. Bishop p. 396*

Mind that if the graph had been fully connected, there would have been no conditional independence properties, and we would have been forced to work directly with the full joint distribution.

Inference on a chain

If you are interested in calculating computational cost of evaluating the required marginal using this re-ordered expression please check [1] Chapter 8.4.1 Inference on a chain p.396.

Inference on a chain

If you are interested in calculating computational cost of evaluating the required marginal using this re-ordered expression please check [1] Chapter 8.4.1 Inference on a chain p.396.

We can give an interpretation of this calculation in terms of *message passing* around the graph.

Inference on a chain

If you are interested in calculating computational cost of evaluating the required marginal using this re-ordered expression please check [1] Chapter 8.4.1 Inference on a chain p.396.

We can give an interpretation of this calculation in terms of *message passing* around the graph.
The marginal decomposes into

$$P(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n). \tag{20}$$

Inference on a chain

If you are interested in calculating computational cost of evaluating the required marginal using this re-ordered expression please check [1] Chapter 8.4.1 Inference on a chain p.396.

We can give an interpretation of this calculation in terms of *message passing* around the graph.
The marginal decomposes into

$$P(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n). \tag{20}$$

We shall interpret $\mu_\alpha(x_n)$ as a *message passed forwards* along the chain from node x_{n-1} to node x_n.

Inference on a chain

If you are interested in calculating computational cost of evaluating the required marginal using this re-ordered expression please check [1] Chapter 8.4.1 Inference on a chain p.396.

We can give an interpretation of this calculation in terms of *message passing* around the graph.
The marginal decomposes into

$$P(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n). \tag{20}$$

We shall interpret $\mu_\alpha(x_n)$ as a *message passed forwards* along the chain from node x_{n-1} to node x_n.
Similarly, $\mu_\beta(x_n)$ can be viewed as a *message passed backwards* along the chain from node x_{n+1} to node x_n.

Inference on a chain

If you are interested in calculating computational cost of evaluating the required marginal using this re-ordered expression please check [1] Chapter 8.4.1 Inference on a chain p.396.

We can give an interpretation of this calculation in terms of *message passing* around the graph.
The marginal decomposes into

$$P(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n). \tag{20}$$

We shall interpret $\mu_\alpha(x_n)$ as a *message passed forwards* along the chain from node x_{n-1} to node x_n.
Similarly, $\mu_\beta(x_n)$ can be viewed as a *message passed backwards* along the chain from node x_{n+1} to node x_n.

Inference on a chain

Each message consist of a set of K values, one for each choice of x_n and so the product of two messages should be interpreted as the point-wise multiplication of the elements of the two messages to give another set of K values.

Inference on a chain

Each message consist of a set of K values, one for each choice of x_n and so the product of two messages should be interpreted as the point-wise multiplication of the elements of the two messages to give another set of K values.

Both $\mu_\alpha(x_n)$ and $\mu_\beta(x_n)$ can be evaluated recursively starting from x_2 and x_N respectively.

Inference on a chain

Each message consist of a set of K values, one for each choice of x_n and so the product of two messages should be interpreted as the point-wise multiplication of the elements of the two messages to give another set of K values.

Both $\mu_\alpha(x_n)$ and $\mu_\beta(x_n)$ can be evaluated recursively starting from x_2 and x_N respectively.

The potential function Z can be easily evaluated by summing the right-hand side of eq. 20 over all states of x_n, an operation that requires only $\mathrm{O}(K)$ computation.

Inference on a chain

Are you interested in calculating *all marginals* and *joint distribution* of neighbouring nodes?

Inference on a chain

Are you interested in calculating *all marginals* and *joint distribution* of neighbouring nodes?

Check out [1] p.398!

Factor graphs

We now introduce a new construction called *factor graph*.

Factor graphs

We now introduce a new construction called *factor graph*.

Both directed and undirected graphs allow a global function of several variables to be expressed as a *product of factors over subsets of those variables*. Factor graphs make this decomposition explicit by introducing *and nodes for the factors* themselves in addition to the nodes representing the variables.

Factor graphs

We now introduce a new construction called *factor graph*.

Both directed and undirected graphs allow a global function of several variables to be expressed as a *product of factors over subsets of those variables*. Factor graphs make this decomposition explicit by introducing *and nodes for the factors* themselves in addition to the nodes representing the variables.
Let us write the joint distribution over a set of variables in the form of a product of factors

$$\mathrm{P}(\boldsymbol{x}) = \prod_{s} f_s(\boldsymbol{x}_s), \tag{21}$$

where $\boldsymbol{x}_s$ denoted a subset of variables.

Factor graphs

We now introduce a new construction called *factor graph*.

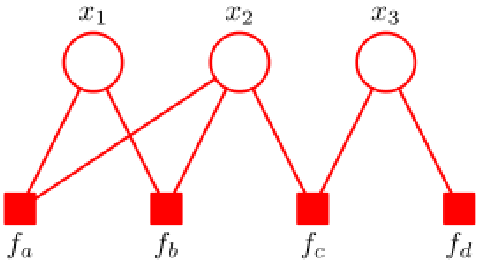
Both directed and undirected graphs allow a global function of several variables to be expressed as a *product of factors over subsets of those variables*. Factor graphs make this decomposition explicit by introducing *and nodes for the factors* themselves in addition to the nodes representing the variables.
Let us write the joint distribution over a set of variables in the form of a product of factors

$$\mathrm{P}(\boldsymbol{x}) = \prod_{s} f_s(\boldsymbol{x}_s), \tag{21}$$

where $\boldsymbol{x}_s$ denoted a subset of variables.
Directed graphs, represent special cases of eq. 21 in which the factors $f_s(x_s)$ are local conditional distributions. Similarly, undirected graphs, are a special case in which the factors are potential functions over the maximal cliques. The normalizing coefficient $1/Z$ can be viewed as a factor defined over the empty set of variables.

Factor graphs

In a factor graph, there is a node (depicted as usual by a circle) for every variable in the distribution, as was the case for directed and undirected graphs. There are also *additional nodes (depicted by small squares) for each factor $f_s(x_s)$* in the joint distribution. Finally, there are *undirected links connecting each factor node to all of the variables nodes on which that factor depends.*

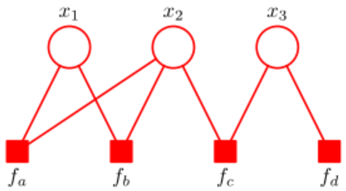


Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 400*

Figure: Factor graph corresponding to $P(\mathbf{x}) = f_a(x_1, x_2)f_b(x_1, x_2)f_c(x_2, x_3)f_d(x_3)$

Factor graphs

Example I



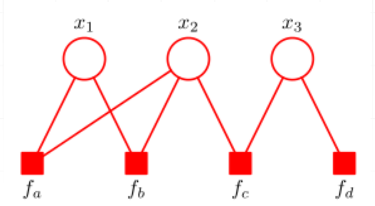
Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 400*

Figure: Factor graph corresponding to $P(\boldsymbol{x}) = f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$

Note that there are two factors $f_a(x_1, x_2)$ and $f_b(x_1, x_2)$ that are defined over the same set of variables.

Factor graphs

Example I



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 400*

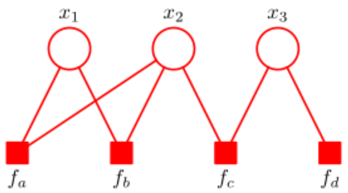
Figure: Factor graph corresponding to $P(\mathbf{x}) = f_a(x_1, x_2)f_b(x_1, x_2)f_c(x_2, x_3)f_d(x_3)$

Note that there are two factors $f_a(x_1, x_2)$ and $f_b(x_1, x_2)$ that are defined over the same set of variables.

In an undirected graph, the product of two such factors would simply be lumped together into the same clique potential. Similarly, $f_c(x_2, x_3)$ and $f_d(x_3)$ could be combined into a single potential over x_2 and x_3.

Factor graphs

Example I



Credit: *Pattern Recognition and Machine Learning*, *Christopher M. Bishop p. 400*

Figure: Factor graph corresponding to $P(\mathbf{x}) = f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$

Note that there are two factors $f_a(x_1, x_2)$ and $f_b(x_1, x_2)$ that are defined over the same set of variables.
In an undirected graph, the product of two such factors would simply be lumped together into the same clique potential. Similarly, $f_c(x_2, x_3)$ and $f_d(x_3)$ could be combined into a single potential over x_2 and x_3.
The factor graph, however, keeps such factors explicit and so is able to convey more detailed information about the underlying factorization.

Factor graphs

Example II

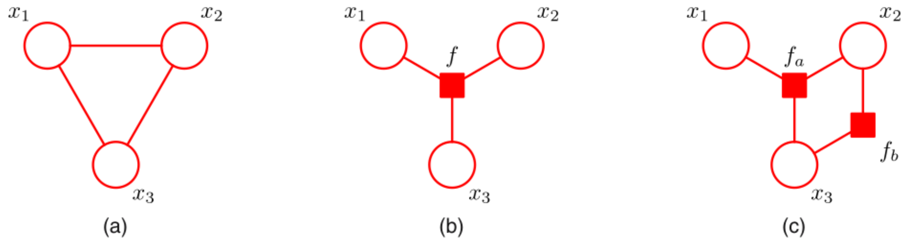


Figure 8.41 (a) An undirected graph with a single clique potential $\psi(x_1, x_2, x_3)$. (b) A factor graph with factor $f(x_1, x_2, x_3) = \psi(x_1, x_2, x_3)$ representing the same distribution as the undirected graph. (c) A different factor graph representing the same distribution, whose factors satisfy $f_a(x_1, x_2, x_3) f_b(x_1, x_2) = \psi(x_1, x_2, x_3)$.

Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 400*

Factor graphs

Construction

Undirected graph Create variable nodes corresponding to the nodes in the original undirected graph, and then create additional factor nodes corresponding to the maximal cliques x_s. The factors $f_s(x_s)$ are then set equal to the clique potentials. Note that there may be several different factor graphs that correspond to the same undirected graph.

Factor graphs

Construction

Undirected graph Create variable nodes corresponding to the nodes in the original undirected graph, and then create additional factor nodes corresponding to the maximal cliques x_s. The factors $f_s(x_s)$ are then set equal to the clique potentials. Note that there may be several different factor graphs that correspond to the same undirected graph.

Drirected graphs Create variable nodes in the factor graph corresponding to the nodes of the directed graph, and then create factor nodes corresponding to the conditional distributions, and then finally add the appropriate links. Again, there can be multiple factor graphs all of which correspond to the same directed graph.

Trees

We have seen that exact inference on a graph comprising a *chain of nodes* can be performed efficiently in time that is *linear in the number of nodes*, using an algorithm that can be interpreted in terms of *messages passed along the chain*. More generally, inference can be performed efficiently using local message passing on a broader class of graphs called *trees*.

Trees

We have seen that exact inference on a graph comprising a *chain of nodes* can be performed efficiently in time that is *linear in the number of nodes*, using an algorithm that can be interpreted in terms of *messages passed along the chain*. More generally, inference can be performed efficiently using local message passing on a broader class of graphs called *trees*.

Undirected graph Tree is defined as a graph in which there is one, and only one, path between any pair of nodes. Such graphs therefore *do not have loops*.

Trees

We have seen that exact inference on a graph comprising a *chain of nodes* can be performed efficiently in time that is *linear in the number of nodes*, using an algorithm that can be interpreted in terms of *messages passed along the chain*. More generally, inference can be performed efficiently using local message passing on a broader class of graphs called *trees*.

Undirected graph Tree is defined as a graph in which there is one, and only one, path between any pair of nodes. Such graphs therefore *do not have loops*.

Drirected graphs Tree is defined such that there is a single node, called the *root*, which has no parents, and all other nodes have one parent.

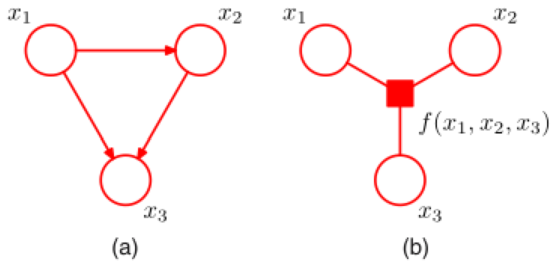
Trees

If we convert a directed tree into an undirected graph, we see that the *moralization step* will not add any links as *all nodes have at most one parent*, and as a consequence the corresponding moralized graph will be an undirected tree.

Trees

If we convert a directed tree into an undirected graph, we see that the *moralization step* will not add any links as *all nodes have at most one parent*, and as a consequence the corresponding moralized graph will be an undirected tree.
In fact, *local cycles in a directed graph* due to *links connecting parents of a node* can be removed on conversion to a factor graph by defining the appropriate factor function.

Figure 8.44 (a) A fragment of a directed graph having a local cycle. (b) Conversion to a fragment of a factor graph having a tree structure, in which $f(x_1, x_2, x_3) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)$.



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 400*

Polytree

If there are *nodes in a directed graph that have more than one parent*, but there is still *only one path (ignoring the direction of the arrows) between any two nodes*, then the graph is a called a *polytree*.

Polytree

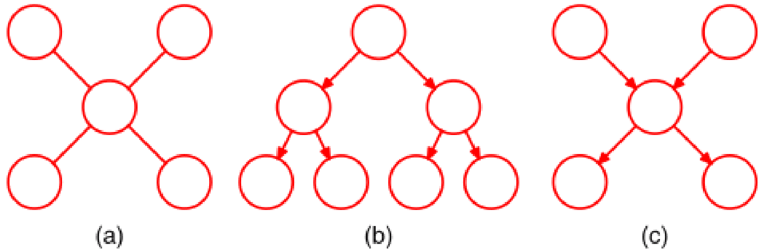
If there are *nodes in a directed graph that have more than one parent*, but there is still *only one path (ignoring the direction of the arrows) between any two nodes*, then the graph is a called a *polytree*.

Such a graph will have *more than one node with the property of having no parents*, and furthermore, the corresponding *moralized undirected graph will have loops.*

Polytree

If there are *nodes in a directed graph that have more than one parent*, but there is still *only one path (ignoring the direction of the arrows) between any two nodes*, then the graph is a called a *polytree*.

Such a graph will have *more than one node with the property of having no parents*, and furthermore, the corresponding *moralized undirected graph will have loops.*



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 399*

Figure: Examples of tree-structured graphs, showing (a) an undirected tree, (b) a directed tree, and (c) a directed polytree.

Polytree

Example

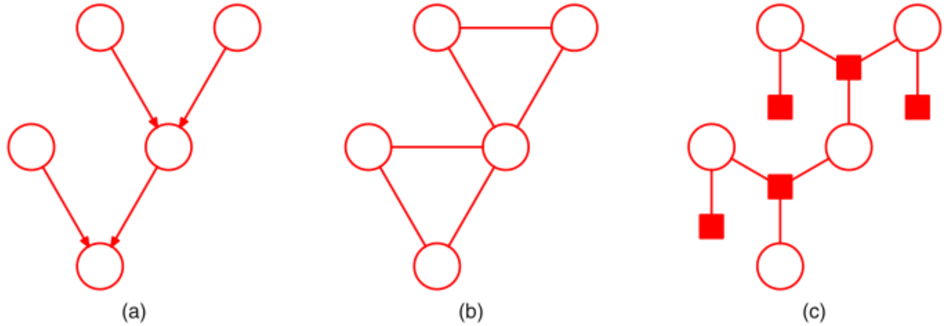


Figure 8.43 (a) A directed polytree. (b) The result of converting the polytree into an undirected graph showing the creation of loops. (c) The result of converting the polytree into a factor graph, which retains the tree structure.

Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 402*

The sum-product algorithm

We shall now make use of the *factor graph framework* to derive a powerful class of *efficient*, *exact* inference algorithms that are applicable to *tree-structured graphs*.

The sum-product algorithm

We shall now make use of the *factor graph framework* to derive a powerful class of *efficient*, *exact* inference algorithms that are applicable to *tree-structured graphs*.

We shall suppose that all of the *variables in the model are discrete*, and marginalization corresponds to *performing sums*. The framework, however, is cable to *linear-Gaussian models* in which case *marginalization involves integration*.

The sum-product algorithm

We shall now make use of the *factor graph framework* to derive a powerful class of *efficient*, *exact* inference algorithms that are applicable to *tree-structured graphs*.

We shall suppose that all of the *variables in the model are discrete*, and marginalization corresponds to *performing sums*. The framework, however, is cable to *linear-Gaussian models* in which case *marginalization involves integration*.

We begin by considering the problem of *finding the marginal* $P(x)$ for particular variable node x. For the moment, we shall suppose that *all of the variables are hidden*. By definition, the marginal is obtained by summing the joint distribution over all variables except x,

$$P(x) = \sum_{\boldsymbol{x} \setminus x} P(\boldsymbol{x}). \tag{22}$$

The sum-product algorithm

We shall now make use of the *factor graph framework* to derive a powerful class of *efficient*, *exact* inference algorithms that are applicable to *tree-structured graphs*.

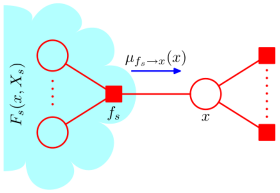
We shall suppose that all of the *variables in the model are discrete*, and marginalization corresponds to *performing sums*. The framework, however, is cable to *linear-Gaussian models* in which case *marginalization involves integration*.

We begin by considering the problem of *finding the marginal* $P(x)$ for particular variable node x. For the moment, we shall suppose that *all of the variables are hidden*. By definition, the marginal is obtained by summing the joint distribution over all variables except x,

$$P(x) = \sum_{\boldsymbol{x} \setminus x} P(\boldsymbol{x}). \tag{22}$$

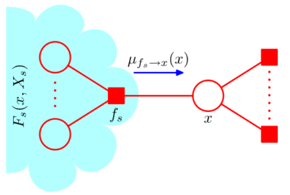
The idea is to substitute $P(x)$ using the *factor graph expression* $P(\boldsymbol{x}) = \prod_s f_s(\boldsymbol{x}_s)$ and then interchange summations and products in order to obtain an efficient algorithm.

The sum-product algorithm



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 404*

The sum-product algorithm



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 404*

The joint distribution can be written as a product of the form

$$P(\boldsymbol{x}) = \prod_{s \in \mathrm{ne}(x)} F_S(x, X_s), \tag{23}$$

where $\mathrm{ne}(x)$ denotes the set of factor nodes that are neighbours of x, X_s denotes the set of all variables in the *subtree* connected to the variable nodes x via the factor node f_s, and $F_s(x, X_s)$ represent the *product of all the factors in the group* associated with factor f_s.

The sum-product algorithm

Substituting eq. 23 into 22 and *interchanging the sums and products*, we obtain

$$P(x) = \prod_{s \in \mathrm{ne}(x)} \left[\sum_{X_s} F_S(x, X_s) \right] \tag{24}$$

The sum-product algorithm

Substituting eq. 23 into 22 and *interchanging the sums and products*, we obtain

$$P(x) = \prod_{s \in \mathrm{ne}(x)} \left[\sum_{X_s} F_S(x, X_s) \right] \tag{24}$$

Here we introduce *message functions from factor nodes to variable factor nodes*

$$\mu_{f_s \to x}(x) = \sum_{X_s} F_S(x, X_s) \tag{25}$$

The sum-product algorithm

Substituting eq. 23 into 22 and *interchanging the sums and products*, we obtain

$$P(x) = \prod_{s \in \mathrm{ne}(x)} \left[\sum_{X_s} F_S(x, X_s) \right] \tag{24}$$

Here we introduce *message functions from factor nodes to variable factor nodes*

$$\mu_{f_s \to x}(x) = \sum_{X_s} F_S(x, X_s) \tag{25}$$

The required marginal P(x) is given by the product of all the incoming messages arriving

at node x.

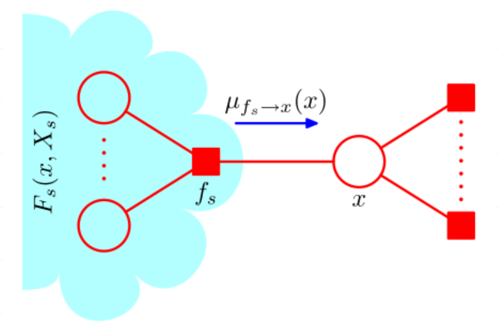
The sum-product algorithm

How to get *messages*?

The sum-product algorithm

How to get *messages*?

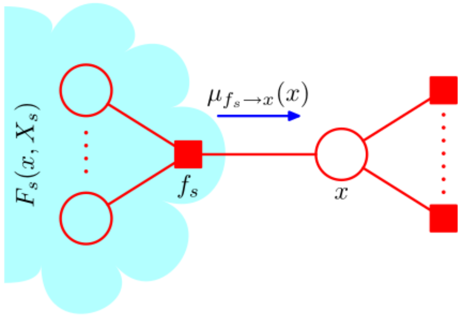
Let us recall



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 404*

The sum-product algorithm

How to get *messages*?
Let us recall



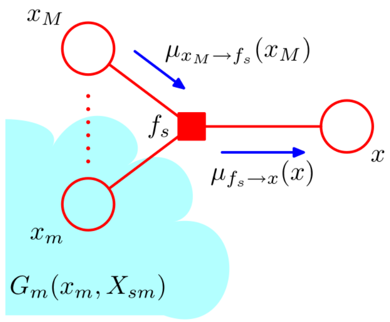
Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 404*

Note that each factor $F_s(x, X_s)$ is described by a *(sub-)graph* of factors and so can itself be factorized!

The sum-product algorithm

In particular, we can write

$$F_s(x, X_s) = f_s(x, x_1, \dots, x_M) G_1(x_1, X_{s1}) \cdots G_M(x_M, X_{sM}) \tag{26}$$



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 405*

The sum-product algorithm

In particular, we can write

$$F_s(x, X_s) = f_s(x, x_1, \ldots, x_M) G_1(x_1, X_{s1}) \cdots G_M(x_M, X_{sM}) \qquad (27)$$

Combining equations gives us

$$\begin{aligned} \mu_{f_s \to x}(x) &= \sum_{x_1} \ldots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \left[\sum_{X_{xm}} G_m(x_m, X_{sm}) \right] \\ &= \sum_{x_1} \ldots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m) \qquad (8.66) \end{aligned}$$

Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 404*

where $\mathrm{ne}(f_s)$ denotes the set of variable nodes that are neighbours of the factor node f_s.

The sum-product algorithm

$$\begin{aligned}\mu_{f_s\to x}(x) &= \sum_{x_1}\cdots\sum_{x_M} f_s(x,x_1,\ldots,x_M)\prod_{m\in\mathrm{ne}(f_s)\setminus x}\left[\sum_{X_{xm}} G_m(x_m,X_{sm})\right]\\ &= \sum_{x_1}\cdots\sum_{x_M} f_s(x,x_1,\ldots,x_M)\prod_{m\in\mathrm{ne}(f_s)\setminus x}\mu_{x_m\to f_s}(x_m) \qquad (8.66)\end{aligned}$$

Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 404*

where $\mathrm{ne}(f_s)$ denotes the set of variable nodes that are neighbours of the factor node f_s.

The sum-product algorithm

$$\begin{aligned}\mu_{f_s\to x}(x) &= \sum_{x_1}\cdots\sum_{x_M} f_s(x,x_1,\ldots,x_M)\prod_{m\in\mathrm{ne}(f_s)\setminus x}\left[\sum_{X_{xm}} G_m(x_m,X_{sm})\right] \\ &= \sum_{x_1}\cdots\sum_{x_M} f_s(x,x_1,\ldots,x_M)\prod_{m\in\mathrm{ne}(f_s)\setminus x}\mu_{x_m\to f_s}(x_m) \qquad (8.66)\end{aligned}$$

Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 404*

where $\mathrm{ne}(f_s)$ denotes the set of variable nodes that are neighbours of the factor node f_s. Here we define *message functions from variable nodes to factor nodes* as

$$\mu_{x_m\to f_s}(x_m) = \sum_{X_s m} G_m(x_m, X_{sm}) \tag{28}$$

The sum-product algorithm

$$\begin{aligned}\mu_{f_s\to x}(x) &= \sum_{x_1}\cdots\sum_{x_M} f_s(x,x_1,\ldots,x_M)\prod_{m\in\mathrm{ne}(f_s)\setminus x}\left[\sum_{X_{xm}} G_m(x_m,X_{sm})\right]\\ &= \sum_{x_1}\cdots\sum_{x_M} f_s(x,x_1,\ldots,x_M)\prod_{m\in\mathrm{ne}(f_s)\setminus x}\mu_{x_m\to f_s}(x_m) \qquad (8.66)\end{aligned}$$

Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 404*

where $\mathrm{ne}(f_s)$ denotes the set of variable nodes that are neighbours of the factor node f_s. Here we define *message functions from variable nodes to factor nodes* as

$$\mu_{x_m\to f_s}(x_m) = \sum_{X_sm} G_m(x_m, X_{sm}) \qquad (28)$$

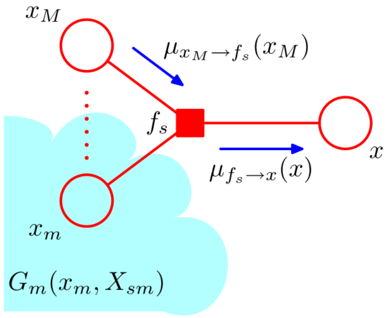
It is important to note that a factor node can send a message to a variable node once it has received incoming messages from all other neighbouring variable nodes.

The sum-product algorithm

We have therefore introduced *two distinct kinds of message*, those that go from factor nodes to variable nodes, and those that go from variable nodes to factor nodes. In each case, we see that messages passed along a link are always *a function of the variable associated with the variable node that link connects to*.

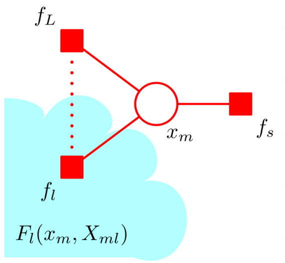
The sum-product algorithm

We have therefore introduced *two distinct kinds of message*, those that go from factor nodes to variable nodes, and those that go from variable nodes to factor nodes. In each case, we see that messages passed along a link are always *a function of the variable associated with the variable node that link connects to*.



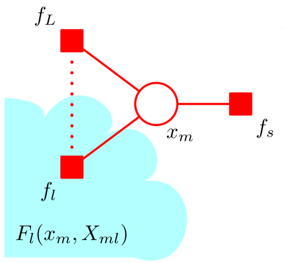
Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 405*

The sum-product algorithm



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 406*

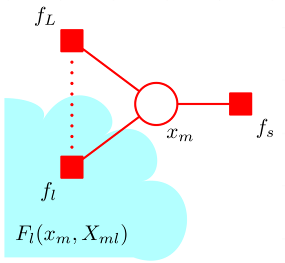
The sum-product algorithm



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 406*

Finally, we derive an expression for evaluating the *messages from variable nodes to factor nodes*, again by making use of the (sub-)graph factorization.

The sum-product algorithm



Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 406*

Finally, we derive an expression for evaluating the *messages from variable nodes to factor nodes*, again by making use of the (sub-)graph factorization.
The term $G_m(x_m, X_{sm})$ associated with node x_m is given by a product of terms $F_l(x_m, X_{ml})$ each associated with one of the factor nodes $f_l$that is linked to node x_m (excluding node f_s), so that

$$G_m(x_m, X_{sm}) = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} F_l(x_m, X_{ml}) \tag{29}$$

The sum-product algorithm

By putting eq. 29 into 28 and *interchanging the sums and products*, we obtain

$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \left[\sum_{X_{ml}} F_l(x_m, X_{ml}) \right] = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m) \tag{30}$$

The sum-product algorithm

By putting eq. 29 into 28 and *interchanging the sums and products*, we obtain

$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \left[\sum_{X_{ml}} F_l(x_m, X_{ml}) \right] = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m) \tag{30}$$

Thus to evaluate the message sent by a *variable node to an adjacent factor node* along the connecting link, we simply take the *product of the incoming messages along all of the other links*.

The sum-product algorithm

By putting eq. 29 into 28 and *interchanging the sums and products*, we obtain

$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \left[\sum_{X_{ml}} F_l(x_m, X_{ml}) \right] = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m) \qquad (30)$$

Thus to evaluate the message sent by a *variable node to an adjacent factor node* along the connecting link, we simply take the *product of the incoming messages along all of the other links*.
Note that any variable node that has *only two neighbours performs no computation but simply passes messages* through unchanged.

The sum-product algorithm

By putting eq. 29 into 28 and *interchanging the sums and products*, we obtain

$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \left[\sum_{X_{ml}} F_l(x_m, X_{ml}) \right] = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m) \tag{30}$$

Thus to evaluate the message sent by a *variable node to an adjacent factor node* along the connecting link, we simply take the *product of the incoming messages along all of the other links*.
Note that any variable node that has *only two neighbours performs no computation but simply passes messages* through unchanged.
Also, we note that a variable node can send a message to a factor node once it has received incoming messages from all other neighbouring factor nodes.

The sum-product algorithm

Recall the original goal –

The sum-product algorithm

Recall the original goal – calculate the marginal for variable node x, and that this marginal is given by the *product of incoming messages along all of the links arriving at that node*. Each of these messages can be *computed recursively* in terms of other messages.

The sum-product algorithm

Recall the original goal – calculate the marginal for variable node x, and that this marginal is given by the *product of incoming messages along all of the links arriving at that node*. Each of these messages can be *computed recursively* in terms of other messages.

In order to start this recursion, we can view the node x as the *root* of the tree and *begin at the leaf nodes*.

The sum-product algorithm

Recall the original goal – calculate the marginal for variable node x, and that this marginal is given by the *product of incoming messages along all of the links arriving at that node*. Each of these messages can be *computed recursively* in terms of other messages.

In order to start this recursion, we can view the node x as the *root* of the tree and *begin at the leaf nodes*.
If the leaf node is a *variable node*, then the message that it sends along its *one and only link* is given by

$$\mu_{x\to f}(x) = 1. \tag{31}$$

The sum-product algorithm

Recall the original goal – calculate the marginal for variable node x, and that this marginal is given by the *product of incoming messages along all of the links arriving at that node*. Each of these messages can be *computed recursively* in terms of other messages.

In order to start this recursion, we can view the node x as the *root* of the tree and *begin at the leaf nodes*.
If the leaf node is a *variable node*, then the message that it sends along its *one and only link* is given by

$$\mu_{x \to f}(x) = 1. \tag{31}$$

Similarly, if the leaf node is a *factor node*, then the message sent should be

$$\mu_{f \to x}(x) = f(x) \tag{32}$$

The sum-product algorithm

Recall the original goal – calculate the marginal for variable node x, and that this marginal is given by the *product of incoming messages along all of the links arriving at that node*. Each of these messages can be *computed recursively* in terms of other messages.

In order to start this recursion, we can view the node x as the *root* of the tree and *begin at the leaf nodes*.
If the leaf node is a *variable node*, then the message that it sends along its *one and only link* is given by

$$\mu_{x \to f}(x) = 1. \tag{31}$$

Similarly, if the leaf node is a *factor node*, then the message sent should be

$$\mu_{f \to x}(x) = f(x) \tag{32}$$

When all nodes will send their messages sequentially we can evaluate the marginal of intrest – Px. For an inductive proof check [1] p.407.

The sum-product algorithm

Rethinking

Now suppose we wish to find the *marginals for every variable node in the graph*.

The sum-product algorithm

Rethinking

Now suppose we wish to find the *marginals for every variable node in the graph*. This could be done by simply running the above algorithm *afresh for each such node*. However, this would be *very wasteful* as many of the required computations would be repeated.

The sum-product algorithm

Rethinking

Now suppose we wish to find the *marginals for every variable node in the graph*.
This could be done by simply running the above algorithm *afresh for each such node*.
However, this would be *very wasteful* as many of the required computations would be repeated.

Can we do any better?

The sum-product algorithm

Rethinking

Now suppose we wish to find the *marginals for every variable node in the graph*.
This could be done by simply running the above algorithm *afresh for each such node*.
However, this would be *very wasteful* as many of the required computations would be repeated.

Can we do any better?

Yes, we can!

The sum-product algorithm

All marginals

- Arbitrarily pick any (variable or factor) node and designate it as the root.

The sum-product algorithm

All marginals

- Arbitrarily pick any (variable or factor) node and designate it as the root.
- Propagate messages from the leaves to the root as before. At this point, the root node will have received messages from all of its neighbours.

The sum-product algorithm

All marginals

- Arbitrarily pick any (variable or factor) node and designate it as the root.
- Propagate messages from the leaves to the root as before. At this point, the root node will have received messages from all of its neighbours.
- Send out messages from the root to all of its neighbours. These in turn will then have received messages from all of their neighbours and so can send out messages along the links going away from the root, and so on.

The sum-product algorithm

All marginals

- Arbitrarily pick any (variable or factor) node and designate it as the root.
- Propagate messages from the leaves to the root as before. At this point, the root node will have received messages from all of its neighbours.
- Send out messages from the root to all of its neighbours. These in turn will then have received messages from all of their neighbours and so can send out messages along the links going away from the root, and so on.
- By now, a message will have passed in both directions across every link in the graph, and every node will have received a message from all of its neighbours.

The sum-product algorithm

All marginals

- Arbitrarily pick any (variable or factor) node and designate it as the root.
- Propagate messages from the leaves to the root as before. At this point, the root node will have received messages from all of its neighbours.
- Send out messages from the root to all of its neighbours. These in turn will then have received messages from all of their neighbours and so can send out messages along the links going away from the root, and so on.
- By now, a message will have passed in both directions across every link in the graph, and every node will have received a message from all of its neighbours.
- THE END!

The sum-product algorithm

All marginals

- Arbitrarily pick any (variable or factor) node and designate it as the root.
- Propagate messages from the leaves to the root as before. At this point, the root node will have received messages from all of its neighbours.
- Send out messages from the root to all of its neighbours. These in turn will then have received messages from all of their neighbours and so can send out messages along the links going away from the root, and so on.
- By now, a message will have passed in both directions across every link in the graph, and every node will have received a message from all of its neighbours.
- THE END!

The sum-product algorithm

All marginals

- Arbitrarily pick any (variable or factor) node and designate it as the root.
- Propagate messages from the leaves to the root as before. At this point, the root node will have received messages from all of its neighbours.
- Send out messages from the root to all of its neighbours. These in turn will then have received messages from all of their neighbours and so can send out messages along the links going away from the root, and so on.
- By now, a message will have passed in both directions across every link in the graph, and every node will have received a message from all of its neighbours.
- THE END!

Again a simple inductive proof can be found in [1].

The sum-product algorithm

All marginals – example

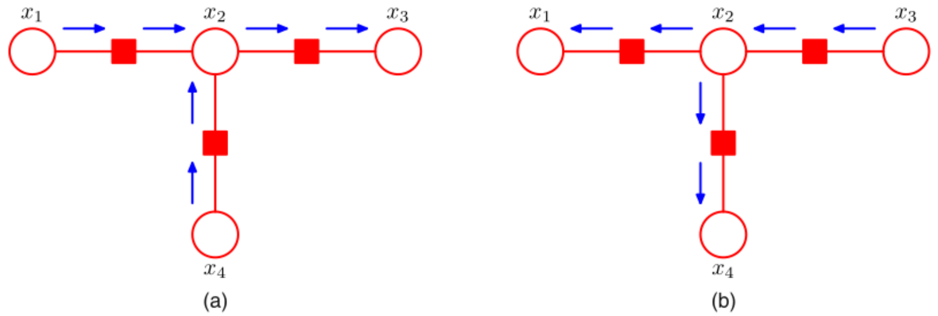


Figure 8.52 Flow of messages for the sum-product algorithm applied to the example graph in Figure 8.51. (a) From the leaf nodes x_1 and x_4 towards the root node x_3. (b) From the root node towards the leaf nodes.

Credit: *Pattern Recognition and Machine Learning, Christopher M. Bishop p. 410*

The sum-product algorithm

Marginals of all nodes in factors

Suppose we wish to find the *marginal distributions* $P(\boldsymbol{x}_s)$ associated with the sets of variables belonging to each of the factors.

The sum-product algorithm

Marginals of all nodes in factors

Suppose we wish to find the *marginal distributions* $P(\boldsymbol{x}_s)$ associated with the sets of variables belonging to each of the factors.
It is easy to see that the marginal associated with a factor is given by the *product of messages arriving at the factor node and the local factor at that node*

$$P(\boldsymbol{x}_s) = f_s(\boldsymbol{x}_s) \prod_{i \in \mathrm{ne}(f_s)} \mu_{x_i \to f_s}(x_i) \tag{33}$$

The sum-product algorithm

Marginals of all nodes in factors

Suppose we wish to find the *marginal distributions* $P(\boldsymbol{x}_s)$ associated with the sets of variables belonging to each of the factors.
It is easy to see that the marginal associated with a factor is given by the *product of messages arriving at the factor node and the local factor at that node*

$$P(\boldsymbol{x}_s) = f_s(\boldsymbol{x}_s) \prod_{i \in \mathrm{ne}(f_s)} \mu_{x_i \to f_s}(x_i) \tag{33}$$

If the factors are *parameterized functions* and we wish to *learn the values of the parameters* using the *EM algorithm*, then these marginals are precisely the quantities we will need to calculate in the E step.

The sum-product algorithm

Questions

There are still some topics connected with sum-product algorithm that where not covered during this lecture.

The sum-product algorithm

Questions

There are still some topics connected with sum-product algorithm that where not covered during this lecture.
These are:

- How about normalization? In case of directed graphs and conditional probabilities everything is properly normalized. For undirected graphs and arbitraty energy functions we end up with unnormalized probabilities. Can we normalize them efficiently?

The sum-product algorithm

Questions

There are still some topics connected with sum-product algorithm that where not covered during this lecture.
These are:

- How about normalization? In case of directed graphs and conditional probabilities everything is properly normalized. For undirected graphs and arbitraty energy functions we end up with unnormalized probabilities. Can we normalize them efficiently?
- How about observed data? How can we incorporate them into the algorithm?

The sum-product algorithm

Questions

There are still some topics connected with sum-product algorithm that where not covered during this lecture.
These are:

- How about normalization? In case of directed graphs and conditional probabilities everything is properly normalized. For undirected graphs and arbitraty energy functions we end up with unnormalized probabilities. Can we normalize them efficiently?
- How about observed data? How can we incorporate them into the algorithm?

The sum-product algorithm

Questions

There are still some topics connected with sum-product algorithm that where not covered during this lecture.
These are:

- How about normalization? In case of directed graphs and conditional probabilities everything is properly normalized. For undirected graphs and arbitraty energy functions we end up with unnormalized probabilities. Can we normalize them efficiently?
- How about observed data? How can we incorporate them into the algorithm?

Both questions has been answered in [1] Chapter 8.4.4 The sum-product algorithm p.407-411

The sum-product algorithm

Let us sum up the sum-product algorithm. It can be applied to

- Find marginal distribution of a single variable

The sum-product algorithm

Let us sum up the sum-product algorithm. It can be applied to

- Find marginal distribution of a single variable
- Find marginal distributions of all variables

The sum-product algorithm

Let us sum up the sum-product algorithm. It can be applied to

- Find marginal distribution of a single variable
- Find marginal distributions of all variables
- Find joint marginal distribution of variables connected to a single factor

The sum-product algorithm

Let us sum up the sum-product algorithm. It can be applied to

- Find marginal distribution of a single variable
- Find marginal distributions of all variables
- Find joint marginal distribution of variables connected to a single factor

The sum-product algorithm

Let us sum up the sum-product algorithm. It can be applied to

- Find marginal distribution of a single variable
- Find marginal distributions of all variables
- Find joint marginal distribution of variables connected to a single factor

What we usually want to do?

- Find a setting of the variables that has the largest probability

The sum-product algorithm

Let us sum up the sum-product algorithm. It can be applied to

- Find marginal distribution of a single variable
- Find marginal distributions of all variables
- Find joint marginal distribution of variables connected to a single factor

What we usually want to do?

- Find a setting of the variables that has the largest probability
- Find the value of that probability

The sum-product algorithm

Let us sum up the sum-product algorithm. It can be applied to

- Find marginal distribution of a single variable
- Find marginal distributions of all variables
- Find joint marginal distribution of variables connected to a single factor

What we usually want to do?

- Find a setting of the variables that has the largest probability
- Find the value of that probability

The sum-product algorithm

Let us sum up the sum-product algorithm. It can be applied to

- Find marginal distribution of a single variable
- Find marginal distributions of all variables
- Find joint marginal distribution of variables connected to a single factor

What we usually want to do?

- Find a setting of the variables that has the largest probability
- Find the value of that probability

This can be done with the *max-sum algorithm* which has a lot in common with the sum-product algorithm. If you are interested in this subject please check [1] Chapter 8.4.5 The max-sum algorithm p.411-415.

Bibliography I

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Peter Clifford. “Markov random fields in statistics”. In: ().

[3] S. L. Lauritzen and N. Wermuth. “Graphical Models for Associations between Variables, some of which are Qualitative and some Quantitative”. In: *Ann. Statist.* 17.1 (Mar. 1989), pp. 31–57.