Message Passing and Expectation Propagation

Efficient Inference in large scale machine learning

Christoph Dehner
Department of Informatics
Technische Universität München
dehner@in.tum.de

ABSTRACT

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

1. INTRODUCTION

Probabilistic graphical models like Markov Random Fields or Bayesian Networks provide clear and illustrative ways to describe probabilistic processes. In such models, nodes represent random variables and edges their conditional dependencies. The joint distribution of all involved variables can be expressed as a product of factors, which are observed or given specific values during modelling. As an example, a Bayesian network with three variables is given in figure 1. It defines the values of x_1 and x_3 to be conditionally independent given x_2.

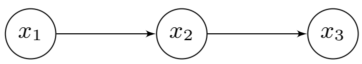

Figure 1: Bayesian network with three variables. x_1 and x_3 are conditionally independent given x_2.

Typical tasks in such a scenario are to do Bayesian inference to extract marginal distribution of specific variables or find maximum apriori estimates. However, with an increasing number of variables these task can become computationally very expensive.

The naive approach for marginalisation is to sum out all but one variable from the joint distribution $p(X)$. This is formally shown in equation 1, where $X \setminus x_i$ describes the set of all random variables expect x_i. This strategy requires exponentially in the number of variables many evaluations of the joint distribution and is thus not applicable to bigger networks.

$$p(x_i) = \sum_{X \setminus x_i} p(\mathbf{x}) \tag{1}$$

Therefore, this paper explains more efficient algorithms to do large scale Bayesian inference in graphical models. The next chapter *todo* deals with exact inference and presents two message passing algorithms to calculate marginals and maximum apriori estimates for graphical models. Subsequently, in chapter *todo* expectation propagation as a method of approximate inference is explained.

2. EXACT INFERENCE

The intuition of the algorithms presented in this section is to exploit independence properties of the random variables. Graphical models define the joint probability to be expressed as a product of factors, considering the conditional independences expressed by the graph. For Bayesian networks those factors correspond to conditional distributions, in Markov Random Fields to clique potentials. Assuming appropriately normalized factors, the joint distribution can be written as a product according to equation 2. The index s iterates here over all factors of the graph; X_s defines the subset of variables, the factor s depends on.

$$p(X) = \prod_s f_s(X_s) \tag{2}$$

Expressing the joint distribution as the product defined by the graph allows to exchange summations and multiplications during marginalization. By this way, marginalization can often be done a lot more efficient. Equation 3 demonstrates this transformation with the Bayesian network from figure 1, whose joint distribution $p(X)$ is defined as $p(X) = p(x_1)p(x_2|x_1)p(x_3|x_1)$.

$$\begin{aligned} p(x_2) &= \sum_{x_1}\sum_{x_3} p(x_1)p(x_2|x_1)p(x_3|x_2) \\ &= \sum_{x_1}\Big[p(x_1)p(x_2|x_1)\sum_{x_3}\Big[p(x_3|x_2)\Big]\Big] \\ &= \underbrace{\Big[\sum_{x_1} p(x_1)p(x_2|x_1)\Big]}_{\mu_{x_1\to x_2}} \cdot \underbrace{\Big[\sum_{x_3} p(x_3|x_2)\Big]}_{\mu_{x_3\to x_2}} \end{aligned} \tag{3}$$

Here, the sum of products from the naive approach for marginalization is transformed to a product of sums, reducing the exponential computational complexity in the number of random variables to linear effort.

The brackets in the last line of equation 3 reveal a powerful interpretation of the marginalizing. The marginal distribution $p(x_2)$ consists of two messages $\mu_{x_1 \to x_2}$ and $\mu_{x_3 \to x_2}$ from its neighboring cells x_1 and x_3. For a longer chain of random variables these messages would again be comprised by messages from their neighboring cell. Applied to a chain of arbitrary length, this method gives a recursive pattern in which messages for the marginalization are sent to the marginalized variable node from both ends of the chain.
This message passing idea is the foundation for the two inference algorithms presented in this chapter subsequently. Before that the next section introduces factor graphs, the structure these algorithms operate on.

2.1 Factor graphs

Factor graphs make the factorisation of a joint probability distribution explicit. The can be generated from Bayesian networks as well as from Markov random fields and thus allow to define inference algorithms independent of how the underlying probabilistic graphical model was introduced.
Additional to the variable nodes, a factor graph also consists of factor nodes representing the factors of the decomposed joint probability distribution as in equation 2. Edges connect the factor nodes to all variables they depend on. By this way they a bipartite graph with variable nodes (usually visualised by circles) on the one side and and factors (depicted as rectangles) on the other side.
Depending on the exact factorisation, a distribution defined by a Bayesian network or Markov random field can be represented by different factor graphs. Figure *todo* depicts a factor graph of the Bayesian network from figure 1, in which all conditional distribution are represented by separate factors.
The inference algorithms on factor graphs of the following sections are valid for trees. Such factor graphs with exactly one path between any pair of nodes can be generated from undirected trees in the case of a Markov random field model as well as from from directed trees and polytrees, if the factor graph is derived from a Bayesian network. More detailed description how to derive factor graph from probabilistic graph models can be found in todo: add reference.

2.2 Marginalization

The sum-product algorithm presented in this section generalises the idea of exchanging summation and multiplication during marginalisation. It consists of local messages passed between factor and variable nodes in a factor graph.
As a starting point let us consider the calculation of the marginal distribution $p(x_i)$ in the factor graph of figure 2. Furthermore we define $F_s(s, X_s)$ for all neighbours s of x_i as the product of all factors in the respective sub-tree of the neighbour. This is visualised by blue circles in figure 2.
Analogous to the general factorisation defined by the graphical model from equation 2 the joint probability $p(\mathbf{x})$ can then be expressed as a product of those neighbour-factor of x_i:

$$p(X) = \prod_{s \in ne(x_i)} F_s(x_i, X_s) \tag{4}$$

Inserting this representation of the joint distribution in the naive marginalisation formula from equation 1 allows to exchange sum and product. By this way the marginal distribution of x_i is expressed as the product of messages $\mu_{f_s \to x_i}$

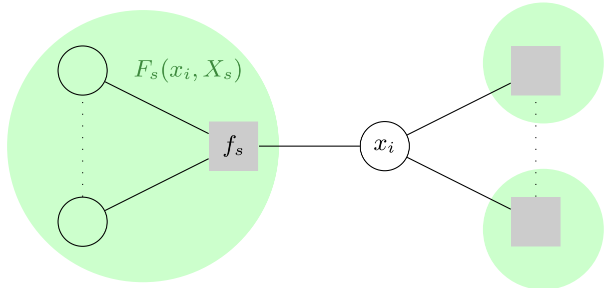

Figure 2: Exemplary factor graph to deduce the messages of the sum-product algorithm.

from all its neighbours in the factor graph:

$$\begin{aligned} p(x_i) &= \sum_{X \setminus x_i} \Big[\prod_{s \in ne(x_i)} F_s(x_i, X_s) \Big] \\ &= \prod_{s \in ne(x_i)} \Big[\sum_{X_s} F_s(x_i, X_s) \Big] =: \prod_{s \in ne(x_i)} \mu_{f_s \to x_i} \end{aligned} \tag{5}$$

The factors $F_s(x, X_s)$ represent subtrees in the factor graphs and can thus be factorized again:

$$F_s(x_i, X_s) = f_s(x_i, X_s) G_1(x_1, X_{s_1}) \dots G_1(x_M, X_{s_M}). \tag{6}$$

The factors $G_1(x_1, X_{s_1}) \dots G_1(x_M, X_{s_M})$ represent here the subgraphs of all neighbours of f_s except from x_i and are illustrated by red circles in figure 3.

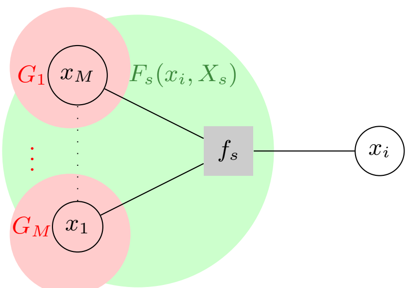

Figure 3: Exemplary factor graph to deduce the messages of the sum-product algorithm.

With that, the messages defined in equation 6 can be further evaluated to

$$\begin{aligned} \mu_{f_s \to x_i}(x_i) &= \sum_{X_s \setminus x_i} f_s(x_i, X_s) \prod_{m \in ne(f_s) \setminus x_i} \Big[\sum_{X_{sm}} G_m(x_m, X_{sm}) \Big] \\ &= \sum_{\mathbf{x_s} \setminus x_i} f_s(x_i, X_s) \prod_{m \in ne(f_s) \setminus x_i} \mu_{x_m \to f_s}(x_m). \end{aligned} \tag{7}$$

We have now defined messages from factor to variable nodes. $\mu_{x_m \to f_s}(x_m) := \sum_{X_{sm}} G_m(x_m, X_{sm})$. Going one step further yields an expression for messages from variables to factors as well. Analogously to before, $G_m(x_m, X_{sm})$ can be factorised again to

$$G_m(x_m, X_{sm}) = \prod_{l \in ne(x_m) \setminus f_s} F_l(x_m, X_{sml}), \tag{8}$$

as visualised in figure 4.

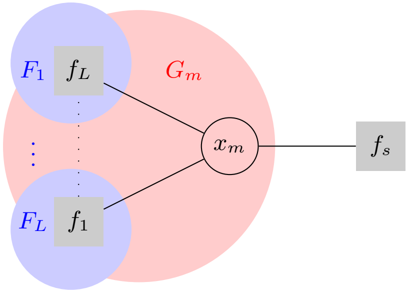

Figure 4: Exemplary factor graph to deduce the messages of the sum-product algorithm.

Inserting this factorisation into the definition of $\mu_{x_m \to f_s}(x_m)$ and exchanging summation and multiplication reveals an explicite formulat for this messages:

$$\begin{aligned}\mu_{x_m \to f_s}(x_m) &= \sum_{X_{sm}} \Big[\prod_{l \in ne(x_m) \setminus f_s} F_l(x_m, X_{sml}) \Big] \\ &= \prod_{l \in ne(x_m) \setminus f_s} \Big[\sum_{X_{sml}} F_l(x_m, X_{sml}) \Big] \\ &= \prod_{l \in ne(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)\end{aligned} \tag{9}$$

We have now induced how a marginal distribution $p(x_i)$ can be expressed by local messages passed between factors and variables through the factor graph. For leave nodes those messages can be stated explicitely. For a variable node as leave of the factor graph equation 11 shows that the message, the node sends to its only neighbor, is given by

$$\mu_{x \to f}(x) = 1. \tag{10}$$

Similarly, from equation 8 then concludes the initial value

$$\mu_{f \to x}(x) = f(x) \tag{11}$$

for a message from a leave factor node to its neighbouring variable node.
Having explicit start values for messages from the leaves, messages can propagated through the factor graphs to an arbitrarily chosen root node. This root node has then received messages from all its neighbours and can thus calculate its marginal distribution according to 5. Analogously, passing all messages back from the root to all leave nodes enables to calculate the marginals of all variables in the factor graph. For this calculation two time the number of edges in the factor graph many messages have to be computed, which is linear in the number of variables of the model. todo: marginal of a factor formula, connection to EM-algorithm, continuous RVs

2.3 Maximum a posteriori estimation

Next to computing marginal distributions, a frequent task is to find maximum a maximum a posteriori estimate. The aim is to finding values for all random variables, which maximize the joint probability distribution and can be formulated mathematically as

$$X^* = \arg\max_X p(X) = \arg\max_X \ \ln(p(X)). \tag{12}$$

The value for the maximal joint probability is given by

$$p(X^*) = \max_X \ \ln(p(X)). \tag{13}$$

The actual algorithms works similarly to the sum-product algorithm from the previous chapter, but with the exception that maximizations instead of sums over the random variables are performed. Analogously to before, the idea behind is to insert the factorized expression of the joint distribution from equation 6 into the problem formulation from equation 13 and exchange maximization and multiplication.
The second step of equation 12 contains a transformation, which allows a simplification. Since the logarithm is a strictly monotonic increasing function, it does not change the optimum X^* but transforms the products in the messages to sums. Thus, the algorithm derived in this section is not called "max-product", which the analogy to the max-sum algorithm would suggest, but "max-sum" algorithm.
Applying all discussed modification to the formulas deduced in the last chapter leads to the following messages:

$$\mu_{f_s \to x_i}(x_i) = \max_{X_s \setminus x_i} \Big[f_s(x_i, X_s) \sum_{m \in ne(f_s) \setminus x_i} \mu_{x_m \to f_s}(x_m) \Big] \tag{14}$$

$$\mu_{x_m \to f_s}(x_m) = \sum_{l \in ne(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m) \tag{15}$$

Also the message initializations for leave nodes from equation **??** and **??** must be adjusted with the logarithm, resulting in

$$\mu_{x \to f}(x) = 0, \tag{16}$$

$$\mu_{f \to x}(x) = \ln(f(x)). \tag{17}$$

By propagation messages from all leave nodes to an arbitrarily chosen root node, the maximal value of the joint probability distribution can be calculated in that root node. The message passing is again done in a fashion, that a node can forward its message to its parent node as soon it has received messages from all children.
Problemformulierung max-sum-algorithms

2.4 Inference in general graphs

loopy belief propagation

3. APPROXIMATE INFERENCE

Known as expectation propagation, similar to variational inference Minimize KL-divergence using moment matching. Interesting properties different from standard VI

3.1 Methodology

3.2 Expectation propagation in graphical models

4. SUMMARY AND OUTLOOK

4.1 Subsection

blabla with 2 sources [2; 1].

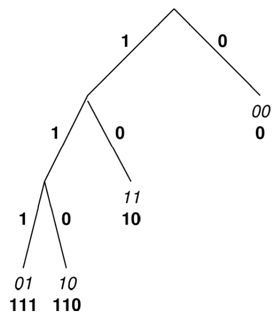

Figure 5: Tree

Table 1: Example table

Column 1	Column 2
0	1

5. REFERENCES

[1] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006.

[2] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.