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

This paper explains two efficient Bayesian inference methods basing on a factorized version of a joint probability distribution. The two message passing algorithms "Max-Sum" and Sum-Product" enable exact inference in tree-shaped graphical models. The underlying concept of message passing can also be generalized to arbitrary graphs.
By expectation propagation a second approximate inference technique is covered. Similarly to variational inference it aims at minimizing the KL divergence of an approximating distribution. This is done in an iterative way leading to a useful approximate inference method for many situations.

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 modeling. 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.

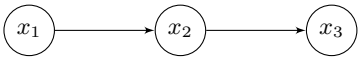

Figure 1: Bayesian network with three variables. x_1 and x_3 are conditionally independent given x_2.

Typical tasks in such a scenario are to do Bayesian inference to extract marginal distributions of specific variables or find maximum a posteriori estimates. However, with an increasing number of variables these task can become computationally very expensive.
The naive approach for marginalization 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 except x_i. This strategy requires exponentially many evaluations of the joint distribution in the number of variables and is thus not applicable to bigger networks.

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

Therefore, this paper explains more efficient algorithms to do large scale Bayesian inference in graphical models.
Before the different inference approaches are explained in detail, the next chapter 2 introduces factor graphs, the structure these algorithms operate on. Then, chapter 3 deals with message passing and presents two belief propagation algorithms to calculate exact marginals and maximum a posteriori estimates for tree-shaped graphical models. Furthermore, generalized message passing approaches for arbitrary graphs are presented at the end of the chapter. Subsequently, in chapter 4 expectation propagation as a method of approximate inference is explained.

2. FACTOR GRAPHS

Factor graphs make the factorization 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 form a bipartite graph with variable nodes (usually visualizes by circles) on the one side and and factors (depicted as rectangles) on the other side.
Depending on the exact factorization, a distribution defined by a Bayesian network or Markov random field can be represented by different factor graphs. Figure 2 depicts a factor graph of the Bayesian network from figure 1, in which all conditional distributions are represented by separate factors.

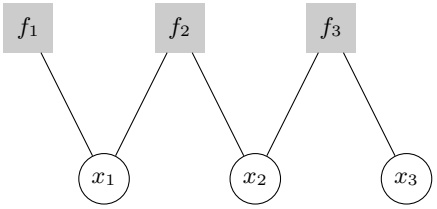

Figure 2: Factor graph of the Bayesian network from figure 1. The factors f_1, f_2 and f_3 correspond to the probability distributions $p(x_1)$, $p(x_2|x_1)$ and $p(x_3|x_2.)$

The inference algorithms on factor graphs of the following section are valid for trees. Such factor graphs with exactly one path between any pair of nodes can be generated from

Markov random fields and Bayesian networks, if the underlying graph of the model is also structured as a tree. More detailed descriptions of how to derive factor graph from probabilistic graph models can be found in [1].[1]

3. MESSAGE PASSING

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.

3.1 "Sum-Product" algorithm

The "Sum-Product" algorithm presented in this section generalizes the idea of exchanging summation and multiplication during marginalization. It consists of local messages passed between factor and variable nodes in a factor graph.

[1]Chapter 8.3.2 for the moralization step of undirected models and chapter 8.4.3 for the concrete buildup of a factor graph from a graphical model.

As a starting point let us consider the calculation of the marginal distribution $p(x_i)$ in the factor graph of figure 3. Furthermore we define $F_s(s, X_s)$ for all neighbors s of x_i as the product of all factors in the respective sub-tree of the neighbor. This is visualized by green circles in figure 3.

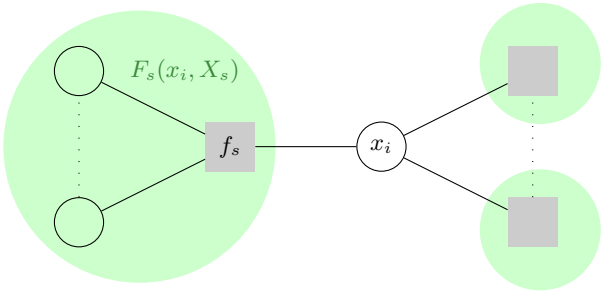

Figure 3: Exemplary factor graph to visualize the deduction of an expression for the marginal distribution of x_i in equation 5. The green circles define for all neighbors of x_i the product of all factors in the respective sub-tree and support the expression of the joint distribution in equation 4.

Analogously to the general factorization defined by the graphical model from equation 2 the joint probability $p(X)$ can then be expressed as a product of those neighbor-factors 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 marginalization 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 $\mu_{f_s\to x_i}$ from all its neighbors in the factor graph:

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

The factors $F_s(x, X_s)$ represent subtrees in the factor graph 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 neighbors of f_s except from x_i and are illustrated by red circles in figure 4.

With that, the messages defined in equation 5 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

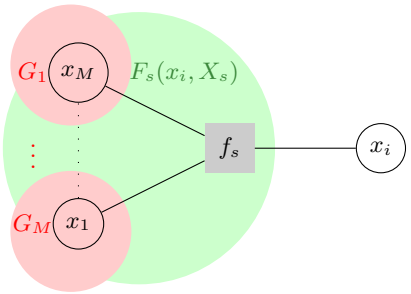

Figure 4: Exemplary factor graph to visualize the deduction of an expression for messages from factors to nodes equation 7. The red circles represent the factorization expressed in equation 6.

factorized 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 visualized in figure 5. The factors F_l are visualized by blue circles.

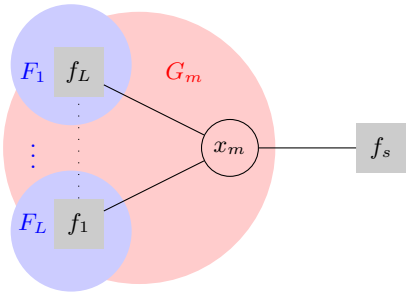

Figure 5: Exemplary factor graph to visualize the deduction of an expression for messages from nodes to factors equation 9. The blue circles represent the factorization expressed in equation 8.

Inserting this factorization into the definition of $\mu_{x_m \to f_s}(x_m)$ and exchanging summation and multiplication reveals an explicit formula 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 leaf nodes those messages can be stated explicitly. In case a variable node is leaf of the factor graph, equation 9 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 7 then concludes the initial value

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

for a message from a leaf factor node to its neighboring 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 neighbors and can thus calculate its marginal distribution according to 5. Analogously, passing all messages back from the root to all leaf nodes enables to calculate the marginals of all variables in the factor graph. During this calculation two messages are sent via each edge edge of the factor graph. The computational effort is thus linear in the number of edges and therefore also linear in the number of variables of the model.

3.2 "Max-Sum" algorithm

Next to computing marginal distributions, a frequent task is to find maximum a posteriori estimates. The aim is to finding values for all random variables, which maximize the joint probability distribution. This 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 algorithm 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 4 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 "Sum-Product" algorithm from the previous section 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 leaf nodes from equation 10 and 11 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 propagating messages from all leaf 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.
The missing step is to find the variable configuration for the maximal joint probability. The central maximization is performed in equation 14 over the variables of each factor before its message is passed further to the root. By saving the maximal configuration of each factor at this step, the values of the variables which give rise to the maximal joint probability can be obtained. In contrast to the "Sum-Product" algorithm, the "Max-Sum" algorithm does not require to propagate messages back from the root to all leaves.

3.3 Discussion

The two algorithms require a fixed number of passes through the graph. Thus their computational complexity depends on the number of edges, which increases linearly with the number of involved random variables. Next to this factor, the computational cost of the two algorithms also consists a polynomial factor in the number of values, the discrete random variables can take.
The big disadvantage of the "Sum-Product" and "Max-Sum" algorithms is that they only work on trees-shaped graphical models. To overcome this, the next section presents approaches to overcome this and do inference via message passing in general graphs.

3.4 Inference via message passing in general graphs

The discussed inference methods only produce valid results for trees. However, since the algorithms only consist of local messages passed through the graph, they can also be applied to graphs which contain circles. This is called "loopy belief propagation". For that, all messages need to be initialized by the unit function and a message passing schedule must be defined to state the order in which the messages are passed through the graph. Then, local messages can be passed through the graph multiple times until marginals or maximum a posteriori estimates can be extracted.
With circles in the graph, loopy belief propagation is not guaranteed to converge. Messages can also start to oscillate or converge to wrong values. In some situations loopy belief propagation produces only poor results, while in other situations the final outcome turned out to be quite accurate. Certain error-correcting codes for example are equivalent to loopy belief propagation.
There are also algorithms, which generalize the message passing idea to work for arbitrary graphs. The junction tree algorithm resolves circles in a general graph by a triangulation step and extraction of a maximal spanning tree. Depending on the underlying graph the algorithm requires exponential effort, but is efficient from the perspective that there are generally no cheaper approaches for exact inference in arbitrary graphs.
Linearized belief propagation is another approach to overcome loops in the graph. By restricting the values to be close to certain default values, the problem can be formulated in a matrix framework, for which there is a closed-form solution. More information can be found in [3].

4. EXPECTATION PROPAGATION

This chapter presents by expectation propagation (EP) a method for approximate inference. In the following section the methodology behind and the concrete algorithm are explained at first. Secondly, some interesting properties when applying expectation propagation to graphical models are highlighted. At last, a general discussion and comparison to variational inference is given.

4.1 Methodology

Similarly to variational inference (VI), in EP the true distribution $p(\mathbf{z})$ of a model is approximated by a simpler distribution $q(\mathbf{z})$. The approximation is computed and measured by minimizing the Kullback-Leibler divergence of the two functions. However, while in VI $KL(q||p)$ is considered, EP swaps the distributions and aims at minimizing $KL(p||q)$.
For $q(\mathbf{z})$ a member of the exponential family with the form

$$q(\mathbf{z}) = h(\mathbf{z})g(\eta)\exp(\eta^T(u(\mathbf{z}))) \tag{18}$$

must be chosen. With that the KL divergence becomes

$$KL(p||q) = -\ln g(\eta) - \eta^T \mathbf{E}_{p(\mathbf{z})}[u(\mathbf{z})] + const. \tag{19}$$

The distribution $q(\mathbf{z})$, which minimizes this KL divergence from equation 19, can be constructed by moment matching. Formulated differently, the distribution $q(\mathbf{z})$ from the exponential family, whose the natural parameters are equal to the moments of the true distribution $p(\mathbf{z})$ we want to approximate, minimizes the KL divergence from equation 19. If $q(\mathbf{z})$ is a Gaussian for example, its optimal mean corresponds to the mean of $p(\mathbf{z})$ and its optimal covariance is equal to the covariance of $p(\mathbf{z})$.

4.2 Expectation propagation algorithm

For the concrete EP algorithm we consider a model of hidden variables and parameters θ and observed data $\mathbf{D}$. Similarly to the previous chapter we assume that the joint distribution can be written as a product of factors f_i. However, one factor is not restricted to depend on only a subset of variables here, but has all latent variables and parameters (together represented by θ) as its arguments.

$$p(\theta, \mathbf{D}) = \prod_i f_i(\theta). \tag{20}$$

The goal is then to approximate the posterior distribution $p(\theta|\mathbf{D})$, which can be expressed by the joint distribution and the evidence $p(\mathbf{D})$:

$$p(\theta|\mathbf{D}) = \frac{1}{p(\mathbf{D})}\prod_i f_i(\theta, \mathbf{D}), \tag{21}$$

with

$$p(\mathbf{D}) = \int \prod_i f_i(\theta) d\theta. \tag{22}$$

Adapting to the structure of the true posterior distribution, $q(\theta)$ is chosen as a product of factors $\tilde{f}_i$ from a exponential family, which each correspond to one of the true factors f_i. By this way, $q(\theta)$ is also from this exponential family. $\frac{1}{Z}$ represents the normalizing factor for the distribution:

$$q(\theta) = \frac{1}{Z}\prod_i \tilde{f}_i(\theta) \tag{23}$$

The factors $\tilde{f}_i$ are refined iteratively. Starting with a rough initialization of all approximating factors $\tilde{f}_i$, in each iteration on factor is refined closer to its true factor f_i, while the

rest of the factors are fixed. Assuming we want to refine factor j, we first define the product of the fixed factors $q^{\setminus j}(\theta)$ for that:

$$q^{\setminus j}(\theta) = \prod_{i \neq j} \tilde{f}_i(\theta) \tag{24}$$

Then we evaluate a new approximation of the posterior $q^{new}(\theta)$ by matching the moments of $f_j \cdot q^{\setminus j}(\theta)$:

$$q^{new}(\theta) \propto f_j \; q^{\setminus j}(\theta) \tag{25}$$

Finally, we can extract and store a new approximation of $\tilde{f}_j$ from that. $\frac{1}{Z_j}$ again represents a necessary normalizing factor here:

$$\tilde{f}_j = \frac{1}{Z_j} \frac{q^{new}(\theta)}{q^{\setminus j}(\theta)} \tag{26}$$

4.3 Expectation propagation in graphical models

Whereas general expectation propagation allows the factors of the joint distribution to depend on all parameters and latent variables of the model (see equation 21), in graphical models the factors of the joint distribution depend on a subset of variables θ_i:

$$p(\theta|\mathbf{D}) = \frac{1}{p(\mathbf{D})} \prod_i f_i(\theta_i), \tag{27}$$

An interesting property gets visible if we use a fully factorized approximation function: Then, the EP algorithm becomes equal the "Sum-Product" algorithm. In other words, the "Sum-Product" algorithm turns out to be a special case of expectation propagation, if using fully independent variables in the approximation function.

With respect to the approximation function of the EP algorithm an interpretation can be formulated: Since the "Sum-Product" algorithm is an exact inference algorithm, a more flexible approximation function compared to the underlying model (as for example one with a few dependency connections in the graph left out) can increase the accuracy.

Another approach to incorporate some extent of generalization to expectation propagation is to refine a set of factors together within one iteration of the algorithm. However, for both methods there are still open research question how to find the best amount of independence for the approximation function of EP as well as the optimal grouping of factors for simultaneous refinement.

4.4 Discussion

A key disadvantage of expectation propagation is that is not guaranteed to converge. For the exponential family the resulting KL-divergence function is proven to have a stationary point, but the iteration of EP might not reach it.

Variational inference in contrast provides a lower bound in each optimization step and thus guarantees to converge. However, EP can nevertheless be a powerful inference method. Because variational inference operates with a lower bound it often underestimates the variance of the true distribution. Therefore, if expectation propagation converges, it often outperforms other variational methods.

On the other hand, if the algorithm does not converge, it might be for a good reason an the chosen exponential family approximates the true posterior only poorly.

Another aspect leading to conceptually different behavior between expectation propagation and variational inference comes from the approach how the KL divergence is minimized. Expectation propagation minimizes $KL(p||q)$, while variational inference aims at reducing $KL(q||p)$. For multimodal distribution this leads to significantly worse results for expectation propagation. Since the EP algorithm tries to incorporate all modes of the underlying probability distribution it can approximate function with more than one mode only badly. This is visualized in figure 6, which is taken from [1].

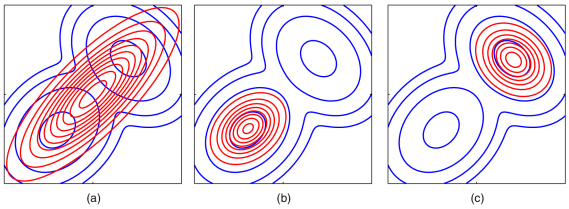

Figure 6: Approximation of a multimodal distribution by a) minimizing $KL(p||q)$, b) minimizing $KL(q||p)$ and c) same approach as b) but with a different result.

5. SUMMARY AND OUTLOOK

In summary, message passing as well as expectation propagation are useful inference methods. If the underlying graphical model is a tree, the "Max-Sum" and "Sum-Product" algorithm compute exact results in an efficient way.

For general models, there exist different approaches for approximate inference. Methods like loopy belief propagation or linearized belief propagation generalize the concept of message passing to arbitrary graphs. Expectation propagation follows an approach similar to variational inference and tries to find an approximation of the true distribution by minimizing the KL divergence. However, none of these methods is in general superior. Depending on the concrete problem setting, either of the presented method can produce more accurate results.

There is also ongoing research about the discussed approaches, especially for message passing. [3] does not only present a linearized belief propagation but also shows an incremental version. Furthermore it describes a single-pass variation, in which only one message for each node is calculated.

Finally, as second outlook [2] is presented. It considers Bayesian inference methods like expectation propagation on partitioned data. By this way, such inference methods could be applied to big data.

6. REFERENCES

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

[2] G. et al. Expectation propagation as a way of life: A framework for bayesian inference on partitioned data. 2017.

[3] W. Gatterbauer, S. Günnemann, D. Koutra, and C. Faloutsos. Linearized and single-pass belief propagation. *Proc. VLDB Endow.*, 8(5):581–592, Jan. 2015.

[4] T. P. Minka. *A Family of Algorithms for Approximate Bayesian Inference*. PhD thesis, Cambridge, MA, USA, 2001. AAI0803033.

[5] T. P. Minka. Expectation propagation for approximate bayesian inference. *CoRR*, abs/1301.2294, 2013.

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

[7] M. Seeger. Expectation propagation for exponential families. Technical report, 2005.