10-708: Probabilistic Graphical Models 10-708, Spring 2014

13 : Variational Inference: Loopy Belief Propagation

Lecturer: Eric P. Xing *Scribes: Rajarshi Das, Zhengzhong Liu, Dishan Gupta*

1 Introduction

The problem of probabilistic inference concerns answering queries about conditional and marginal probabilities in graphical models. Consider two disjoint subsets E and F of the nodes in a graphical model $\mathcal{G}$. A query regarding marginal distribution $p(x_F)$ can be calculated by the marginalization operation $\sum_{\mathcal{G}\setminus F} p(x)$. A query regarding conditional distribution $p(x_F|x_E)$ can be calculated by $p(x_F|x_E) = \frac{p(x_F,x_E)}{p(x_E)}$. A query could also ask to compute a mode of the density $\hat{x} = \arg\max_{x\in\mathcal{X}^m} p(x)$. In the previous lectures, we have learnt many exact inference techniques such as naive brute force marginalization, variable elimination and family of message passing algorithms such as sum-product, belief propagation and junction tree. In brute force and variable elimination techniques, individual queries are computed independently and as a result several intermediate terms may be computed repeatedly, while the message passing algorithms allows sharing of intermediate terms and hence is more effective in the long run.

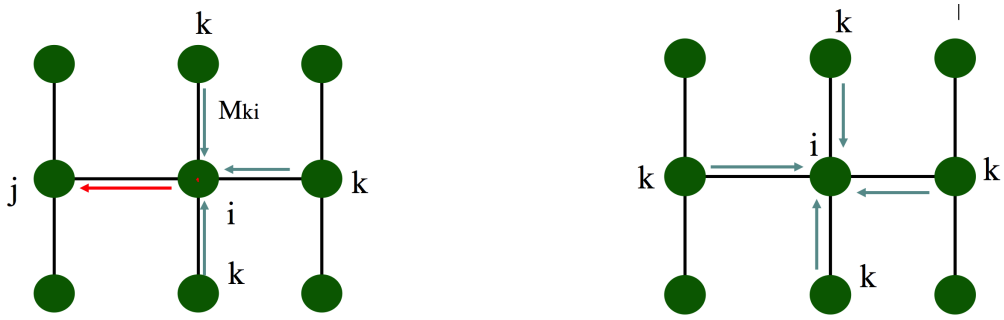
Algorithms like message passing perform well on tree structured graphs. However, for loopy graphs (graph that contains loops), messages may circulate indefinitely around the loops and hence may not converge. Even when they converge, the stable equilibrium may not represent the posterior probabilities of the nodes. To resolve this, we introduce the junction tree algorithm, eliminating the loops by constructing a structure known as the clique tree. The junction tree algorithm produces the exact solution, but the time complexity is exponential to the number of nodes in the largest clique. This can make inference intractable for a real world problem, for example, for an Ising model (grid structure Markov Network), the minimum size of the largest clique is n, the dimension of the grid. In this lecture,we discuss our first approximate inference technique - variational algorithm. In particular, we will introduce in detail **Loopy Belief Propagation**, and give a relative simple introduction on **Mean Field Approximation**.

2 Loopy Belief Propagation

The general idea behined Loopy Belief Propagation (LBP) is to run Belief Propagation on a graph containing loops, despite the fact that the presence of loops does not guarantee convergence. Before introducing the theoretical groundings of the methods, we first discuss the algorithm, built on the normal Belief Propagation method. We then introduce a pilot empirical study by Murphy et al. (1999), with a little historical background.

2.1 Message Passing Revisit: Belief Propagation and Belief Update

The procedure of Message Passing is very simple: A node can send a message to its neighbors when and only when it has recieved message from all its other neighbors. The message can be calculated simply by



(a) Message passing

(b) Computing node marginal

Figure 1: Example for Belief Propagation

mutipling the message from all the other nodes, the initial potential of the node itself (singleton), and the edge potential with its neighbour (doubleton). As an example, the message in Figure 1a is defined by

$$m_{i\to j}(x_j) = \sum_{x_i} \left(\psi(x_i)\psi(x_i, x_j) \prod_{k\in Neighbor(i)\setminus j} m_{k\to i}(x_i) \right) \tag{1}$$

The node marginal can be calculated by multiplying all the messages with its own potential (when it receives all the messages from its neighbour),2 as given by:

$$p(x_i) \propto \psi(x_i) \prod_{k\in Neighbour(i)} m_{k\to i}(x_i) \tag{2}$$

The message passing protocol used in the Belief Propagation method (Sum-Product) requires a node to be ready (receive message from all its neighbours) before sending. The Belief Update algorithm, however, allows message to be sent along arbitrary edge. The belief update message is done in the following procedure:

$$\sigma_{i\to j} = \sum_{C_i - S_{i,j}} \beta_i \tag{3}$$

$$\beta_j = \beta_j \frac{\sigma ij}{\mu_{ij}} \tag{4}$$

$$\mu_{ij} = \sigma_{i\to j} \tag{5}$$

The belief β_j of a node j is updated by the message δ_{ij} from its neighbour node i. At every step it is divided by the previous message μ_{ij} between them (which can be considered to be stored on the edge). All μ's are intilized to 1 and all local belief β are initialized to their local potential. The algorithm can then run without the message passing protocol constraint. At each update, the previous update will be eliminated by the division, and hence the algorithm is equivalent to the original Sum-Product scheme.

Similarly, for factor graph, we can compute the message to a factor node a from node i and from a factor node a to node i in the following way:

$$m_{i\to a}(x_i) = \prod_{c\in Neighbour(i)\setminus a} m_{c\to i}(x_i) \tag{6}$$

$$m_{a\to i}(x_i) = \sum_{X_a\setminus x_i} f_a(X_a) \prod_{j\in Neighbour(a)\setminus i} m_{j\to a}(x_j) \tag{7}$$

The belief on the factor nodes are computed by multiplying all the incoming message and the local potentials.

2.2 LBP: The Algorithm

Loopy Belief Propagation (LBP) is running the same message passing algorithm on a loopy graph. It is in essential a fixed point iteration procedure that tries to minimize F_{bethe} (discussed later). The algorithm is to repeat the following procedure until convergence:

$$b_a(x_a) \propto f_a(X_a) \prod_{i \in Neighbour(a)} m_{i \to a}(x_i) \tag{8}$$

$$b_i(x_i) \propto f_i(x_i) \prod_{a \in Neighbour(i)} m_{a \to i}(x_i) \tag{9}$$

$$m_{i \to a}(x_i) = \prod_{c \in Neighbour(i) \setminus a} m_{c \to i}(x_i) \tag{10}$$

$$m_{a \to i}(x_i) = \sum_{X_a \setminus x_i} f_a(X_a) \prod_{j \in Neighbour(a) \setminus i} m_{j \to a}(x_j) \tag{11}$$

Recall that by using the techniques in Belief Update algorithm, a node does not need to wait for all its other neighbours to send it message. The algorithm can then be started easily on a loopy graph. However, it is not guaranteed to converge. As a result, LBP was not popular in the community for a long time. However, a breakthrough in coding theory by Berrou et al. (1993) called Turbo Code, revived the method. Turbo Code performs well in practice and the technique was later shown to be an application of Loopy Belief Propagation. A substantial amount of work has been devoted in understanding its behavior and theoretical background. In several empirical experiments Murphy et al. (1999), showed that LBP can work on many other types of graphs, including large graphs and graphs with a lot of loops. Although there are no convergence guarantees, Murphy et al. (1999) show that good approximations are still achievable if we

1. Stop the algorithm after a fixed number of iteration.
2. Stop when no significant difference in belief update.
3. **When the solution converges, it is usually a good approximation.**

Murphy et al. (1999) observed that small priors and small weights can cause oscillation. However, the actual causes of oscillation versus converge are still being studied. For practioners, Koller et al. (2009) suggest a few skills to make help LBP work better, including specific techniques such as *message scheduling, residual belief propagation* or standard techniques to resolve local maximal, such as *heuristical initialization* and *multiple restarts*.

2.3 LBP: The Bethe Approximation

While running LBP in a loopy graph, there is a possibility that the algorithm runs indefinitely without convergence. However, empirically a good approximation is achieved when it does converge (Murphy et. al 1999) making it a popular algorithm for approximate inference. Running an algorithm which is not even guaranteed to converge may seem like a hack to the first time reader, but in this section we will show that the LBP approximation is mathematically principled.

To understand the theory behind LBP, let us first define the true distribution (P) over a graphical model as:

$$P(X) = \frac{1}{Z} \prod_{f_a \in F} f_a(X_a) \tag{12}$$

where Z is the partition function, F denotes the set of all factors and P is the product of the individual factors in the the factor graph. Since, calculating P is intractable in many cases, we wish to find a distribution Q which approximates P. To measure the distance between Q and P, we can use the information-theoretic measure KL-divergence which is defined as:

$$KL(Q||P) = \sum_X Q(X) \log \frac{Q(X)}{P(X)} \tag{13}$$

Note that the KL metric is asymmetric, is non-negative and has the mininmum value when P=Q. The above equation can be written as:

$$KL(Q||P) = \sum_X Q(X) \log Q(X) - \sum_X Q(X) \log P(X) \tag{14}$$

$$= -H_Q(X) - E_Q \log P(X) \tag{15}$$

If $P(X)$ is replaced with the definition in Eq.(13), we get:

$$KL(Q||P) = -H_Q(X) - E_Q \log (\frac{1}{Z} \prod_{f_a \in F} f_a(X_a)) \tag{16}$$

$$= -H_Q(X) - \log \frac{1}{Z} - \sum_{f_a \in F} E_Q \log f_a(X_a) \tag{17}$$

$$= -H_Q(X) - \sum_{f_a \in F} E_Q \log f_a(X_a) + \log Z \tag{18}$$

From Eq.(18) we can see that $KL(Q||P)$, can be computed without performing inference on P (unknown and intractable to compute in many cases). This would not be the case had we tried to compute $KL(P||Q)$, as the first and second terms on the right hand side of Eq.(18) would have had expectations over $P(X)$ instead of $Q(X)$. We can define a separate quantity for the first two terms in Eq.(18) as:

$$F(P, Q) = -H_Q(X) - \sum_{f_a \in F} E_Q \log f_a(X_a) \tag{19}$$

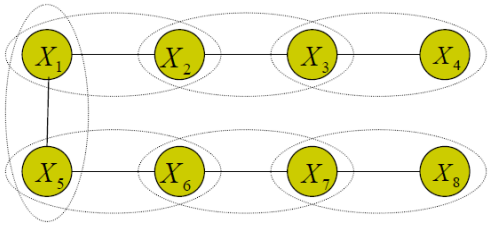
The physicists define $F(P, Q)$ as the "Gibbs free energy". In order to compute $\sum_{f_a \in F} E_Q \log f_a(X_a)$, we need to calculate the expectation (wrt Q) of log of individual factors which is relatively cheap computationally (especially if the scope of the each factor is small). However, computing $H_Q(X)$ requires summation over all possible values. Therefore, computing $F(P, Q)$ is hard in general. One approach to tackle this is to approximate $F(P, Q)$ with an easy to compute $\hat{F}(P, Q)$.

Consider the Markov network shown in Fig.(2a). The factor graph resulting from such a network is essentially a tree. The probability distribution for this tree (or any factor tree in general) can be factored as:

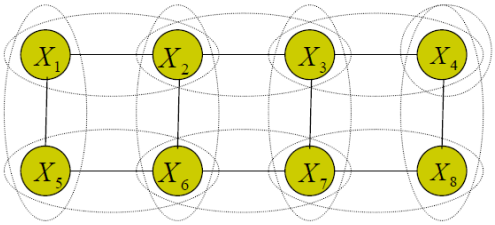
$$b(x) = \prod_a b_a(x_a) \prod_i b_i(x_i)^{1-d_i} \tag{20}$$

where the first product term is over the doubleton factors, the second product term is over the singleton factors and d_i is the number of direct neighbors of a singleton node X_i. The entropy of this distribution is given by:

$$H_{tree} = -\sum_a \sum_{x_a} b_a(x_a) \log b_a(x_a) + \sum_i (d_i - 1) \sum_{x_i} b_i(x_i) \log b_i(x_i) \tag{21}$$



(a) Markov network with tree-structured factor graph



(b) Markov network with a general factor graph

Figure 2: An illustration of tree-structured and general factor graphs

Thus, using Eq.(19) the Gibbs free energy for a tree-structured distribution Q can be written as:

$$F_{tree} = \sum_a \sum_{x_a} b_a(x_a) \log \left(\frac{b_a(x_a)}{f_a(x_a)}\right) + \sum_i (1 - d_i) \sum_{x_i} b_i(x_i) \log b_i(x_i) \tag{22}$$

$$= F_{12} + F_{23} + ... + F_{67} + F_{78} - F_1 - F_5 - F_2 - F_6 - F_3 - F_7 \tag{23}$$

The above equation (Eq. (22)) only involves summation over edges and vertices and is therefore easy to compute. We want to use a similar form to approximate the Gibbs free energy for any general factor graph. Consider a more general Markov network (Fig. (2b)). The factor graph here is not a tree, and the distribution cannot be "exactly" factorized as in Eq.(20). However, we can still "approximate" it to be the same. This approximation is known as the Bethe approximation and the corresponding free energy (known as the Bethe free energy) is given by:

$$H_{Bethe} = -\sum_a \sum_{x_a} b_a(x_a) \log b_a(x_a) + \sum_i (d_i - 1) \sum_{x_i} b_i(x_i) \log b_i(x_i) \tag{24}$$

$$F_{Bethe} = \sum_a \sum_{x_a} b_a(x_a) \log \left(\frac{b_a(x_a)}{f_a(x_a)}\right) + \sum_i (1 - d_i) \sum_{x_i} b_i(x_i) \log b_i(x_i) \tag{25}$$

$$= F_{12} + F_{23} + ... + F_{67} + F_{78} - F_1 - F_5 - 2F_2 - 2F_6 - ... - F_8 \tag{26}$$

The Bethe free energy (F_{Bethe}) is equal to the Gibbs free energy ($F(P, Q)$) for tree-structured graphs, but for general graphs H_{Bethe} is not the same as H_{tree}. In the latter case, we can only approximate $\hat{F}(P, Q)$ to be F_{Bethe}. The advantage of F_{Bethe} is that it is relatively easy to compute. The major disadvantage is that it may or may not be connected to the actual $F(P, Q)$. It could, in general, be greater or smaller than $F(P, Q)$. The form of Eqs.(24) and (25) match the design of a Bethe cluster graph (Fig. 3).

Now we want to minimize F_{Bethe}, with doubleton (b_a's) and singleton(b_i's) potentials as parameters. The

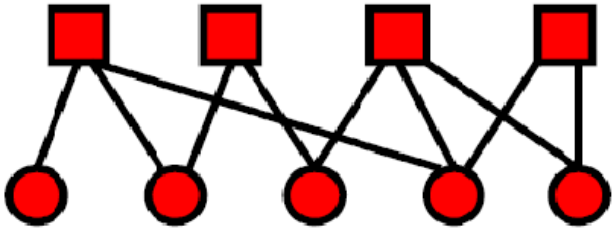


Figure 3: A Bethe cluster graph

optimization problem with constraints can be written in the Lagrangian dual form as:

$$L = F_{Bethe} + \sum_i \gamma_i \left\{ 1 - \sum_{x_i} b_i(x_i) \right\} + \sum_a \sum_{i \in N(a)} \sum_{x_i} \lambda_{ai}(x_i) \left\{ b_i(x_i) - \sum_{X_a \setminus x_i} b_a(X_a) \right\} \tag{27}$$

Setting the derivate with respect to the paramaters to zero:

$$\frac{\partial L}{\partial b_i(x_i)} = 0 \implies \quad b_i(x_i) \propto \exp\big(\frac{1}{d_i - 1} \sum_{a \in N(i)} \lambda_{ai}(x_i)\big) \tag{28}$$

$$\frac{\partial L}{\partial b_a(X_a)} = 0 \implies \quad b_a(X_a) \propto \exp\big(- \log f_a(X_a) + \sum_{i \in N(a)} \lambda_{ai}(x_i)\big) \tag{29}$$

If we set $\lambda_{ai}(x_i) = \log m_{i \to a} = \log \prod_{b \in N(i) \setminus a} m_{b \to i}(x_i)$, we obtain:

$$b_i(x_i) \propto f_i(x_i) \prod_{a \in N(i)} m_{a \to i}(x_i) \tag{30}$$

$$b_a(X_a) \propto f_a(X_a) \prod_{i \in N(a)} \prod_{c \in N(i) \setminus a} m_{c \to i}(x_i) \tag{31}$$

Now, if we use the fact that $m_{a \to i}(x_i) = \sum_{X_a \setminus x_i} b_a(X_a)$, where we are excluding the message $m_{i \to a}$:

$$m_{a \to i}(x_i) = \sum_{X_a \setminus x_i} f_a(X_a) \prod_{j \in N(a) \setminus i} \prod_{b \in N(j) \setminus a} m_{b \to j}(x_j) \tag{32}$$

The above equations are exactly the same as the LBP algorithm updates (Eqs. (8), (9), (10), and (11)). Therefore, the doubleton and singleton potentials in the Bethe optimization problem can be interpreted as beliefs at each iteration of LBP. Morever, the analysis shows that belief propagation on factor graphs is equivalent to minimizing the Bethe energy function. Similarly, the LBP algorithm would converge to true values in case of tree-structured graphs and may only approximate the true values in case of general graphs.

2.4 LBP: The General Theory

From the above discussion, it can be seen that LBP is an approximate inference method that allows us to approximate a distrbution $p(X|\theta)$ over a complex graph which makes computing marginal (or conditional) probability over arbitrary sets of random variables intractable, by another tractable distribution $q(X|\theta)$. The problem can be cast into an optimization problem:

$$q^\star = \arg\min_{q \in S} \{F(p, q)\} \tag{33}$$

As was shown in Section 2.3, we do not need to optimize explicitly for $q(X)$ over the entire space of possibilities (S). We can just focus on the set of doubleton and singleton beliefs $b = \{b_{i,j} = \tau(x_i, x_j), b_i = \tau(x_i)\}$ and relax the optimization objective:

$$b^\star = \arg\min_{b \in M_o} \{F_{Bethe}(p, b)\} \tag{34}$$

where M_o is a relaxed feasible set:

$$M_o = \left\{ \tau \geq 0 \;\mid\; \sum_{x_i} \tau(x_i) = 1, \;\; \sum_{x_i} \tau(x_i, x_j) = \tau(x_j) \right\} \tag{35}$$

LBP is a fixed-point iterative procedure that tries to solve for $b^\star$.

3 Mean Field Approximation

In many situations, the true distribution $p(X|\theta)$ does not factorize and exact inference is diffcult to compute due to intractable summations or integrals. The mean field approximation is a variational approximate inference technique that assumes a class of distributions $q(X|\theta)$ of the fully-factorizable form. That is,

$$q(x_1....x_m) = \prod_i q_i(x_i) \tag{36}$$

More generally, we do need to assume a separate factor for each variable. Factorization into disjoint clusters of all variables $\{C_1, C_2...C_m\}$ is also permitted. That is,

$$q(x_1....x_m) = \prod_{C_i} q_i(X_{C_i}) \tag{37}$$

This is known as the mean field approximation. On one hand, the approximation of $p(X|\theta)$ as fully factored distribution is likely to lose a lot of information in the distribution. On the other hand, this approximation is computatinally attractive, since we can evalute any query on $q(X|\theta)$ as a product over the terms that involve variables in the scope of the query.

The problem now is similar to LBP, that is, finding a fixed-point characterization for the Gibbs free energy $F(P, Q)$. We again have to use the Lagrange multipliers to derive a characterization of the stationary points of $F(P, Q)$. If we assume Q to factorize according to Eq.(36), the optimization problem becomes:

$$\textit{maximize} \qquad F(P,Q) = -H_Q(X) - \sum_{f_a \in F} E_Q \log f_a(x_a) \tag{38}$$

$$\textit{subject to} \qquad q(x_1....x_m) = \prod_i q_i(x_i) \tag{39}$$

$$\textit{and} \qquad \sum_{x_i} q_i(x_i) = 1 \qquad \forall i \tag{40}$$

The two terms in the right hand side of Eq.(38) can be simplified as:

$$H_Q(X) = \sum_i H_Q(X_i) \tag{41}$$

$$E_Q \log f_a(x_a) = \sum_{x_a} \Big(\prod_{X_i \in X_a} q_i(x_i) \Big) \log f_a(x_a) \tag{42}$$

It is evident from the above simplification that the optimization problem can be solved using a coordinate descent procedure in each of the q_i coordinates. Specifically, to optimize for $q_i(x_i)$ we define the Lagrangian

that consists of all terms in $F(P, Q)$ that involve $q_i(x_i)$:

$$L_i(q_i) = -H_Q(X_i) - \sum_{x_a} E_Q \log f_a(x_a) + \lambda \sum_{x_i} q_i(x_i) - 1 \tag{43}$$

where the second term involves summation over all factors f_a whose scope X_a is such that $X_i \in X_a$.

References

Berrou, C., Glavieux, A., and Thitimajshima, P. (1993). Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1. *Proceedings of ICC '93 - IEEE International Conference on Communications*, 2.

Koller, D., Friedman, N., and Kollar, D. (2009). *Probabilistic Graphical Models: Principles and Techniques (Adaptive Computation and Machine Learning series)*, volume 2009.

Murphy, K. P., Weiss, Y., and Jordan, M. I. (1999). Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of Uncertainty in AI*, volume 9, pages 467–475.