CS 683: Advanced Design and Analysis of Algorithms

February 13, 2008
Lecturer: Lukas Kroc

Scribes: Shaomei Wu (sw475@cornell.edu) *

1 Recap: Factor Graph

Let's continue our journey through graphical models. Last time, we defined the problem we try to solve:

1. Given a discreet system with state vectors (configurations) $\vec{x}$, each $x_i \in \mathcal{D}$;
2. Each configuration has a weight associated with it, $weight(\vec{x}) \in \mathcal{R}_0^+$.

We also introduced two graphical models to represent the problem given above: Bayesian Network and Factor Graph. Today's lecture will be based on factor graph.

1.1 Configuration

A factor graph is a bipartite graph consisting of two kinds of nodes:

1. variable nodes (◯): stand for the variable x_i's;
2. factor nodes(□): stand for the functions f_α.

We also define the weight associated with each state as:

$$weight(\vec{x}) := \prod_\alpha f_\alpha(\vec{x_\alpha}) \in \mathcal{R}_0^+$$

For example, the following factor graph is constructed to represent the SAT problem

$$\underbrace{(x \vee y)}_{\alpha} \wedge \underbrace{(\neg x \vee z)}_{\beta}$$

*Many thanks to Lukas for his help at revising the initial scribe of notes.

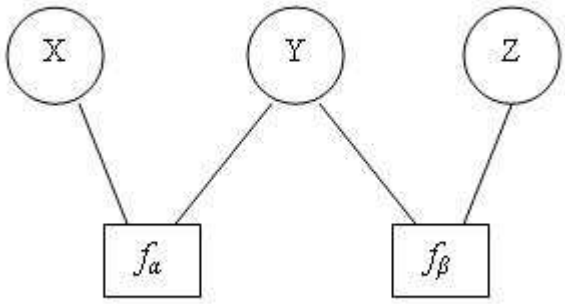


Figure 1: Factor graph for example SAT problem

Also, we define

$$f_\alpha(x,y) = \begin{cases} 1 & \text{if } x \vee y = True; \\ 0 & \text{if } x \vee y = False. \end{cases}$$

$$f_\beta(x,z) = \begin{cases} 1 & \text{if } \neg x \vee z = True; \\ 0 & \text{if } \neg x \vee z = False. \end{cases}$$

1.2 Probabilistic interpretation

To approach the solution of SAT problem with factor graphs, we need to introduce probabilitity into our model. Here we define that:

$$p(\vec{x}) := \Pr(\vec{X} = \vec{x}) = \frac{1}{Z} \prod_\alpha f_\alpha(\vec{x}),$$

where $\vec{x}_\alpha$ is a projection of $\vec{x}$ to only variables that appear in clause α. The normalization constant Z is defined as:

$$Z := \sum_{\vec{x}} \prod_\alpha f_\alpha(\vec{x}_\alpha).$$

The queries we can propose include:

a) Z=? What is the value of Z? By finding out whether Z is zero or not, we would be able to solve SAT problem.

b) Marginal probabilities:

$$p_i(x_i) := \Pr[X_i = x_i] = \sum_{\vec{x}_{-i}} \Pr[\vec{X} = \vec{x}]$$

where the sum is across all possible value combinations of all variables except x_i. For some variable x_i, $p_i(x_i = True)$ represents the fraction of solutions where the variable has value $True$. This information can be used to find a solution: pick a variable that has the highest marginal and set it its preferred way, thus simplifying the problem. Recompute marginals and repeat until all variables have been assigned a value (so called *decimation*).

2 Today: Belief Propagation Algorithm (BP)

Objective: "compute" marginal probabilities AND value of Z for given factor graph. However, as we know, marginals and Z are probably not solvable, since factor graph can represent a very difficult problem such as SAT. In this lecture, we will introduce a method to *approximate* the solution instead of directly solving the problem. Today's lecture is based on Jonathan S. Yedidia, William T. Freeman and Yair Weiss's paper "Constructing Free Energy Approximations and Generalized Belief Propagation Algorithms"(2004)[1].

2.1 Algorithm Framework

To compute $p_i(x_i)$ and Z, our basic intuition is to get rid of the exponential $\sum$ in those formulas. To implement this intuition, the algorithm is design as follows:

A) Design a parameterized[1] family of probability distributions where it is *easy* to answer previous queries. We call such probability distributions $b(\vec{x})$.

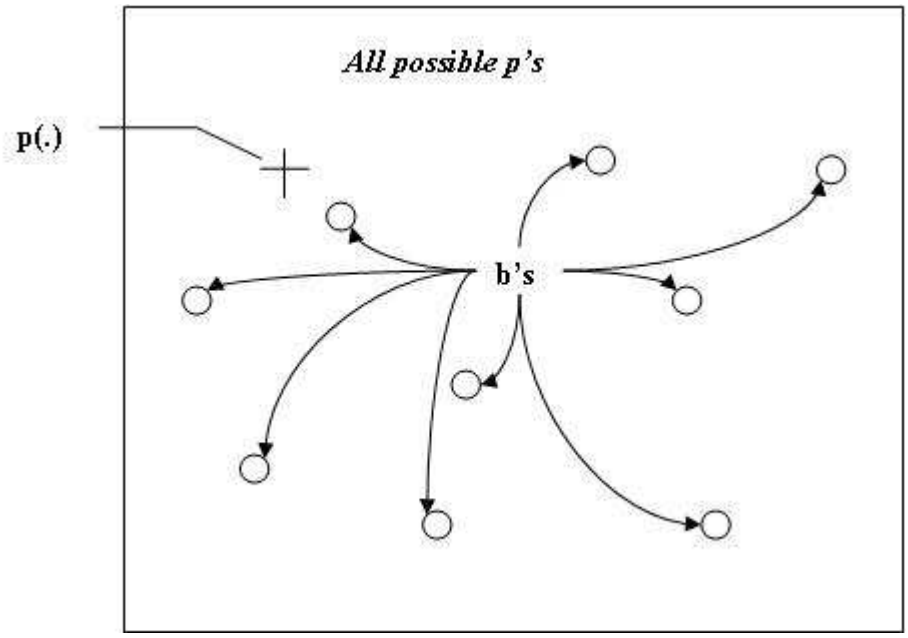
   For example, a family of normal distribution is parameterized with a set of (μ, σ^2) pairs.

B) By tuning $b(\vec{x})$'s parameters, we try to minimize the difference between $p(\cdot)$ and $b(\cdot)$.

C) Use b's answers to queries as estimates of the actual p's answers.

Let's establish the paradigm with an example. In the following figure, we use a box to represent the space of all possible p's. Even though we don't know the exact position of $p(\vec{X} = \vec{x})$ in the space, it is fixed and pre-existing. Given a set of $b(.)$'s, if we can estimate the distance of each $b(.)$ point to the true solution $p(.)$, we will thus be able to compare the distances and find the closest $b(.)$ to $p(.)$.

[1]The number of parameters should be large enough to approximate the optimal solution, but reasonably small to work with.



Although there is no guarantee that we can find a $b(.)$ that is absolutely precise, by carefully defining the distance metric and making a reasonable assumption about the distribution $b(.)$, we can approach the problem in a scalable and practical way.

The important questions left to us are:

a) How to define the distance between $b(.)$ and $p(.)$?

b) How to find the reasonable distribution $b(.)$?

2.2 Brief Propagation Steps

Here we present the technical process to establish the parameters and associated distribution for $b(.)$, and to estimate the distance between $b(.)$ and $p(.)$.

A1: Calculate $b(\vec{x})$. Consider $b(\vec{x})$ such that

$$b(\vec{x}) = \frac{\prod_\alpha b_\alpha(\vec{x}_\alpha)}{\prod_i b_i(x_i)^{d_i-1}}.$$

Here $b(\vec{x})$ is the joint probability. $b_\alpha(\vec{x}_\alpha)$'s and $b_i(x_i)$'s are the marginal probabilities, and d_i is a degree of the variable in the factor graph (e.g. how many times it occurs in some clause). This condition is true for $p(.)$ (and therefore $b(.)$ can exactly correspond to it) when the factor graph is a tree (i.e., no loop exists).

Hence we have $b_\alpha(\vec{x}_\alpha)$'s and $b_i(x_i)$'s as the parameters for $b(\vec{x})$. Since the number of variables appearing in each clause is polynomial in the number of variable, and the possible values for each variable are just 0,1, we can list all the x_i and x_α and assign an initial $b_\alpha(\vec{x}_\alpha)$ and $b_i(x_i)$ with them, as long as the following conditions are fulfilled.

Conditions for parameters:

$$(\star) \begin{cases} \sum_{x_i} b_i(x_i) = 1, & \forall i; \\ \sum_{\vec{x}_\alpha} b_\alpha(\vec{x}_\alpha) = 1, & \forall \alpha; \\ \sum_{\vec{x}_{\alpha - i}} b_{\vec{\alpha}}(\vec{x}_\alpha) = b_i(\omega_i), & \forall i, \forall \alpha \ni i, \forall \omega_i. \end{cases}$$

There is also an implicit assumption that all variable values are in $[0, 1]$. Note that we have to give up satisfying $\sum_{\vec{x}} b(\vec{x}) = 1$, because it is computationally too expensive to add in this list of constraints (the sum is exponential).

B1: Use KL-divergence $D_{KL}(b||p)$ as a difference measure. KL-divergence is defined as:

$$D_{KL}(b||p) = \sum_{\vec{x}} b(\vec{x}) \log \frac{b(\vec{x})}{p(\vec{x})}.$$

Here, $b(\vec{x})$ is already defined in A1, and $p(\vec{x})$ can be computed by

$$p(\vec{x}) = \frac{1}{Z} \prod_\alpha f_\alpha(x_\alpha).$$

The properties of KL-divergence include:

1. $D_{KL}(b||p) \geq 0$;
2. $D_{KL}(b||p) = 0$ iff $b \equiv p$.

We define

$$Dist(b_i(x_i), \cdots, b_\alpha(x_\alpha), \cdots) := D_{KL}(b||p) \tag{1}$$

to be a function measuring difference between $p(.)$ and $b(.)$, parameterized by $b(.)$'s parameters ($p(.)$ is unknown, but fixed).

B2: Massage the formula for $D_{KL}(b||p)$ By rearranging terms in (1), we can rewrite $Dist(b_i(x_i), \cdots, b_\alpha(x_\alpha), \cdots)$ so that the only terms in it are:

$$\text{some\_function\_of}(b_i(x_i), b_\alpha(\vec{x}_\alpha), f_\alpha(\vec{x}_\alpha), d_i) - \log Z.$$

Here $f_\alpha(\vec{x}_\alpha)$, and d_i can be found from the factor graph, $b_i(x_i)$ and $b_\alpha(\vec{x}_\alpha)$ are variables we will use to optimize, with the objective of achieving the minimal value of $D_{KL}(b||p)$. And Z is unknown but constant, so it does not matter for the optimization.

B3: Use method of Lagrange multipliers to solve the optimization problem. We want to solve:

$$\min_{b_i(x_i), b_\alpha(\vec{x}_\alpha)} Dist(b_i(x_i), b_\alpha(\vec{x}_\alpha), f_\alpha(\vec{x}_\alpha), d_i), \text{ subject to } (\star).$$

We will use the method of Lagrange multipliers to turn it into an *unconstrained* problem, and then search for stationary points of $Dist(\cdot)$ s.t. $(\star)$ where $b_i(x_i), b_\alpha(\vec{x}_\alpha)$ are real parameters.

There are different ways to get the optimal, the Brief Propagation algorithm uses the following iteration equations (a lot of derivation is skipped here):

$$n_{i\to\alpha}(x_i) = \prod_{\beta \ni i \setminus \alpha} m_{\beta\to i}(x_i);$$

$$m_{\alpha\to i}(x_i) \propto \sum_{x_{\alpha\setminus i}} f_\alpha(\vec{x}_\alpha) \prod_{j\in\alpha\setminus i} n_{j\to\alpha}(x_j)$$

where $\beta \ni i \setminus \alpha$ are all clauses where variable i appears except for clause α. Here we can see that the values of n's in current iteration depend on the values of m's, while the values of m's depend on the values of n's in previous iterations. Starting at a random initial values for m's and n's, this is how the "messages" get propagated and updated.

C1: Finish Up. In the end, we will get the marginals as:

$$b_i(x_i) \propto \prod_{\alpha \ni i} m_{\alpha\to i}(x_i).$$

The marginals $b_\alpha(\vec{x}_\alpha)$ can be computed in a similar manner. Also, assuming that we have found the optimal solution, we get $D_{KL}(b||p) = 0$. Therefore, the value of Z can be computed by:

$$Z = \exp\Big(\text{some\_function\_of}(b_i(x_i), b_\alpha(\vec{x}_\alpha), f_\alpha(\vec{x}_\alpha), d_i)\Big).$$

We have provided estimates to the queries.

References

[1] J. YEDIDIA, W. FREEMAN, and Y. WEISS. Constructing free energy approximations and generalized belief propagation algorithms. 2004.