IFT 6269: Probabilistic Graphical Models **Fall 2018**

Lecture 13 — October 16

Lecturer: Simon Lacoste-Julien *Scribe: Pravish Sainath*

Proofread and quickly corrected by Simon Lacoste-Julien.

13.1 Inference

13.1.1 Motivation

We have seen about different types of probabilistic graphical models, their properties and how they model probability distributions by encoding the conditional independences. Let us try to find out how we can put these graphical models to use to answer specific questions about their distributions.

In many situations, we want to compute the following probabilities from PGMs :

(1) Marginal $p(x_F)$ for some $F \subseteq V$

(2) Conditional $p(x_F|x_E)$ for *query nodes* $F \subseteq V$ and *evidence nodes* $E \subseteq V$

(3) Partition function (for UGM) (normalization constant)

$$Z = \sum_{x_V} \prod_{C \epsilon \mathcal{C}} \psi_C(x_C)$$

Some situations that require inference

(1) Determining missing data : $p(x_{\text{unobserved}}|x_{\text{observed}})$

Example : Image infilling task in Computer Vision

(2) Prediction : $p(x_{\text{future}}|x_{\text{past}})$

Example : Prediction of next observation in a sequence / time series

(3) Identifying latent cause : $p(x_{\text{cause}}|x_{\text{observation}})$

Example : QMR Model (Quick Medical Reference of diseases - symptoms) - identify presence of a disease from observed symptoms

(4) (Related to inference) Decoding : $\arg\max_{x_F} p(x_F|x_E)$

Example : Speech Recognition - identifying the best sentence from speech data

(5) Inference is also needed sometimes when *estimating parameters*

Example : When doing MLE in a latent variable model, we need to compute $p(z|x)$ during the E-step of EM algorithm.

Remark 13.1.1 *We will present inference algorithms for only UGMs as they are simpler and more general. These can be applied to DGMs after converting them to UGMs using the process of moralization studied in the previous lecture. The joint probability distribution represented by a DGM can be expressed by an equivalent UGM as follows :*

$$p(x) = \prod_i p(x_i|x_{\pi_i})$$

$$\Big\downarrow \textit{moralization}$$

$$p(x) = \frac{1}{Z}\prod_i \psi_{C_i}(x_{C_i})$$

$$\textit{where}$$

$$Z = 1$$

$$C_i \triangleq \{i\} \cup \pi_i$$

$$\psi_{C_i}(x_{C_i}) \triangleq p(x_i|x_{\pi_i})$$

13.1.2 Key Idea for graph eliminate algorithm

The main trick to compute the marginalization efficiently is to re-organize the computation using the *distributivity property* in a specific order (this will yield the graph eliminate inference algorithm that we will describe soon)

Distributivity Property

We use the distributivity property to reorganize sum in the probability expression.

By the distributivity of $\oplus$ over $\odot$, it can be stated for any a, b, c that :

$$c \odot (a \oplus b) = c \odot a \oplus c \odot b$$

For two functions $f\cdot)$ and $g(\cdot)$, we have

$$\sum_{x_1,x_2} f(x_1)g(x_2) = \left(\sum_{x_1} f(x_1)\right)\left(\sum_{x_2} g(x_2)\right)$$

More generally,

$$\boxed{\sum_{x_{1:n}} \prod_i f_i(x_i) = \prod_i \left(\sum_{x_i} f_i(x_i) \right)}$$

Suppose that each variables x_i can take k values. Using this trick, we have transformed a sum of k^n terms, each including the product of n values (and thus $O(k^n \cdot n)$ complexity) to a product of n terms, each which is a sum over k terms, thus a complexity of $O(k \cdot n)$! We now see how to generalize this idea to more complicated potentials.

13.1.3 Graph Elimination Algorithm (for inference)

The graph eliminate algorithm uses the idea of ditributivity to successively eliminate variables (i.e. summing over their values) and infer the marginal probability of the *query*. This is called the *Variable Elimination* (VE) or the *Graph Eliminate* (GE) algorithm.

We present the formal procedure for the Graph Eliminate (GE) algorithm to compute the marginal probability $p(x_F)$ of the given *query* corresponding to the set of nodes F from the UGM G with set of cliques $\mathcal{C}$.

Initialize

(a) Choose an *elimination ordering* such that the nodes in F are the last nodes.

(b) Put all the terms $\psi_C(x_C)$ in an *active list*

Update

(c) Repeat in the order of variables to eliminate :

Pick the variable x_i to eliminate from the *active list*.

(1) Remove all factors from active list that contains x_i as argument and take their product.

$$i.e. \prod_{\alpha \text{ s.t. } i \in \alpha} \psi_\alpha(x_\alpha)$$

(2) Sum the product over the variable x_i to get a new factor $m_i(x_{S_i})$ where S_i contains all the variables in the factors except i

$$i.e.\ m_i(x_{S_i}) \triangleq \sum_{x_i} \underbrace{\prod_\alpha \psi_\alpha(x_\alpha)}_{\text{new clique to sum over}}$$

$$S_i \triangleq \left(\bigcup_{\alpha \text{ s.t. } i \in \alpha} \alpha \right) \setminus \{i\}$$

(3) Put back $m_i(x_{S_i})$ in the *active list* (call it $\psi_{S_i}(x_{S_i})$ for consistency of notation).

Normalize

(d) Last factors left have only x_F terms.
The required probability $p(x_F)$ is proportional to this and needs to be normalized to obtain the final value.

13.1.4 Illustrating example

We want to compute the probability distribution $p(x_4)$ from the UGM whose graph is given in Figure 13.1.

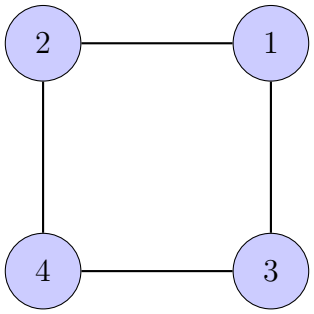

Figure 13.1: Graph G

Writing the joint distribution factorized by the UGM,

$$p(x_1, x_2, x_3, x_4) = \frac{1}{Z}\psi(x_1, x_2).\psi(x_1, x_3).\psi(x_3, x_4).\psi(x_2, x_4)$$

The required probability $p(x_4)$ can be expressed as a marginal of the joint probability by summing over the remaining variables,

$$\implies p(x_4) = \frac{1}{Z}\sum_{x_1,x_2,x_3} \psi(x_1, x_2).\psi(x_1, x_3).\psi(x_3, x_4).\psi(x_2, x_4)$$

$$\implies p(x_4) = \frac{1}{Z}\sum_{x_1}\sum_{x_2}\sum_{x_3} \psi(x_1, x_2).\psi(x_1, x_3).\psi(x_3, x_4).\psi(x_2, x_4)$$

Splitting the summation by the distributive property,

$$\implies p(x_4) = \frac{1}{Z}\sum_{x_3} \psi(x_4, x_3) \sum_{x_2} \psi(x_2, x_4) \sum_{x_1} \psi(x_1, x_2).\psi(x_1, x_3)$$

Let us choose an elimination ordering : $1 \to 2 \to 3 \to 4$

All the factors ψ are currently in the *active list*.

Successively applying the updates in the chosen order, m messages are added to the *active list*, removing the factors containing the eliminated variables.

$$\implies p(x_4) = \frac{1}{Z}\sum_{x_3}\psi(x_4,x_3)\sum_{x_2}\psi(x_2,x_4)\underbrace{\sum_{x_1}\psi(x_1,x_2).\psi(x_1,x_3)}_{m_1(x_2,x_3)}$$

$$\implies p(x_4) = \frac{1}{Z}\sum_{x_3}\psi(x_4,x_3)\sum_{x_2}\psi(x_2,x_4).m_1(x_2,x_3)$$

$$\implies p(x_4) = \frac{1}{Z}\sum_{x_3}\psi(x_4,x_3)\underbrace{\sum_{x_2}\psi(x_2,x_4).m_1(x_2,x_3)}_{m_2(x_3,x_4)}$$

$$\implies p(x_4) = \frac{1}{Z}\sum_{x_3}\psi(x_4,x_3).m_2(x_3,x_4)$$

$$\implies p(x_4) = \frac{1}{Z}\underbrace{\sum_{x_3}\psi(x_4,x_3)m_2(x_3,x_4)}_{m_3(x_4)}$$

$$\implies \boxed{p(x_4) = \frac{1}{Z}m_3(x_4)}$$

As $p(x_4)$ is a probability distribution and it is proportional to the message $m_3(x_4)$, Z can be computed as :

$$Z = \sum_{x_4} m_3(x_4)$$

The GE algorithm modifies the original graph by consecutively removing nodes and passing messages to the other nodes that lead up to the *query* node 4 as shown in Figure 13.2

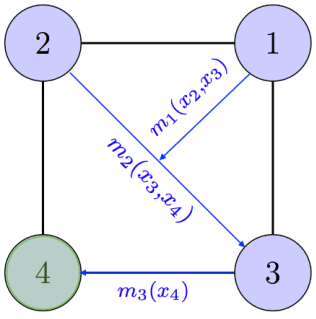

Figure 13.2: Graph G with the computed messages

13.1.5 Properties of the Graph Eliminate Algorithm

Memory Cost

(Suppose for simplicity that each x_i can take 2 values (i.e. $k = 2$)) The memory cost can be expressed in terms of the number of active variables at each stage S_i and the number of factors in the *active list* :

$$\approx 2^{\max_i |S_i|} \cdot (\#\text{factors})$$

Computational Cost

It can be expressed in terms of the number of active variables at each stage S_i and the number of nodes n in the graph :

$$\approx 2^{\max_i |S_i|+1} \cdot n$$

Augmented Graph is Triangulated!

It can be observed that new *cliques* are formed as side effects while running the GE algorithm. Running the algorithm, keeping track of all the edges added in between yields an *augmented graph* that has the property of being a *triangulated graph*.

Figure 13.3: **Left**: Non-triangulated graph **Right**: Triangulated graph

A *chord* is an edge between two non-neighboring nodes in a cycle. **Definition:** A *triangulated graph* is a graph with no cycle of size 4 or more that cannot be broken by a *chord*. In other words, any cycle of size or 4 can be broken by a chord in a triangulated graph, as illustrated in Figure 13.3.

During the graph eliminate algorithm, new edges are added, and it turns out that enough edges are added to ensure that the resulting augmented graph is triangulated. See an example in Figure 13.4. Here, the black lines indicate the original edges and the blue lines indicate the edges introduced by the GE algorithm during elimination.

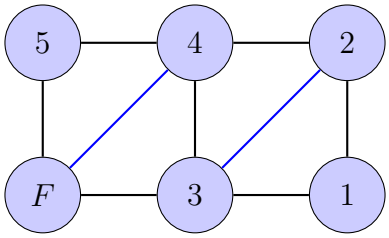

Figure 13.4: Augmented graph after Graph Eliminate

Treewidth of a graph

For an undirected graph G, its *treewidth* is defined as :

$$\boxed{\text{treewidth} \triangleq \min_{\text{over all elimination orderings}} \{\text{size of biggest clique} - 1\}}$$

The "minus one" convention is so that the treewidth of a tree is 1 :

$$\text{treewidth(tree)} = 1$$

- Both memory and running time of the GE algorithm are determined by the number of variables in the largest elimination clique i.e. the term $2^{(\text{size of biggest clique}+1)}$

  For the GE algorithm to be tractable, we need to achieve an ordering giving minimum size of the largest clique which is the *treewidth*

  $\therefore$ Best ordering gives the term $\approx 2^{(treewidth+2)}$ in the complexities.

- Not all orderings are good.

  Example :
  Removing the central node in the $(n+1)$-node star graph gives a large clique of size n leading to a very big factor in the *active list* which is not computationally efficient as seen in Figure 13.5.

  Whereas, removing the leaf nodes gives cliques of size 2, consistent with its *treewidth* of 1.

Bad News about inference in UGM

(a) It is actually NP-hard to compute the *treewidth* of a graph (or to find the best elimination ordering).

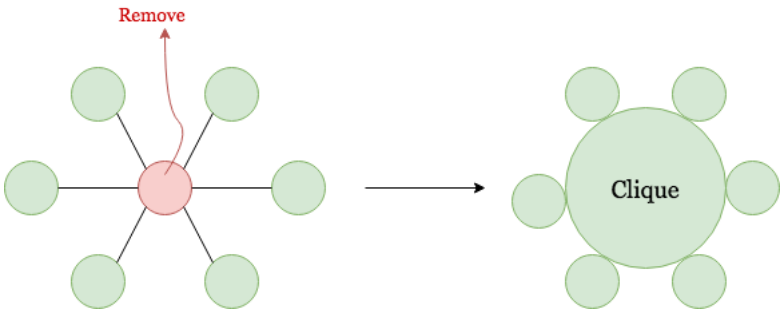

Figure 13.5: Bad ordering in a star graph

(b) It is NP-hard to do *exact* inference in general.
We thus instead need to use *approximate* inference methods in general.

Example : The treewidth of a *grid* graph with $|V|$ nodes is actually growing with the side of the grid $\approx \sqrt{|V|}$ shown in Figure 13.6. We'll see later that Ising models are popular models in computer vision, and they often have this grid structure. In later lectures, we will show how to do approximate inference in such UGM using Gibbs sampling or a variational method (mean field). These terms will be defined in later lectures.

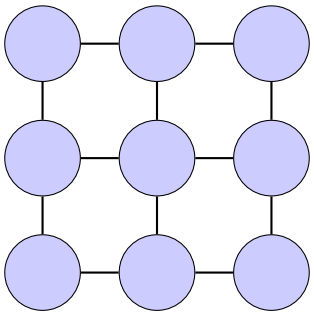

Figure 13.6: Grid graph with $|V|$ vertices

Good News about inference in UGM

(a) Inference in linear time $(|V| + |E|)$ for graphs that are trees *(treewidth = 1)*.

Sum-Product algorithm can be derived for trees like Hidden Markov Models(HMM) and Markov chains.

(b) Efficient for *small treewidth graphs*.

For general graphs, *Junction Tree algorithm* is used.