Exact Inference in Graphical Models

Hoang M. Le

CS 159

(with materials from lectures by Stefano Ermon, Zoubin Gharamani, Bert Huang)
Recap: Graphical Model Representation
Bayesian Networks

Directed Acyclic Graph, a.k.a Belief Networks

\[ p(l, g, i, d, s) = p(l|g)p(g|i, d)p(i)p(d)p(s|i) \]
Markov Random Fields

- Example: voting preference in social networks

Assign scores to each assignment to these variables and then define a probability as a normalized score.

$$\tilde{p}(A, B, C, D) = \phi(A, B)\phi(B, C)\phi(C, D)\phi(D, A),$$

$$\phi(X, Y) = \begin{cases} 
10 & \text{if } X = Y = 1 \\
5 & \text{if } X = Y = 0 \\
1 & \text{otherwise.} 
\end{cases}$$

$$p(A, B, C, D) = \frac{1}{Z} \tilde{p}(A, B, C, D),$$

where

$$Z = \sum_{A,B,C,D} \tilde{p}(A, B, C, D)$$
Markov Random Fields

A MRF is a probability distribution $p$ over variables $x_1, \ldots, x_n$ defined by an undirected graph $G$ with nodes correspond to $x_i$

$$p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(x_c),$$

where $C$ is set of cliques

$$Z = \sum_{x_1, \ldots, x_n} \prod_{c \in C} \phi_c(x_c)$$
Factor Graph Representation

\[ p(A, B, C, D) \propto \phi_1(A, B)\phi_2(B, C)\phi_3(C, D)\phi_4(D, A) \]
Factor Graph Representation

...instead of
Probabilistic Inference

- Let \((E,F)\) be disjoint subset of node indices of a graphical model. Two basic kinds of inference problems:
  - **Marginal probability inference**
    \[
    p(x_E) = \sum_{x_F} p(x_E, x_F)
    \]
  - **Maximum a posteriori (MAP) inference**
    \[
    p^*(x_E) = \max_{x_F} p(x_E, x_F)
    \]

- Conditional probability \(p(x_F|x_E)\)
  \[
  p(x_F|x_E) = \frac{p(x_E, x_F)}{\sum_{x_F} p(x_E, x_F)}
  \]
- Combining conditioning and marginalization
  \[
  p(x_F|x_E) = \frac{p(x_E, x_F)}{\sum_{x_F} p(x_E, x_F)} = \frac{\sum_{x_H} p(x_E, x_F, x_H)}{\sum_{x_F} \sum_{x_H} p(x_E, x_F, x_H)}
  \]
Inference + Learning
Inference + Learning
Roadmap

- Overview of Graphical Model, Inference and Learning, Structured Predictions
- Conditional Random Field and applications
- Graphical Model Inference
  - Exact Inference: Message Passing
  - Approximate Inference: LP Relaxation
  - More approximate inference: Sampling-based methods, Variational Inference
- Graphical Model Learning: Parameter learning & structure learning
- Structured Predictions: Structured SVMs, random forests, deep structured models
- Advanced topics that combine inference and learning
Roadmap

- Overview of Graphical Model, Inference and Learning, Structured Predictions
- Conditional Random Field and applications

- **Graphical Model Inference**
  - **Exact Inference: Message Passing, graph cuts**
  - **Approximate Inference: LP Relaxation**
    - More approximate inference: Sampling-based methods, Variational Inference

- Graphical Model Learning: Parameter learning & structure learning
- Structured Predictions: Structured SVMs, random forests, deep structured models
- Advanced topics that combine inference and learning
Roadmap

• Overview of Graphical Model, Inference and Learning, Structured Predictions

• Conditional Random Field and applications

• Graphical Model Inference
  • Exact Inference: Message Passing
  • Approximate Inference: LP Relaxation
  • More approximate inference: Sampling-based methods, Variational Inference

• Graphical Model Learning: Parameter learning & structure learning

• Structured Predictions: Structured SVMs, random forests, deep structured models

• Advanced topics that combine inference and learning
Exact Inference: Message Passing
Variable Elimination

\[ p(x_1, \ldots, x_n) = p(x_1) \prod_{i=2}^{n} p(x_i \mid x_{i-1}). \]

Inference Goal: calculate \( p(x_n) \)

Naive Method:
\[
p(x_n) = \sum_{x_1} \cdots \sum_{x_{n-1}} p(x_1, \ldots, x_n).
\]
Variable Elimination

\[
p(x_1, \ldots, x_n) = p(x_1) \prod_{i=2}^{n} p(x_i | x_{i-1}).
\]

Inference Goal: calculate \( p(x_n) \)

More efficient method:

\[
p(x_n) = \sum_{x_1} \cdots \sum_{x_{n-1}} p(x_1) \prod_{i=2}^{n} p(x_i | x_{i-1})
= \sum_{x_{n-1}} p(x_n | x_{n-1}) \sum_{x_{n-2}} p(x_{n-1} | x_{n-2}) \cdots \sum_{x_1} p(x_2 | x_1) p(x_1)
\]

\[
p(x_n) = \sum_{x_{n-1}} p(x_n | x_{n-1}) \sum_{x_{n-2}} p(x_{n-1} | x_{n-2}) \cdots \sum_{x_2} p(x_3 | x_2) m_{12}(x_2).
\]
Variable Elimination

Inference Goal: calculate $p(x_n)$

More efficient method:

$$p(x_n) = \sum_{x_{n-1}} p(x_n \mid x_{n-1}) \sum_{x_{n-2}} p(x_{n-1} \mid x_{n-2}) \cdots \sum_{x_2} p(x_3 \mid x_2)m_{12}(x_2)$$

$$= \ldots$$

$$= m_{(n-1)n}(x_n)$$
Variable Elimination

Inference Goal: calculate $p(x_5)$

$$p(x) = \frac{1}{Z} \phi_{12}(x_1, x_2) \phi_{23}(x_2, x_3) \phi_{34}(x_3, x_4) \phi_{35}(x_3, x_5)$$

Elimination order: (1, 2, 4, 3)
Variable Elimination

\[
p(x_5) = \frac{1}{Z} \sum_{x_3} \phi_{35}(x_3, x_5) \sum_{x_4} \phi_{34}(x_3, x_4) \sum_{x_2} \phi_{23}(x_2, x_3) \sum_{x_1} \phi_{12}(x_1, x_2)
\]

\[
= \frac{1}{Z} \sum_{x_3} \phi_{35}(x_3, x_5) \sum_{x_4} \phi_{34}(x_3, x_4) \sum_{x_2} \phi_{23}(x_2, x_3) m_{12}(x_2)
\]

\[
= \frac{1}{Z} \sum_{x_3} \phi_{35}(x_3, x_5) \sum_{x_4} \phi_{34}(x_3, x_4) m_{23}(x_3)
\]

\[
= \frac{1}{Z} \sum_{x_3} \phi_{35}(x_3, x_5) m_{23}(x_3) \sum_{x_4} \phi_{34}(x_3, x_4) m_{43}(x_3)
\]

\[
= \frac{1}{Z} \sum_{x_3} \phi_{35}(x_3, x_5) m_{23}(x_3) m_{43}(x_3)
\]

\[
= \frac{1}{Z} m_{35}(x_5)
\]
Variable Elimination Algorithm

1. Place all potentials $\phi_C(x_C)$ on the active list

2. choose an ordering I of the indices (1, 2, 4, 3)

3. for each $x_i$ in I
   
   A. find all potentials on the active list that reference $x_i$ and remove them from the active list
   
   B. define a new potential as the sum (w.r.t $x_i$) of the product of these potentials
   
   C. place the new potential on the active list

4. return the product of the remaining potentials
Variable Elimination

- Two key operations for factors: **product** and **sum** (marginalization)

- Orderings: matters a lot. NP hard to find the best. Common heuristics: min-neighbors, min-weight, min-fill

- Running time: $O(nd^M)$, where $M$ is the max size of any factor during elimination

- Introducing evidence: $P(Y|E=e) = \frac{P(Y, E = e)}{P(E = e)}$ where $P(X, Y, E)$ is the joint, query variables $Y$, observed $E$, and unobserved variables $X$

- perform variable elimination once on $P(Y,E=e)$ and once more on $P(E=e)$
Variable Elimination as Message Passing

Inference Goal: calculate $p(x_5)$ and $p(x_3)$

$$p(x) = \frac{1}{Z} \phi_{12}(x_1, x_2) \phi_{23}(x_2, x_3) \phi_{34}(x_3, x_4) \phi_{35}(x_3, x_5)$$

Elimination order $p(x_5)$: (1,2,4,3)
Elimination order $p(x_3)$: (1,2,5,4)
Variable Elimination as Message Passing

Inference Goal: calculate $p(x_5)$ and $p(x_3)$

$$p(x) = \frac{1}{Z} \phi_{12}(x_1, x_2) \phi_{23}(x_2, x_3) \phi_{34}(x_3, x_4) \phi_{35}(x_3, x_5)$$

Elimination order $p(x_5)$: (1,2,4,3)
Elimination order $p(x_4)$: (1,2,5,4)

Problem: Naive method wasteful and computationally burdensome

Solution: Reuse intermediate terms efficiently!
Message Passing on Trees

\[
\begin{align*}
  m_{23}(x_3) & \quad \text{from } x_2 \\
  m_{43}(x_3) & \quad \text{from } x_4 \\
  m_{53}(x_3) & \quad \text{from } x_5
\end{align*}
\]
Message Passing on Trees

At each step, eliminate $x_j$ by

$$f_k(x_k) = \sum_{x_j} \phi(x_k, x_j) f_j(x_j)$$

where $x_k$ is parent of $x_j$ in the tree

$f_j(x_j)$ is a message that $x_j$ sends to $x_k$ to summarize all it knows about its children
Message Passing on Trees

Now compute $p(x_k)$ as well:

**Key insight:**
messages $x_k$ will receive from $x_j$
will the same as when $x_i$ was the root

How do we compute all the message we need?

**Solution:**
A node $x_i$ sends a message to a neighbor $x_j$ whenever it has received messages from all nodes beside $x_j
Message Passing on Trees

(collect messages to root)
(if the graph is a tree, any node can be a root)
Message Passing on Trees

collect messages to root
distribute back to leaves
Message Passing on Trees

collect messages to root
distribute back to leaves
Forward-Backward in HMM

collect
Forward-Backward in HMM
Inference in Hidden Markov Models and Linear Gaussian state-space models

\[
p(X_1, \ldots, T, Y_1, \ldots, T) = p(X_1)p(Y_1|X_1) \prod_{t=2}^{T} [p(X_t|X_{t-1})p(Y_t|X_t)]
\]

- In HMMs, the states \( X_t \) are discrete.
- In linear Gaussian SSMs, the states are real Gaussian vectors.
- Both HMMs and SSMs can be represented as singly connected DAGs.
- The forward–backward algorithm in hidden Markov models (HMMs), and the Kalman smoothing algorithm in SSMs are both instances of belief propagation / factor graph propagation.
Loopy Belief Propagation

\[ b_t(x_t) \propto \prod_{s \in \text{neighbors}(t)} m_{t \to s}(x_s) \]

\[ m_{s \to t}(x_t) := \sum_{x_s} \left( \phi_{st}(x_s, x_t) \prod_{u \in \text{neighbors}(s) \setminus t} m_{u \to s}(x_s) \right) \]
**Loopy Belief Propagation**

\[
m_{s\rightarrow t}(x_t) := \sum_{x_s} \left( \phi_{st}(x_s, x_t) \prod_{u \in \text{neighbors}(s) \setminus t} m_{u\rightarrow s}(x_s) \right)
\]

\[
b_t(x_t) \propto \prod_{s \in \text{neighbors}(t)} m_{t\rightarrow s}(x_s)
\]
Loopy Belief Propagation

\[ b_t(x_t) \propto \prod_{s \in \text{neighbors}(t)} m_{t \rightarrow s}(x_s) \]

\[
m_{s \rightarrow t}(x_t) := \sum_{x_s} \left( \phi_{st}(x_s, x_t) \prod_{u \in \text{neighbors}(s) \setminus t} m_{u \rightarrow s}(x_s) \right) \]

\[
m_{B \rightarrow D}(x_D) = \sum_{x_B} \phi(x_B, x_C) \times m_{A \rightarrow B}(x_B) \times m_{C \rightarrow B}(x_B) \]
Loopy Belief Propagation

\[ b_t(x_t) \propto \prod_{s \in \text{neighbors}(t)} m_{t \rightarrow s}(x_s) \]

\[ b_B(x_B) \propto (m_{A \rightarrow B}(x_B))(m_{C \rightarrow B}(x_B))(m_{D \rightarrow B}(x_B)) \]

\[ m_{s \rightarrow t}(x_t) := \sum_{x_s} \left( \phi_{st}(x_s, x_t) \prod_{u \in \text{neighbors}(s) \setminus t} m_{u \rightarrow s}(x_s) \right) \]

\[ m_{B \rightarrow D}(x_D) = \sum_{x_B} \phi(x_B, x_D) \times m_{A \rightarrow B}(x_B) \times m_{C \rightarrow B}(x_B) \]
Sum-Product Message Passing

- While there is node $x_s$ ready to transmit to $x_t$, send the message

$$m_{s \rightarrow t}(x_t) := \sum_{x_s} \left( \phi(x_s) \phi(x_s, x_t) \prod_{u \in \mathcal{N}(s) - t} m_{u \rightarrow s}(x_s) \right)$$

- After computing all messages, any marginal query can be computed in $O(1)$:

$$p(x_t) \equiv b_t(x_t) \propto \prod_{s \in \mathcal{N}(t)} m_{t \rightarrow s}(x_s)$$
Sum-Product Messages

- Function of receiving node’s variable
- Vectors for discrete variables
- Should be normalized (by taking log)
- Alternate form explicitly encodes **unary** and **pairwise** potentials

\[
p(x) \propto \prod_{s \in \mathcal{V}} \phi_s(x_s) \prod_{(s,t) \in \mathcal{E}} \phi_{st}(x_s, x_t)
\]
Max-Product Message Passing

Consider MAP query

$$\max_{x_1, \ldots, x_n} p(x_1, \ldots, x_n)$$

Chain MRF, partition function

$$Z = \sum_{x_1} \cdots \sum_{x_n} \phi(x_1) \prod_{i=2}^{n} \phi(x_i, x_{i-1})$$

$$= \sum_{x_n} \sum_{x_{n-1}} \phi(x_n, x_{n-1}) \sum_{x_{n-2}} \phi(x_{n-1}, x_{n-2}) \cdots \sum_{x_1} \phi(x_2, x_1) \phi(x_1).$$

To compute the mode of \( \tilde{p}(x_1, \ldots, x_n) \)

$$\tilde{p}^* = \max_{x_1} \cdots \max_{x_n} \phi(x_1) \prod_{i=2}^{n} \phi(x_i, x_{i-1})$$

$$= \max_{x_n} \max_{x_{n-1}} \phi(x_n, x_{n-1}) \max_{x_{n-2}} \phi(x_{n-1}, x_{n-2}) \cdots \max_{x_1} \phi(x_2, x_1) \phi(x_1).$$
Message Passing

- a.k.a Belief Propagation (BP)
- Trees: Guaranteed to converge to marginals
- Trees: Takes just one update per message
- Guarantees not as nice in non-trees
  - only provably converges on trees and on graphs with at most one cycle
  - general graph: treat loopy BP as approximate inference
Junction Tree Algorithm

Main idea: Turn graph into a tree of clusters that are amenable to the variable elimination algorithm

A junction tree \( T = (C, E_T) \) over \( G = (\mathcal{X}, E_G) \) is a tree whose nodes \( c \in C \) are associated with subsets \( x_c \in \mathcal{X} \) of the graph vertices (i.e. sets of variables); the junction tree must satisfy the following properties:

Family preservation: For each factor \( \phi \), there exists a cluster \( c \) such that \( \text{Scope}[\phi] \subset x_c \)

Running intersection: for every pair of clusters \( c^{(i)}, c^{(j)} \), every cluster on the path between \( c^{(i)}, c^{(j)} \) contains \( x^{(i)}_c \cap x^{(j)}_c \)
Redefine potential $\psi_c(x_c) = \text{product of all the factors } \phi \text{ in } G \text{ that have been assigned to } c$

$$p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_{c \in C} \psi_c(x_c).$$
Message Passing between Clique Potentials

- Choose a pair of adjacent clusters $c(i), c(j)$ in $T$ and compute a message whose scope is the sep-set $S_{ij}$ between the two clusters

$$m_{i \rightarrow j}(S_{ij}) = \sum_{x_c \setminus S_{ij}} \psi(x_c) \prod_{\ell \in N(i) \setminus j} m_{\ell \rightarrow i}(S_{\ell i})$$

- Belief computation: define the belief of each cluster based on all the messages it receives

$$\beta_c(x_c) = \psi(x_c) \prod_{\ell \in N(i)} m_{\ell \rightarrow i}(S_{\ell i})$$

- Finally marginalizing out the variables in its belief

$$\tilde{p}(x) \propto \sum_{x_c \setminus x} \beta_c(x_c)$$
Finding A Good Junction Tree

• Again NP-Hard

• By hand: Typically, our models will have a very regular structure, for which there will be an obvious solution. For example, very often our model is a grid, in which case clusters will be associated with pairs of adjacent rows (or columns) in the grid

• Using variable elimination: One can show that running the VE elimination algorithm implicitly generates a junction tree over the variables. Thus it is possible to use the heuristics we previously discussed to define this ordering