688 Lecture 8: Exact Inference by Message Passing

Benjamin M. Marlin

Department of Computer Science
University of Massachusetts Amherst

The Complexity of Inference

- The conditioning stage of inference in Markov networks is actually easy. It can be accomplished from first principles or using the factor reduction algorithm.
- The hard part of inference is marginalizing out all of variables that aren't involved in the query and computing the partition function.
- Both of these operations involve summing over products of factors, which has exponential worst case complexity.
- Distributing sums into products and caching intermediate results is the basis for efficient exact inference methods.

Variable Elimination

- The variable elimination algorithm is nothing more than a formalization of the simple idea of distributing sums into products.
- It ensures that all factors involving a variable to be summed over are correctly grouped together.
- It introduces intermediate factors for caching the results of factor products and factor sums.

Variable Elimination Algorithm

Given D variables $X_1, ..., X_D$, we assign the L variables we want to sum over to $V_1, ..., V_L$. VE sums out the L variables and returns a joint factor over the remaining $D - L$ variables.

$SumProductVE(\{V\}_{1:L}, \{\phi\}_{1:K})$
$F_0 \leftarrow \{\phi_1, ..., \phi_K\}$ *//Initial factors from the model*
$F \leftarrow F_0$ *//Initialize working set of factors*
for (i From 1 to L) **do**
  $F_i \leftarrow \{\phi | \phi \in F, V_i \in Scope(\phi)\}$ *//Get working set factors that involve V_i*
  $F_i' \leftarrow F / F_i$ *//Get factors in working set that don't involve V_i*
  $\psi_i \leftarrow \prod_{\phi \in F_i} \phi$ *//Compute product of working set factors that involve V_i*
  $\tau_i \leftarrow \sum_{v \in Val(V_i)} \psi_i$ *//Sum V_i out of the factor product.*
  $F \leftarrow F_i' \bigcup \{\tau_i\}$ *//Update the working set.*
end for
return F *//Return collection of factors over remaining variables.*

Using Variable Elimination

We can use the variable elimination algorithm to sum out any given set of variables. It allows us to compute arbitrary marginals, and, together with factor reduction, answer arbitrary probability queries.

Let's look at some examples.

Variable Elimination Limitations

- The drawback of variable elimination is that the computation is redundant when we use it to compute many marginals.
- A common problem is to compute all single variable marginal distributions as well as all pairwise marginals for variables connected by an edge in the graph.
- We can do much better than variable elimination using clique trees and the sum-product message passing algorithm.

Clique Trees

Given a Markov network $\mathcal{G}$ with variables $Y_1, ..., Y_N$ and factors $\phi_1, ..., \phi_k$, a clique tree is a graph-based data structure satisfying the following properties:

- Clique Factors: The clique tree contains a set of factors ω_i that each consist of a product of a distinct subset of factors in $\mathcal{G}$. Each factor ω_i is associated with a node C_i in the clique tree.
- Family Preservation: Each node C_i in the clique tree represents a set of variables corresponding to the union of the scopes of the factors that appear in ω_i. The nodes are connected to form a valid tree.
- Running Intersection: If a variable appears in cliques C_i and C_k, then it appears in every clique C_j on the unique path between C_i and C_k in the tree.

Clique Trees

- The edges between each pair of neighboring cliques C_i and C_j are labeled with the set of variables S_{ij} that appear in both cliques: $S_{ij} = C_i \cap C_j$.
- The sets of variables S_{ij} are referred to as separator sets or sepsets.
- A valid clique tree can be extracted for any graph $\mathcal{G}$ from a run of the variable elimination algorithm. We'll see how later. For now, we'll focus on clique trees for chain-structured graphs.

Clique Trees For Chains

- Constructing clique trees for chain-structured markov networks is particularly easy.
- We simply place each pair of adjacent variables in the chain into a clique.
- Each clique factor consists of the pairwise factor defined over the pair of variables.
- Single-variable factors can be placed in any clique that contains that variable.
- Cliques are connected when the variables they contain overlap.
- Let's look at an example...

Sum-Product Message Passing

- The sum-product message passing algorithm uses a clique tree data structure to perform exact inference in Markov networks.
- It sums out all the variables except those within a given clique, producing a factor called the *beliefs* from which we can compute arbitrary probabilities.
- The amazing part is that it does this computation **simultaneously for all cliques** by caching intermediate factors in the clique tree data structure.

Sum-Product Message Passing: Intuitions

- The sum-product message passing algorithm computes local intermediate factors called *messages* that are passed between each clique C_i in the tree and each of its neighbors C_j. The set of neighbors of C_i is Nb_i.
- Roughly, the message sent from C_i to C_j represents the information that C_i contains and has received about the distribution over the variables in the separator set S_{ij}.
- The message passing algorithm itself is asynchronous. It relies on a *ready-to-send* criteria. C_i is said to be ready to send its message to C_j when C_i has received incoming messages from all nodes other than j and has not yet sent a message to j.

Sum-Product Message Passing: Algorithm

$SumProductMP(\{C\}_{1:K}, \{S\}_{1:K-1}, \{\omega\}_{1:K})$

while (A clique i is ready-to-send to j) **do**

$$\delta_{i \to j}(S_{ij}) \leftarrow \sum_{C_i / S_{ij}} \omega_i(C_i) \prod_{k \in Nb_i / \{j\}} \delta_{k \to i}(S_{ki})$$

end while

for k from 1 to K **do**

$$\beta_i(C_i) = \omega_i(C_i) \prod_{k \in Nb_i} \delta_{k \to i}(S_{ki})$$

end for

return $\{\beta\}_{1:K}$

Sum-Product Message Passing: Scheduling

- While the algorithm is defined asynchronously, its not difficult to devise a specific order in which to transmit the messages.
- We need only pick any clique as the root of the tree. We pass messages from the leaves of the tree to the root of the tree and then pass messages in the opposite direction from the root to the leaves.
- For a chain, a simple schedule where the messages are first passed from right to left, and then from left to right (or vice versa) can be used.
- Let's look at an example...

Sum-Product Message Passing: Numerical Accuracy

- When the Markov network factor values are very small or very large, the messages and beliefs can potentially underflow or overflow since they can involve products of many small or large terms.
- The most robust alternative is to manipulate the factors in log space.
- All of the factors are converted into log factors (potentials).
- The factor products are replaced by sums of potentials.
- The factor sums must be performed using a general technique called the log-sum-exp algorithm.

Log-sum-exp Algorithm

- Given a set of K values $a_1, ..., a_K$, the log-sum-exp algorithm computes the log of the sum of the exponentiated values $\log\left(\sum_{i=1}^{k} \exp(a_i)\right)$ in a numerically stable way:

$$
\begin{aligned}
&\text{logsumexp}(a_1, ..., a_K)\{ \\
&\quad c \leftarrow \max(a_1, ..., a_K) \\
&\quad \text{Return } \; c + \log\left(\sum_{i=1}^{K} \exp(a_i - c)\right) \\
&\}
\end{aligned}
$$

- Given a tensor A in any number of dimensions D, we can use exactly the same idea to stably compute the log of the sum of exponentials along any dimension d.

Sum-Product Message Passing: Log-Space Algorithm

Assume the mapping $\lambda_k = \log \omega_k$. The messages and beliefs computed below are all computed in log space.

$SumProductMPLS(\{C\}_{1:K}, \{S\}_{1:K-1}, \{\lambda\}_{1:K})$
while (A clique i is ready to send to j) **do**

$$\delta'_{i \to j}(S_{ij}) \leftarrow \log \left(\sum_{C_i / S_{ij}} \left(\exp \left(\lambda_i(C_i) + \sum_{k \in Nb_i / \{j\}} \delta'_{k \to i}(S_{ki}) \right) \right) \right)$$

end while
for k from 1 to K **do**

$$\beta'_i(C_i) = \lambda_i(C_i) + \sum_{k \in Nb_i / \{j\}} \delta'_{k \to i}(S_{ki})$$

end for
return $\{\beta'\}_{1:K}$

Review and Preview

- **Review:** We covered variable elimination, clique trees and sum-product message passing.
- **Preview:** Next class we'll go over the proof of correctness for the sum-product algorithm and discuss learning in Markov networks.