Inference: Exploiting Local Structure

Daphne Koller
Stanford University

CS228 Handout #14

We have seen that BN inference exploits the network structure, in particular the conditional independence and the locality of influence (small number of long-range paths). But when we discussed representation, we also allowed for the representation of finer-grained structure within the CPDs. It turns out that we can exploit that structure as well.

1 Causal Independence

The earliest and simplest instance of exploiting local structure was for CPDs that have independence of causal influence, such as noisy or.

Consider the network in Figure 1(a) representing the node $X$ and its family, and assume that all of the variables are binary. Assume that we want to get a factor over $X$. The operations required to do that (assuming we do the operations as efficiently as possible) is:

- 4 multiplications for $P(Y_1) \times P(Y_2)$
- 8 multiplications for $P(Y_1, Y_2) \times P(Y_3)$
- 16 multiplications for $P(Y_1, Y_2, Y_3) \times P(Y_4)$
- 32 multiplications for $P(Y_1, Y_2, Y_3, Y_4) \times P(X \mid Y_1, Y_2, Y_3, Y_4)$

The total is 60 multiplications, plus another 30 additions to sum out $Y_1, \ldots, Y_4$ (to reduce the resulting factor of size 32 back to a factor of size 2).

If we exploit causal independence and split up the “family” of $X$, we can do a lot better. Recall that we could split up each noisy-or node into a deterministic or of independent noise nodes. Thus, the figure above can be decomposed into the network of Figure 1(b). This, by itself, is not very helpful. The factor $P(X \mid Z_1, Z_2, Z_3, Z_4)$ is still of size 32 if we represent it as a full factor, so we haven’t saved anything.

![Figure 1: (a) A very simple BN. (b) The noisy-or decomposition of the family.](image-url)
Figure 2: Two possible cascaded decompositions for a noisy-or node

The key idea is that the deterministic OR node can be decomposed into various cascades of smaller deterministic OR nodes. There are two main ways to do the split, shown in Figure 2.

In the first decomposition, the optimal ordering for variable elimination is $Y_1, O_1, Y_2, O_2, Y_3, O_3, Y_4$. The cost is 4 multiplications and 2 additions to eliminate each $Y_i$ (multiplying it with the factor for $O_i$ and summing out), and 8 multiplications and 4 additions to eliminate each $O_i$. The total cost is $4 + 8 + 8 + 4 + 8 + 4 + 4 = 40$ multiplications and $2 + 4 + 2 + 4 + 2 + 4 + 2 = 20$ additions.

In the second decomposition, an optimal elimination ordering is $Y_1, Y_2, Y_3, Y_4, O_1, O_2$. We need 8 multiplications and 4 additions to eliminate each of $Y_1, Y_3$, and $O_1$; we need 4 multiplications and 2 additions to eliminate each of $Y_2, Y_4$, and $O_2$. The total is 36 multiplications and 18 additions.

We can generalize this phenomenon to more complex networks and other types of CPDs that have independence of causal influence. We take a node whose CPD has independence of causal influence, and generate its decomposition into a set of independent noise models and a deterministic function. We then cascade the computation of the deterministic function into a bunch of smaller steps. (This can’t always be done, but it can be done in most practical cases.) We do this for every node of this type in the network, and let the variable elimination (or clique tree) algorithm find a good elimination ordering in the decomposed network with the smaller factors. As we saw, the complexity of the inference can go down substantially if we have smaller CPDs.

2 Context specific independence

It turns out that we can use a construction that is very similar in spirit, although different in detail, to decompose CPD-trees into network fragments involving smaller families. Thus, the same benefits for inference also arise in this case.

In order to understand the transformation, first consider a generic node $X$ in a Bayesian network. Let $A$ be one of $X$’s parents, and let $B_1, \ldots, B_k$ be the remaining parents. Assume, for simplicity, that $X$ and $A$ are both binary-valued. Intuitively, we can view the value of the random variable $X$ as the outcome of two conditional variables: the value that $X$ would take if $A$ were true, and the value that $X$ would take if $A$ were false. We can conduct a thought experiment where these two variables are decided separately, and then, when the value of $A$ is revealed, the appropriate value for $X$ is chosen.

Formally, we define a random variable $X_{a^1}$, with a conditional distribution that depends only on $B_1, \ldots, B_k$:

$$P(X_{a^1} \mid B_1, \ldots, B_k) = P(X \mid a^1, B_1, \ldots, B_k).$$

We can similarly define a variable $X_{\bar{a}}$. The variable $X$ is equal to $X_{a^1}$ if $A = a^1$ and is equal
Figure 3: (a) A simple decomposition of the node $X$; (b) A more effective decomposition of $X$, utilizing CSI.

Figure 4: (a) An example CPD-tree. (b) A decomposition of the CPD-tree to $X_{a^0}$ if $A = a^0$. Note that the variables $X_{a^1}$ and $X_{a^0}$ both have the same set of values as $X$. This perspective allows us to replace the node $X$ in any network with the subnetwork illustrated in Figure 3(a). The node $X$ is a deterministic node, which we call a \textit{multiplexer node}, since it takes either the value of $X_{a^1}$ or of $X_{a^0}$, depending on the value of $A$.

For a generic node $X$, this decomposition is not particularly useful. For one thing, the total size of the two new CPDs is exactly the same as the size of the original CPD for $X$; for another, the resulting structure (with its many tightly-coupled cycles) does not admit a more effective decompositions into cliques. However, if $X$ exhibits a significant amount of CSI, this type of transformation can result in a far more compact representation. For example, let $k = 4$, and assume that $X$ depends only on $B_1$ and $B_2$ when $A$ is true, and only on $B_3$ and $B_4$ when $A$ is false. Then each of $X_{a^1}$ and $X_{a^0}$ will have only two parents, as in Figure 3(b). If these variables are binary, the new representation requires two CPDs with four entries each, plus a single deterministic multiplexer node with 8 (predetermined) ‘distributions’. By contrast, the original representation of $X$ had a single CPD with 32 entries. Furthermore, the structure of the resulting network may well allow the construction of a join tree with much smaller cliques.
We can extend this transformation to arbitrary CPD-trees by applying the decomposition recursively. Essentially, each node $X$ is first decomposed according to the parent $A$ which is at the root of its CPT tree. Each of the conditional nodes ($X_{a^1}$ and $X_{a^0}$ in the binary case) has, as its CPD, one of the subtrees of the root in the CPD-tree for $X$ ($X_{a^1}$ will have the subtree corresponding to the $a^1$ child of the root). The resulting conditional nodes can be decomposed recursively, in a similar fashion. Consider, for example, the CPD-tree of Figure 4(a). There, the node corresponding to $X_{a^0}$ can be decomposed into $X_{a^0, b^1}$ and $X_{a^0, b^0}$. The node $X_{a^0, b^0}$ can then be decomposed into $X_{a^0, b^0, c}$ and $X_{a^0, b^0, c^0}$. The resulting network fragment is shown in Figure 4(b).

It is clear that this procedure is beneficial only if there is a structure in the CPD of a node. Thus, in general, we want to stop the decomposition when the CPD of a node is a full tree. (Note that this includes leaves as a special case.)

As in the structural transformation for noisy-or nodes, this decomposition can allow variable elimination or clustering algorithms to form smaller clusters. After the transformation, there are many more nodes in the network, but each generally has far fewer parents, as shown in our example.

Note that the graphical structure of our (transformed) BN cannot capture all independencies implicit in the CPTs. In particular, none of the CSI relations—induced by particular value assignments—can be read from the transformed structure. These CSI relations are captured by the deterministic relationships used in the transformation: in a multiplexer node, the value depends only on one parent once the value of the “selecting” parent (the original variable) is known.

3 Continuous variables

As we discussed, we can represent hybrid BNs with discrete and continuous variables. Can we do inference in such networks? For compact representations of discrete CPDs, this was not an issue. We could always resort to generating explicit CPTs and using standard inference algorithms. For continuous variables, the issue is not so simple.

At some level, the principle of the computation is unchanged. The joint distribution is represented as a product of density functions, each of which depends only on a few variables. We can marginalize out the irrelevant variables by integrating them out, instead of summing them out. Therefore, we simply replace summation with integration, and everything continues to hold. In particular, we can take irrelevant factors out of the integral, in exactly the same way as we did for summations. Indeed, the entire formulation of the clique tree algorithm was based only on qualitative properties of conditional independence, so it continues to hold without change.

The main problem is that the “factors” that we now have to deal with — the CPDs, the intermediate results, the messages — are now continuous or hybrid (continuous/discrete) functions. It is not clear how these functions can be represented compactly. Unfortunately, it turns out that very few families of continuous functions continue to have a nice compact closed form when subjected to multiplication, marginalization, and conditioning. Rather, as we do the inference process — multiplying these functions, eliminating variables — the functions become more and more complicated. These are usually complex enough that we cannot represent them in closed form.

The main exception is Gaussians. The product of two Gaussians is a Gaussian. Marginalizing out a Gaussian results in a Gaussian over the other dimensions. Even conditioning works. In fact, we can do all these operations in closed form, maintaining only the mean and covariance matrices for the multivariate Gaussians. Thus, for networks consisting of linear Gaussian relationships, our clique tree algorithm (or variable elimination) continues to work. Sadly, that’s not very interesting, since there is a well-known simple algorithm that works for purely Gaussian networks: Since a
Gaussian BN is simply a multi-variate Gaussian, the entire joint distribution has a nice compact representation using a mean and a covariance matrix. One easy and fairly efficient way to do inference in a Gaussian network is to generate this representation of the joint and then marginalize it and condition it directly.

However, the clique tree algorithm is useful in another class of networks: one where all CPDs are either discrete or Conditional Linear Gaussian. It turns out that for conditional Gaussian networks, an extension of the clique tree algorithm works in some sense. To understand the sense in which this algorithm works, consider the network shown in Figure 5. The marginal distribution over $X_4$ is a mixture of univariate Gaussians, with one mixture component for every assignment of values to $D_1, D_2, D_3, D_4$. In general, if we continue this construction further, we can see that the number of mixture components for $X_n$ would grow exponentially with $n$. Thus, we cannot represent the distribution in finite closed form. However, it turns out that we can use a trick where we collapse the mixture components into a single Gaussian. In other words, as we are doing variable elimination, if we end up with a distribution over some set of continuous variables which is a mixture, we approximate it with a single Gaussian over that set, and continue our inference process with that simpler density. If we are very careful about the elimination ordering (we eliminate all continuous variables before we eliminate any of the discrete ones), we can show that the resulting distribution has the right mean and covariance. Note that it is a very poor approximation overall: it is unimodal instead of multimodal. However, if our only goal is to compute the mean and variance of one of the continuous random variables, this algorithm can do that exactly.