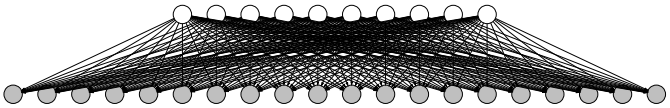
Lecture 8.2: Bayesian networks II

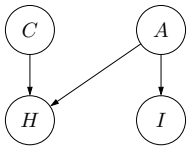


Bayesian networks

Note: this lecture will be done on whiteboard. These are related slides from last year

- Last time, we talked about Bayesian networks, which was a fun and convenient modeling framework. We posit a collection of variables that describe the state of the world, and then create a story on how the variables are generated (recall the probabilistic program interpretation).
- A Bayesian network specifies two parts: (i) a graph structure which governs the qualitative relationship between the variables, and (ii) local conditional distributions, which specify the quantitative relationship.
- Formally, a Bayesian network defines a **joint** probability distribution over many variables (e.g., $\mathbb{P}(C, A, H, I)$) via the **local** conditional distributions (e.g., $p(i \mid a)$). This joint distribution specifies all the information we know about how the world works.

Review: Bayesian network



$$\mathbb{P}(C = c, A = a, H = h, I = i) \stackrel{\text{def}}{=} p(c)p(a)p(h \mid c, a)p(i \mid a)$$

Definition: Bayesian network

Let $X = (X_1, \ldots, X_n)$ be random variables.

A **Bayesian network** is a directed acyclic graph (DAG) that specifies a joint distribution over X as a product of local conditional distributions, one for each node:

$$\mathbb{P}(X_1 = x_1, \ldots, X_n = x_n) = \prod_{i=1}^{n} p(x_i \mid x_{\text{Parents}(i)})$$

Roadmap

Inference

Forward-backward

Review: probabilistic inference

Input

Bayesian network: $\mathbb{P}(X_1 = x_1, \ldots, X_n = x_n)$

Evidence: $E = e$ where $E \subseteq X$ is subset of variables

Query: $Q \subseteq X$ is subset of variables

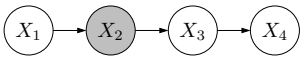
Output

$\mathbb{P}(Q = q \mid E = e)$ for all values q

Example: if coughing and itchy eyes, have a cold?

$$\mathbb{P}(C \mid H = 1, I = 1)$$

Example: Markov model



Query: $\mathbb{P}(X_3 = x_3 \mid X_2 = 5)$ for all x_3

Tedious way:

$$\propto \sum_{x_1, x_4} p(x_1)p(x_2 = 5 \mid x_1)p(x_3 \mid x_2 = 5)p(x_4 \mid x_3)$$

$$\propto \left(\sum_{x_1} p(x_1)p(x_2 = 5 \mid x_1)\right) p(x_3 \mid x_2 = 5)$$

$$\propto p(x_3 \mid x_2 = 5)$$

Fast way:

[whiteboard]

- Let's first compute the query the old-fashioned way by grinding through the algebra. Then we'll see a faster, more graphical way, of doing this.
- We start by transforming the query into an expression that references the joint distribution, which allows us to rewrite as the product of the local conditional probabilities. To do this, we invoke the definition of marginal and conditional probability.
- One convenient shortcut we will take is make use of the proportional-to ($\propto$) relation. Note that in the end, we need to construct a distribution over X_3. This means that any quantity (such as $\mathbb{P}(X_2 = 5)$) which doesn't depend on X_3 can be folded into the proportionality constant. If you don't believe this, keep it around to convince yourself that it doesn't matter. Using $\propto$ can save you a lot of work.
- Next, we do some algebra to push the summations inside. We notice that $\sum_{x_4} p(x_4 \mid x_3) = 1$ because it's a local conditional distribution. The factor $\sum_{x_1} p(x_1)p(x_2 = 5 \mid x_1)$ can also be folded into the proportionality constant.
- The final result is $p(x_3 \mid x_2 = 5)$, which matches the query as we expected by the consistency of local conditional distributions.

General strategy

Query:

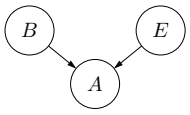
$$\mathbb{P}(Q \mid E = e)$$

Algorithm: general probabilistic inference strategy

- Remove (marginalize) variables that are not ancestors of Q or E.
- Convert Bayesian network to factor graph.
- Condition on $E = e$ (shade nodes + disconnect).
- Remove (marginalize) nodes disconnected from Q.
- Run probabilistic inference algorithm (manual, variable elimination, Gibbs sampling, particle filtering).

- Our goal is to compute the conditional distribution over the query variables $Q \subseteq H$ given evidence $E = e$. We can do this with our bare hands by chugging through all the algebra starting with the definition of marginal and conditional probability, but there is an easier way to do this that exploits the structure of the Bayesian network.
- Step 1: remove variables which are not ancestors of Q or E. Intuitively, these don't have an influence on Q and E, so they can be removed. Mathematically, this is due to the consistency of sub-Bayesian networks.
- Step 2: turn this Bayesian network into a factor graph by simply introducing one factor per node which is connected to that node and its parents. It's important to include all the parents and the child into one factor, not separate factors. From here out, all we need to think about is factor graphs.
- Step 3: condition on the evidence variables. Recall that conditioning on nodes in a factor graph shades them in, and as a graph operation, rips out those variables from the graph.
- Step 4: remove nodes which are not connected to Q. These are independent of Q, so they have no impact on the results.
- Step 5: Finally, run a standard probabilistic inference algorithm on the reduced factor graph. We'll do this manually for now using variable elimination. Later we'll see automatic methods for doing this.

Example: alarm



b	$p(b)$
1	ϵ
0	$1-\epsilon$

e	$p(e)$
1	ϵ
0	$1-\epsilon$

b	e	a	$p(a \mid b, e)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

[whiteboard]

Query: $\mathbb{P}(B)$

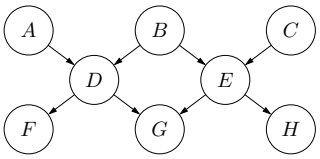
- Marginalize out A, E

Query: $\mathbb{P}(B \mid A = 1)$

- Condition on $A = 1$

- Here is another example: the simple v-structured alarm network from last time.
- $\mathbb{P}(B) = p(b)$ trivially after marginalizing out A and E (step 1).
- For $\mathbb{P}(B \mid A = 1)$, step 1 doesn't do anything. Conditioning (step 3) creates a factor graph with factors $p(b)$, $p(e)$, and $p(a = 1 \mid b, e)$. In step 5, we eliminate E by replacing it and its incident factors with a new factor $f(b) = \sum_e p(e)p(a = 1 \mid b, e)$. Then, we multiply all the factors (which should only be unary factors on the query variable B) and normalize: $\mathbb{P}(B = b \mid A = 1) \propto p(b)f(b)$.
- To flesh this out, for $b = 1$, we have $\epsilon(\epsilon + (1 - \epsilon)) = \epsilon$. For $b = 0$, we have $(1 - \epsilon)(\epsilon + 0) = \epsilon(1 - \epsilon)$. The normalized result is thus $\mathbb{P}(B = 1 \mid A = 1) = \frac{\epsilon}{\epsilon + \epsilon(1-\epsilon)} = \frac{1}{2-\epsilon}$.
- For a probabilistic interpretation, note that all we've done is calculate $\mathbb{P}(B = b \mid A = 1) = \frac{\mathbb{P}(B=b)\mathbb{P}(A=1 \mid B=b)}{\mathbb{P}(A=1)} = \frac{p(b)f(b)}{\sum_{b_i \in \text{Domain}(B)} p(b_i)f(b_i)}$, where the first equality follows from Bayes' rule and the second follows from the fact that the local conditional distributions are the true conditional distributions. The Bayesian network has simply given us a methodical, algorithmic way to calculate this probability.

Example: A-H (section)



[whiteboard]

Query: $\mathbb{P}(C \mid B = b)$

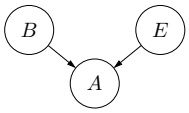
- Marginalize out everything else, note $C \perp\!\!\!\perp B$

Query: $\mathbb{P}(C, H \mid E = e)$

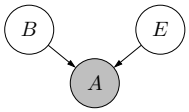
- Marginalize out A, D, F, G, note $C \perp\!\!\!\perp H \mid E$

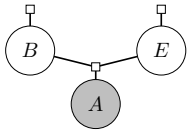
- In the first example, once we marginalize out all variables we can, we are left with C and B, which are disconnected. We condition on B, which just removes that node, and so we're just left with $\mathbb{P}(C) = p(c)$, as expected.
- In the second example, note that the two query variables are independent, so we can compute them separately. The result is $\mathbb{P}(C = c, H = h \mid E = e) \propto p(c)p(h \mid e) \sum_b p(b)p(e \mid b, c)$.
- If we had the actual values of these probabilities, we can compute these quantities.

Pattern 1: v-structure



Parents B and E are **conditionally dependent** (condition on A):

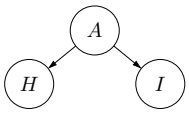




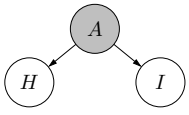
Parents B and E are **independent** (marginalize out A):

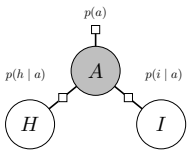
- For factor graphs, independence and conditional independence were straightforward graph properties. Two (sets of) variables B and E were conditionally independent given A if there was no path from B to E in the graph.
- Conditional independence in Bayesian networks warrant some discussion. First, suppose we condition on A. Are B and E independent? The answer is no. A common mistake is to look at the DAG and conclude that A separates B and E. It is important to look at the factor graph rather than the DAG. Recall that the factor graph corresponding to a Bayesian network includes a single factor that depends on both A and its two parents B and E. When we condition on A, that only removes A from the picture, but B and E still have a factor in common. This is intuitive: recall that given $A = 1$, E explained away B.
- Now let's not condition on A but ask whether B and E are independent? Here, we have to eliminate A. But remember that for Bayesian networks, eliminating variables that are not ancestors of the query variables simply amounts to removing them: the new factor produced is $f(b, e) = \sum_a p(a \mid b, e) = 1$, which can be safely ignored. As a result, B and E are actually independent. This is a special property of Bayesian networks: **eliminating children renders parents independent**. This is also intuitive if we think about the interpretation of Bayesian networks: Without any information (such as A), the parents B and E have nothing to do with each other (and thus are independent).
- Note: this three-node Bayesian network is called a **v-structure**. If two variables A, B are Bayesian network conditionally independent given C, then they are said to be **d-separated**.

Pattern 2: inverted v-structure

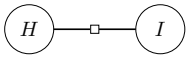


Children H and I are **conditionally independent** (condition on A):



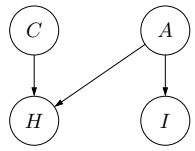


Children H and I are **dependent** (marginalize out A):



- The other structure to keep in mind is the inverted v-structure. Do not get these two confused! While the two structures look graphically similar, if you pause a moment to think about the semantics of Bayesian networks, you'll realize that the two are quite different. First, rather than having one ternary factor $p(a \mid b, e)$ and two unary factors, we now have two binary factors, $p(h \mid a)$ and $p(i \mid a)$, and one unary factor.
- In fact, the inverted v-structure is nothing special: all the independences follow from its factor graph structure.

Independence: examples



$C \perp\!\!\!\perp A$? yes	$C \perp\!\!\!\perp A \mid H$? no
$C \perp\!\!\!\perp I$? yes	$C \perp\!\!\!\perp I \mid H$? no
$C \perp\!\!\!\perp H$? no	$A \perp\!\!\!\perp I \mid H$? no
$A \perp\!\!\!\perp I$? no	$C \perp\!\!\!\perp H \mid A$? no
$A \perp\!\!\!\perp H$? no	$C \perp\!\!\!\perp I \mid A$? yes
$I \perp\!\!\!\perp H$? no	$I \perp\!\!\!\perp H \mid A$? yes

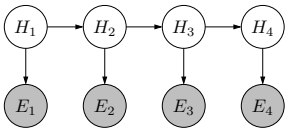
- Now let us look at the medical diagosis example from earlier, which combines both patterns.
- Which of these are Bayesian network independent? Go through each of these and convince yourself of the answer.
- First, marginalize out all the variables that do not have any descendants that we're conditioning on. This just means dropping those nodes by consistency properties.
- Next, we can either check the two patterns directly for a small enough factor graph, or else we convert the Bayesian network to a factor graph, where independence checking is easy (graph connectivity).

Roadmap

Inference

Forward-backward

Object tracking

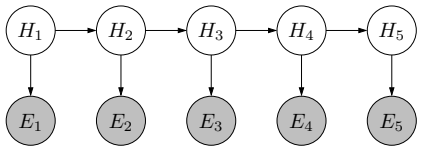


Problem: object tracking

$H_i \in \{1, \ldots, K\}$: location of object at time step i
$E_i \in \{1, \ldots, K\}$: sensor reading at time step i
Start: $p(h_1)$: uniform over all locations
Transition $p(h_i \mid h_{i-1})$: uniform over adjacent loc.
Emission $p(e_i \mid h_i)$: uniform over adjacent loc.

- So far, we have computed various ad-hoc probabilistic queries on ad-hoc Bayesian networks by hand. We will now switch gears and focus on the popular hidden Markov model (HMM), and show how the forward-backward algorithm can be used to compute typical queries of interest.
- As motivation, consider the problem of tracking an object. The probabilistic story is as follows: An object starts at H_1 uniformly drawn over all possible locations. Then at each time step thereafter, it **transitions** to an adjacent location with equal probability. For example, if $H_2 = 3$, then $H_3 \in \{2, 4\}$ with equal probability. At each time step, we obtain a sensor reading E_i which is also uniform over locations adjacent to H_i.

Hidden Markov model



$$\mathbb{P}(H = h, E = e) = \underbrace{p(h_1)}_{\text{start}} \prod_{i=2}^{n} \underbrace{p(h_i \mid h_{i-1})}_{\text{transition}} \prod_{i=1}^{n} \underbrace{p(e_i \mid h_i)}_{\text{emission}}$$

Query (**filtering**):

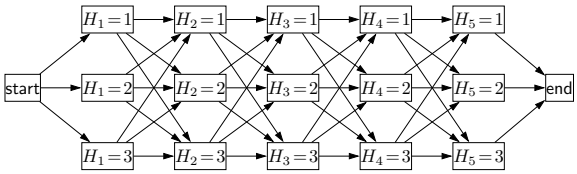
$$\mathbb{P}(H_3 \mid E_1 = e_1, E_2 = e_2, E_3 = e_3)$$

Query (**smoothing**):

$$\mathbb{P}(H_3 \mid E_1 = e_1, E_2 = e_2, E_3 = e_3, E_4 = e_4, E_5 = e_5)$$

- In principle, you could ask any type of query on an HMM, but there are two common ones: filtering and smoothing.
- Filtering asks for the distribution of some hidden variable H_i conditioned on only the evidence up until that point. This is useful when you're doing real-time object tracking, and you can't see the future.
- Smoothing asks for the distribution of some hidden variable H_i conditioned on all the evidence, including the future. This is useful when you have collected all the data and want to retroactively go and figure out what H_i was.

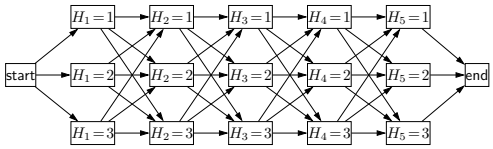
Lattice representation



- Edge $\boxed{\text{start}} \Rightarrow \boxed{H_1 = h_1}$ has weight $p(h_1)p(e_1 \mid h_1)$
- Edge $\boxed{H_{i-1} = h_{i-1}} \Rightarrow \boxed{H_i = h_i}$ has weight $p(h_i \mid h_{i-1})p(e_i \mid h_i)$
- Each path from $\boxed{\text{start}}$ to $\boxed{\text{end}}$ is an assignment with weight equal to the product of node/edge weights

- Now let's actually compute these queries. We will do smoothing first. Filtering is a special case: if we're asking for H_i given $E_1, \ldots, E_i$, then we can marginalize out the future, reducing the problem to a smaller HMM.
- A useful way to think about inference is returning to state-based models. Consider a graph with a start node, an end node, and a node for each assignment of a value to a variable $H_i = v$. The nodes are arranged in a lattice, where each column corresponds to one variable H_i and each row corresponds to a particular value v. Each path from the start to the end corresponds exactly to a complete assignment to the nodes.
- Note that in the reduction from a variable-based model to a state-based model, we have committed to an ordering of the variables.
- Each edge has a weight (a single number) determined by the local conditional probabilities (more generally, the factors in a factor graph). For each edge into $\boxed{H_i = h_i}$, we multiply by the transition probability into h_i and emission probability $p(e_i \mid h_i)$. This defines a weight for each path (assignment) in the graph equal to the joint probability $P(H = h, E = e)$.
- Note that the lattice contains $O(Kn)$ nodes and $O(K^2 n)$ edges, where n is the number of variables and K is the number of values in the domain of each variable.

Lattice representation



Forward: $F_i(h_i) = \sum_{h_{i-1}} F_{i-1}(h_{i-1}) w(h_{i-1}, h_i)$

sum of weights of paths from $\boxed{\text{start}}$ to $\boxed{H_i = h_i}$

Backward: $B_i(h_i) = \sum_{h_{i+1}} B_{i+1}(h_{i+1}) w(h_i, h_{i+1})$

sum of weights of paths from $\boxed{H_i = h_i}$ to $\boxed{\text{end}}$

Define $S_i(h_i) = F_i(h_i) B_i(h_i)$:

sum of weights of paths from $\boxed{\text{start}}$ to $\boxed{\text{end}}$ through $\boxed{H_i = h_i}$

- The point of bringing back the search-based view is that we can cast the probability queries we care about in terms of sums over paths, and effectively use dynamic programming.
- First, define the forward message $F_i(v)$ to be the sum of the weights over all paths from the start node to $\boxed{H_i = v}$. This can be defined recursively: any path that goes $\boxed{H_i = h_i}$ will have to go through some $\boxed{H_{i-1} = h_{i-1}}$, so we can so over all possible values of h_{i-1}.
- Analogously, let the backward message $B_i(v)$ be the sum of the weights over all paths from $\boxed{H_i = v}$ to the end node.
- Finally, define $S_i(v)$ to be the sum of the weights over all paths from the start node to the end node that pass through the intermediate node $\boxed{X_i = v}$. This quantity is just the product of the weights of paths going into $\boxed{H_i = h_i}$ ($F_i(h_i)$) and those leaving it ($B_i(h_i)$).

Lattice representation

Smoothing queries (marginals):

$$\mathbb{P}(H_i = h_i \mid E = e) \propto S_i(h_i)$$

Algorithm: forward-backward algorithm

Compute $F_1, F_2, \ldots, F_L$
Compute $B_L, B_{L-1}, \ldots, B_1$
Compute S_i for each i and normalize

Running time: $O(LK^2)$

- Let us go back to the smoothing queries: $\mathbb{P}(H_i = h_i \mid E = e)$. This is just gotten by normalizing S_i.
- The algorithm is thus as follows: for each node $\boxed{H_i = h_i}$, we compute three numbers: $F_i(h_i), B_i(h_i), S_i(h_i)$. First, we sweep forward to compute all the F_i's recursively. At the same time, we sweep backward to compute all the B_i's recursively. Then we compute S_i by pointwise multiplication.
- Implementation note: we technically can normalize S_i to get $\mathbb{P}(H_i \mid E = e)$ at the very end but it's useful to normalize F_i and B_i at each step to avoid underflow. In addition, normalization of the forward messages yields $\mathbb{P}(H_i = v \mid E_1 = e_1, \ldots, E_i = e_i) \propto F_i(v)$.

Summary

- Lattice representation: paths are assignments (think state-based models)
- Dynamic programming: compute sums efficiently
- Forward-backward algorithm: share intermediate computations across different queries